UNIVERSIDADE FEDERAL DE JUIZ DE FORA
FACULDADE DE ENGENHARIA
GRADUAÇÃO EM ENGENHARIA COMPUTACIONAL

Guilherme Machado Farache Silva

MESHTOOLS: UM SOLUCIONADOR DE ELEMENTOS FINITOS E MANIPULADOR DE MALHAS EM C++

Juiz de Fora

2023

Guilherme Machado Farache Silva

MESHTOOLS: UM SOLUCIONADOR DE ELEMENTOS FINITOS E MANIPULADOR DE MALHAS EM C++

Trabalho de conclusão de curso apresentado apresentada ao Graduação em Engenharia Computacional da Universidade Federal de Juiz de Fora como requisito parcial à obtenção do grau de bacharel em Engenharia Computacional.

Orientador: Prof. Dr. José Jerônimo Camata

Juiz de Fora

2023

Ficha catalográfica elaborada através do Modelo Latex do CDC da UFJF
com os dados fornecidos pelo(a) autor(a)

Silva, Guilherme.
MESHTOOLS: UM SOLUCIONADOR DE ELEMENTOS FINITOS E MANIPULADOR DE MALHAS EM C++ / Guilherme Machado Farache Silva. – 2023.
74 f. : il.

Orientador: José Jerônimo Camata
Trabalho de Conclusão de Curso (graduação) – Universidade Federal de Juiz de Fora, Faculdade de Engenharia. Graduação em Engenharia Computacional, 2023.

1. Método dos Elementos Finitos. 2. Computação de Alto Desempenho. 3. Biblioteca de código aberto. I. Camata, José, orient. II. Título.

Guilherme Machado Farache Silva

MESHTOOLS: UM SOLUCIONADOR DE ELEMENTOS FINITOS E MANIPULADOR DE MALHAS EM C++

Trabalho de conclusão de curso apresentado apresentada ao Graduação em Engenharia Computacional da Universidade Federal de Juiz de Fora como requisito parcial à obtenção do grau de bacharel em Engenharia Computacional.

Aprovada em 13 de Dezembro de 2023

BANCA EXAMINADORA

Prof. Dr. José Jerônimo Camata - Orientador
Universidade Federal de Juiz de Fora

Prof. Dr. Bernardo Martins Rocha
Universidade Federal de Juiz de Fora

Prof. Dr. Ruy Freitas Reis
Universidade Federal de Juiz de Fora

Dedico este trabalho às pessoas que me inspiraram em algum momento e continuam inspirando a querer buscar mais conhecimento. Familiares, amigos e professores.

AGRADECIMENTOS

Agradeço primeiramente à minha família pela confiança, motivação e suporte, no sentido mais amplo da palavra, durante todo o processo da graduação. Sem todo o apoio que vocês me deram, e continuam dando, eu não seria capaz de realizar essa conquista. Estudar em uma cidade, até então, desconhecida não é uma tarefa fácil. Ao mesmo tempo, saber que existe um lugar em que a sua família te espera, sempre de braços abertos, ameniza esta dificuldade.

Agradeço aos amigos que fiz em Juiz de Fora, durante a graduação, por fazerem essa caminhada mais leve, prazerosa e alegre. Feliz é quem compartilha uma risada. Com vocês eu pude me sentir em casa, pude desabafar e celebrar. Agradeço também a vocês por compartilhar dos estudos comigo e pelo tanto que me ensinaram nos momentos em que eu mais necessitei. Também, não poderia deixar de agradecer ao meu amigo de infância, também Friburguense, e também Guilherme, com quem tive o prazer de compartilhar o tempo e conhecimento em disciplinas, trabalhos e apresentações agora no final da graduação.

Agradeço à Luana, minha amada noiva, desde sempre me apoiou e que me impulsionou a buscar sempre o melhor. Como falei, mudar para uma cidade em que todos são desconhecidos, não é tarefa fácil. Porém, por causa de você, da sua companhia, eu tive forças para fazer e serei sempre grato. Você me deu suporte nos momentos mais difíceis quando necessitei de conforto e comemorou intensamente nos momentos das minhas vitórias. Sou grato por compartilhar a vida com você. Eu te amo. Você é inspiração.

Agradeço também a todos os professores que passaram pela minha vida compartilhando humildemente dos seus conhecimentos. Esses agradecimentos vão especialmente aos professores que me inspiraram, em conversas ou mesmo durante suas aulas, me mostrando que estava seguindo por um caminho que me desperta interesse e curiosidade. Ou seja, um caminho que vale a pena trilhar.

Agradeço ao professor José Jerônimo Camata com quem tive o prazer de aprender o básico da programação, no primeiro período da graduação, e um prazer ainda maior de continuar aprendendo com ele desde então, através dos projetos de Iniciação Científica que culminaram na realização deste trabalho. Estes projetos moldaram meu interesse acadêmico e, novamente, demonstraram que o caminho valia a pena ser trilhado. Agradeço imensamente por todo conhecimento compartilhado, sempre da forma mais elegante e amigável. Você é um desses professores que inspiram e sinto um orgulho enorme de ter tido você como meu orientador.

"A very intelligent student, he knew enough to ask questions." Percy W. Bridgman citado em (11)

"Imagination is more important than knowledge. For knowledge is limited to all we now know and understand, while imagination embraces the entire world, and all there ever will be to know and understand." Einstein, A.

> "The desire to economize time and mental effort in arithmetical computations, and to eliminate human liability to error is probably as old as the science of arithmetic itself." Aiken, H. (3)

RESUMO

Diversos fenômenos da natureza são modelados matematicamente por meio de Equações Diferenciais Parciais. Assim, a resolução destes tipos de equações se torna indispensável para a devida modelagem dos diversos problemas das ciências e engenharias. Em alguns casos, a solução analítica destas equações são difíceis ou até mesmo impossíveis de serem obtidas. Dessa forma, a aplicação de métodos numéricos como o Método dos Elementos Finitos é recomendada. Neste método, o domínio contínuo do problema é discretizado em uma malha a qual é composta por nós e elementos. Tendo em vista que a resolução do domínio discretizado, ou seja, o nível de refinamento da malha, influencia na acurácia do método, a utilização de domínios com elemento pequenos é visada a fim de garantir resultados mais precisos. No entanto, domínios mais refinados demandam maior custo computacional na sua solução, tornando-se necessária a utilização de técnicas de alto desempenho como a paralelização do problema para a resolução do mesmo em tempo hábil. Neste trabalho é proposto um arcabouço computacional para manipulação de malhas de elementos finitos e solução destes problemas em ambiente paralelo. Os resultados demonstram uma resolução correta implementada pelo arcabouço e escalabilidade paralela pertinente. Palavras-chave: Método dos Elementos Finitos. Computação de Alto Desempenho. Biblioteca de código aberto.

ABSTRACT

Partial Differential Equations are used to mathematically model several natural phenomena. However, some of these equations are difficult or even impossible to solve analytically. In such cases, numerical methods such as the Finite Element Method are recommended for obtaining accurate solutions. This method discretizes the continuous problem domain into a mesh composed of nodes and elements. The accuracy of the method depends on the resolution of the discretized domain. Therefore, more refined domains lead to more precise results and increase the computational cost required for their resolution. To solve such problems in a timely manner, high-performance techniques like parallelizing the problem become necessary. In this work, a computational framework has been proposed for manipulating finite element meshes and solving these problems in a parallel environment. The framework uses an open source library and the results show that the method is both correct and scalable in a parallel environment. The keywords associated with this work are Finite Element Method, High Performance Computing, and Open Source

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

EDP	Equação Diferencial Parcial
EDO	Equação Diferencial Ordinária
MEF	Método dos Elementos Finitos
CPU	*Central Processing Unit*
GPU	*Graphics Processing Unit*
SUPG	*Streamline Upwind Petrov-Galerkin*
CAU	*Consistent Approximate Upwind*
OCD	Operador de Capture de Descontinuidades
CAD	*Computer Aided Design*
RCM	*Reverse Cuthill McKee*
MPI	*Message Passing Interface*
CSR	*Compressed Sparse Row*
GMRES	*Generalized Minimal Residual Method*
VTK	*Visualization Toolkit*
XML	*Extensible Markup Language*
RAM	*Random-access Memory*

LISTA DE SÍMBOLOS

$\forall$	Para todo
$\in$	Pertence
κ	Taxa de difusividade
∇	Nabla
Ω	Ômega
∂	Del
Γ	Gama
∞	Infinito
$\sum$	Somatório
Δ	Delta maiúsculo
θ	Teta
δ	Delta minúsculo

SUMÁRIO

1 INTRODUÇÃO

Diversos fenômenos naturais são modelados matematicamente por meio de Equações Diferenciais Parciais (EDPs). Enquanto algumas dessas equações são possíveis de serem resolvidas analiticamente via Transformadas de Fourier ou estratégias de integração para Equações Diferenciais Ordinárias (EDOs), outras necessitam da utilização de métodos numéricos. Isto ocorre por conta da dificuldade ou até mesmo pela impossibilidade de obter a solução por meio de métodos analíticos. Nestes casos, a utilização de métodos numéricos como o Método dos Elementos Finitos (MEF) é recomendada (26). No MEF, o domínio contínuo do problema é discretizado em nós e elementos, que definem uma malha de elementos finitos. A solução do problema é calculada nos nós e interpolada nos elementos da malha.

Devido à necessidade de resultados cada vez mais precisos, malhas mais refinadas, ou seja, com maior discretização espacial são imprescindíveis. Entretanto, o processamento dessas malhas em tempo hábil requer o emprego de técnicas que aprimoram tanto a definição da estrutura de armazenamento quanto os algoritmos para sua manipulação.

O advento de sistemas computacionais de alto desempenho possibilitou o desenvolvimento de diversas aplicações que aproveitassem tanto o paralelismo em memória compartilhada quanto o paralelismo em memória distribuída embarcados nestes sistemas. No paralelismo de memória compartilhada, a memória pode ser acessada simultaneamente por diferentes fluxos de execução (*threads*) de uma dada aplicação. Contudo, é essencial contornar a condição de corrida, ou seja, a possibilidade de diferentes *threads* manipularem simultaneamente um mesmo endereço de memória, podendo, assim, causar inconsistências nos cálculos computacionais (37). Já em sistemas com memória distribuída, as estruturas de dados devem ser particionadas e distribuídas entre os processos. Para obter uma solução eficiente, o processo de particionamento deve maximizar o balanceamento de carga e minimizar o custo da comunicação entre os processos.

Nos últimos anos surgiram diversas opções de *softwares* de código-livre para resolver problemas modelados por EDPs através do Método dos Elementos Finitos e com suporte ao paralelismo, como as bibliotecas `libMesh` (31), `deal.II` (7) e `FEniCS` (4). A biblioteca `libMesh` é escrita em linguagem C++ com enfoque em refinamento adaptativo paralelo de malhas não estruturadas arbitrárias (AMR/C). Através desta biblioteca é possível particionar uma malha, utilizando da biblioteca METIS (29) ou ParMETIS (30) e solucionar um problema de elementos finitos fazendo uso das bibliotecas LaPACK (6), PETSc (9) e SLEPc (25) para solução de sistemas lineares e problemas de autovalor. Sua estrutura é definida em classes utilizando da Programação Orientada à Objetos com definições de classes polimórficas para a malha, elementos, nós, graus de liberdade, etc. A utilização constante de polimorfismo é negativa pelo fato desta ser resolvida em tempo de execução

do código, e não em tempo de compilação reduzindo sua eficiência. O paralelismo da biblioteca `libMesh` é concentrado na montagem do sistema linear e na sua solução, devido a isso, visando melhor balanceamento de carga entre os processos concorrentes, a `libMesh` armazena, por padrão, uma cópia da malha completa em cada processo. Através desta biblioteca é possível solucionar sistemas estacionários, transientes, lineares ou não-lineares que são resultantes dos mais variados problemas da engenharia, como aqueles modelados por equações de Navier Stokes, Advecção-difusão-reação, equação de Burgers, entre outras. Portanto, é possível solucionar estes problemas em paralelo fazendo uso da biblioteca `libMesh`.

A biblioteca `deal.II`, também escrita na linguagem C++, procura ofertar todo ferramental necessário para o usuário a fim de resolver uma Equação Diferencial Parcial de forma discreta através do Método dos Elementos Finitos. Também como na `libMesh`, `deal.II` opta por utilizar de bibliotecas terceiras como a PETSc (9), Trilinos (48), cuSPARSE (36) e outras. E, além disso, também opta por utilizar a estratégia de *arrays* de estruturas aumentando a complexidade da representação do problema computacionalmente. Porém, `deal.II` se preocupa em contornar o problema causado pelo uso de excessivo do polimorfismo visto no `libMesh`, aprimorando sua eficiência computacional.

Como na biblioteca `libMesh`, a biblioteca `deal.II` também fornece ferramental para aplicar refinamento adaptativo de malha. A biblioteca `deal.II` disponibiliza diferentes tipos de solução de uma malha de elementos finitos, são estes em série, compartilhado e distribuído. No caso da solução de uma malha compartilhada ou distribuída, ambas são particionadas entre diferentes processos MPI. Para a solução compartilhada, toda malha é distribuída entre os processos MPI aprimorando o uso em métodos que necessitam de toda a malha para todo processo paralelo, como método de elementos de contorno. Porém, esta estratégia se torna inviável para um número elevado de processos quando é mais recomendado utilizar da solução através do método distribuído em que a malha é particionada, distribuída entre os processos e solucionada em paralelo. Estes problemas podem ser paralelizados com `deal.II` tanto em CPUs, quanto em GPUs utilizando de CUDA (39).

Utilizando programação de alto nível de abstração, o *software* `FEniCS` é uma alternativa para as bibliotecas em C++ `libMesh` e `deal.II`. `FEniCS` é um *software* Python com intuito de ser *user-friendly*, ou seja, cuja vantagem reside na maior facilidade de escrita do código de elementos finitos em detrimento às bibliotecas em C++. Nele, a partir de um *script* Python é gerado código em C++ através do componente FEniCS Form Compiler (FFC). A utilização da linguagem Python para escrita do código tem como vantagem a legibilidade do código, porém a desvantagem está na eficiência computacional reduzida em comparação às bibliotecas escritas diretamente em C++. O *software* `FEniCS` consiste em um conjunto de componentes, como o já citado FFC para compilar o código Python em C++, também é utilizado o componente `mshr` para geração de malha de elementos

finitos, além de bibliotecas com classes para computação de elementos finitos como a DOLFIN e bibliotecas de álgebra linear como a PETSc (9), uBLAS (50), Trilinos (48), entre outras. Tendo em vista o problema de desempenho computacional no `FEniCS`, está em desenvolvimento o `FEniCSx`, uma versão voltada para computação de alto desempenho da `FEniCS` com sua versão inicial *alpha* publica em outubro de 2023.

Neste trabalho é apresentado o MeshTools, um software escrito na linguagem C++ para solução de problemas de elementos finitos em paralelo. O MeshTools tem como premissa um meio-termo entre legibilidade de código e eficiência computacional. As bibliotecas `libMesh` e `deal.II` utilizam a ideia de *arrays* de estruturas em que os nós, elementos e outras estruturas da malha tem suas próprias classes. Esta estratégia facilita a legibilidade do código, porém torna a estrutura de representação complexa perdendo em eficiência. Em detrimento às bibliotecas citadas, no MeshTools é utilizada a ideia de estruturas de *arrays* em que é feito o armazenamento de todos os dados de uma mesma categoria em um *array*, como, por exemplo, as coordenadas dos nós da malha, as conectividades dos elementos, etc. Este tipo de estrutura é similar à estrutura de dados VTK (32), o formato utilizado para escrita dos dados brutos resultantes da simulação. Dessa forma a obtenção desses dados é dada de forma direta, aprimorando a eficiência do código. Tendo em vista que, em um problema de elementos finitos é feito o laço nos elementos da malha diversas vezes, a simplificação dessa estrutura é visada objetivando uma maior eficiência na etapa de montagem do sistema.

No MeshTools, a etapa de pré-processamento prepara a malha para o uso de sistemas computacionais híbridos massivamente paralelos. MeshTools faz uso de diferentes esquemas de reordenação nodal para melhoria de localidade dos dados na memória, incorpora a biblioteca estado-da-arte METIS (29) para o particionamento de domínio e também implementa o algoritmo de coloração (22) de grafos para permitir o processamento da malha em ambientes de memória compartilhada. Pelo MeshTools é possível resolver uma Equação Diferencial Parcial através do Método dos Elementos Finitos de forma paralela em sistemas distribuídos e/ou compartilhados. Para isso foi incorporado a biblioteca PETSc (9) a qual disponibiliza diversos métodos de solução para sistemas lineares em paralelo. O problema de Elementos Finitos pode ser solucionado no MeshTools em malhas bidimensionais ou tridimensionais, dando suporte apenas a elementos lineares quandrangulares e triangulares em domínios bidimensionais, e tetraédricos em domínios tridimensionais. Os resultados obtidos através desta etapa podem ser visualizado através da escrita dos dados brutos em formato VTK, ou através do co-processamento da interface de programação do ParaView Catalyst (8). Através desta interface é possível gerar imagens, tabelas, séries temporais ou outras formas de visualizações para o resultado sem a necessidade de escrita do dado bruto.

Este trabalho está organizado da seguinte forma: inicialmente na Seção 2 é introduzido o Método dos Elementos Finitos a fim de abordar a teoria por trás do método

numérico implementado no MeshTools. Em seguida, na Seção 3 é abordada a organização do arcabouço MeshTools com a descrição dos dois componentes principais, o responsável pela manipulação de malhas e o de solução de elementos finitos. Na Seção 4 são apresentados e discutidos os resultados obtidos no MeshTools com a execução de testes padrão que demonstram a escalabilidade paralela e corretude dos resultados na solução do Método de Elementos Finitos, além apresentar os resultados para as técnicas de alto desempenho implementadas. E finalmente, na Seção 5 são apresentadas observações finais e trabalhos futuros.

2 MÉTODO DOS ELEMENTOS FINITOS

Nesta seção, introduziremos as formulações de elementos finitos para os problemas que serão empregados para a avaliação do MeshTools. Duas classes de problemas serão apresentadas: problemas difusivos estacionários e problemas difusivos-advectivos dependentes do tempo.

2.1 FORMULAÇÃO PARA PROBLEMAS DIFUSIVOS ESTACIONÁRIOS

2.1.1 Formulações forte e fraca

Considera-se um problema de difusão de uma quantidade escalar $u = u(\mathbf{x})$ em um domínio $\Omega \in \mathbb{R}^d$, com $d = 2, 3$, dada pela seguinte equação governante

$$-\nabla \cdot \kappa \nabla u = s \quad \text{em } \Omega, \tag{2.1}$$

onde κ é a taxa de difusividade, s o termo fonte e Ω o domínio do problema em $\mathbb{R}^d$ onde d é a dimensão espacial do problema. Considerando κ isotrópico, ou seja, com o valor independente da direção, e constante em todo domínio, a equação (2.1) pode ser reduzida à

$$-\kappa \nabla^2 u = s, \tag{2.2}$$

onde o operador ∇ é definido por

$$\nabla = \mathbf{i}\frac{\partial}{\partial x} + \mathbf{j}\frac{\partial}{\partial y} + \mathbf{k}\frac{\partial}{\partial z}. \tag{2.3}$$

Dessa forma, para o exemplo em questão, chega-se a

$$\nabla^2 u = \mathbf{i}\frac{\partial^2 u}{\partial x^2} + \mathbf{j}\frac{\partial^2 u}{\partial y^2} + \mathbf{k}\frac{\partial^2 u}{\partial z^2} \tag{2.4}$$

a qual é conhecido como gradiente do divergente, ou laplaciano da função u.

Considere o domínio Ω, com contorno suave Γ. O contorno consiste em uma porção Γ_D em que o valor u é prescrito e uma porção complementar Γ_N em que o fluxo difusivo é prescrito de forma que $\Gamma_D \cup \Gamma_N = \Gamma$ e $\Gamma_D \cap \Gamma_N = \emptyset$. Condições em Γ_D são conhecidas como condições de Dirichlet (ou essenciais) e condições em Γ_N são conhecidas como condições de Neumann.

Dessa forma, a formulação forte para o problema com valor de contorno associado à equação de difusão é definida da seguinte forma: encontrar u tal que

$$\begin{cases} -\kappa \nabla^2 u & = s \text{ em } \Omega \\ u & = u_D \text{ em } \Gamma_D \\ \mathbf{n} \cdot \kappa \nabla u & = h \text{ em } \Gamma_N \end{cases}, \tag{2.5}$$

onde u_D o valor da variável de interesse prescrita no contorno de Dirichlet Γ_D, e h o valor do fluxo prescrito no contorno de Neumann Γ_N, e $\mathbf{n}$ o vetor normal à superfície Γ_N.

Para o Método dos Elementos Finitos, é necessário obter a formulação fraca da equação governante para a devida discretização do problema. A formulação fraca nada mais é que uma formulação integral da equação governante. Para a aplicação dessa formulação é necessário definir dois conjuntos de funções: as funções peso e as funções teste.

O conjunto de funções peso são funções que pertencem ao espaço de Hilbert que se anulam na porção do domínio de Dirichlet, Γ_D, denotadas por $\mathcal{V}$. Definido da seguinte forma

$$\mathcal{V} = \{w \mid w \in H^1,\ w = 0 \ \text{ em } \Gamma_D\}. \tag{2.6}$$

O espaço de Hilbert, denotado por H^k, é definido como um espaço de funções quadrado-integráveis onde suas derivadas, até ordem k, também funções quadrado-integráveis. Esse espaço é uma classe particular do espaço de Sobolev. Funções quadrado-integráveis são funções com resultado definido quando seu quadrado é integrado no domínio do problema, ou seja, um resultado não-infinito. Para uma função genérica g é definido como

$$\int_\Omega (g)^2 \, d\Omega < \infty. \tag{2.7}$$

Dessa forma, como o conjunto de funções peso, o conjunto de funções teste também são funções pertencentes ao espaço de Hilbert, porém nesse conjunto é respeitado o valor da condição de contorno na porção do domínio em Γ_D. O conjunto de funções teste é denotado por $\mathcal{S}$ e tem a seguinte definição

$$\mathcal{S} = \{u \mid u \in H^1,\ u = \overline{u} \ \text{ em } \Gamma_D\}. \tag{2.8}$$

A partir de uma equação diferencial, para a obtenção da sua formulação fraca, deve-se multiplicar a equação governante pela função peso $w \in \mathcal{V}$ e aplicar a integração no domínio do problema. No exemplo apresentado anteriormente da equação de difusão, obtém-se sua formulação fraca multiplicando a equação pela função peso e integrando no domínio, como apresentado a seguir

$$-\int_\Omega w \cdot \kappa \nabla^2 u \, d\Omega = \int_\Omega w \cdot s \, d\Omega \quad \forall w \in \mathcal{V}. \tag{2.9}$$

A fim de possibilitar a utilização de funções de aproximação menos suaves, é necessário aplicar o Teorema do Divergente de Gauss. Dessa forma precisa-se obter um termo com divergente. Dessa forma, será aplicado o Teorema do Divergente do Produto, definido por

$$\nabla(f \cdot g) = f \nabla g + g \nabla f. \tag{2.10}$$

Assim, definindo $f = w$ e $g = k\nabla u$, é possível aplicar o Teorema do Divergente do Produto na equação trabalhada. Obtém-se,

$$\nabla(w \cdot \kappa \nabla u) = w \cdot \kappa \nabla^2 u + \kappa \nabla u \cdot \nabla w, \tag{2.11}$$

isolando o termo $w \cdot \kappa\nabla^2 u$ e aplicando a integral em Ω, tem-se

$$\int_\Omega w \cdot \kappa\nabla^2 u \, d\Omega = \int_\Omega \nabla(w \cdot \kappa\nabla u) \, d\Omega - \int_\Omega \kappa\nabla u \cdot \nabla w \, d\Omega. \tag{2.12}$$

Logo, substituindo a Equação (2.12) na Equação (2.9), é concluído que

$$\int_\Omega \kappa\nabla u \cdot \nabla w \, d\Omega - \int_\Omega \nabla(w \cdot \kappa\nabla u) \, d\Omega = \int_\Omega w \cdot s \, d\Omega. \tag{2.13}$$

A partir da Equação (2.13) é possível aplicar o Teorema do Divergente de Gauss no segundo termo, obtendo

$$\int_\Omega \kappa\nabla u \cdot \nabla w \, d\Omega - \int_\Gamma w(\mathbf{n} \cdot \kappa\nabla u) \, d\Gamma = \int_\Omega w \cdot s \, d\Omega. \tag{2.14}$$

Nesta equação é possível utilizar funções de aproximação menos suaves pelo fato de existir somente a derivada de primeira ordem das funções de teste e de forma. Aplicando o valor da condição de Neumann no segundo termo da Equação (2.14) tem-se

$$\int_\Omega \kappa\nabla u \cdot \nabla w \, d\Omega - \int_{\Gamma_N} w \cdot h \, d\Gamma = \int_\Omega w \cdot s \, d\Omega. \tag{2.15}$$

Isolando o termo incógnito temos

$$\int_\Omega \kappa\nabla u \cdot \nabla w \, d\Omega = \int_\Omega w \cdot s \, d\Omega + \int_\Gamma w \cdot h \, d\Gamma. \tag{2.16}$$

A fim de tornar a notação mais leve será adotada a forma compacta de integrais fazendo uso de formas bilineares, como

$$a(u, w) = \int_\Omega \kappa\nabla u \cdot \nabla w \, d\Omega, \tag{2.17}$$

e funcionais lineares como

$$(w, s) = \int_\Omega w \cdot s \, d\Omega \tag{2.18}$$

$$(w, h)_\Gamma = \int_\Gamma w \cdot h \, d\Gamma. \tag{2.19}$$

Dessa forma tem-se a formulação fraca do problema, onde desejamos encontrar $u \in \mathcal{S}$ que satisfaça a Equação (2.20) para todo $w \in \mathcal{V}$.

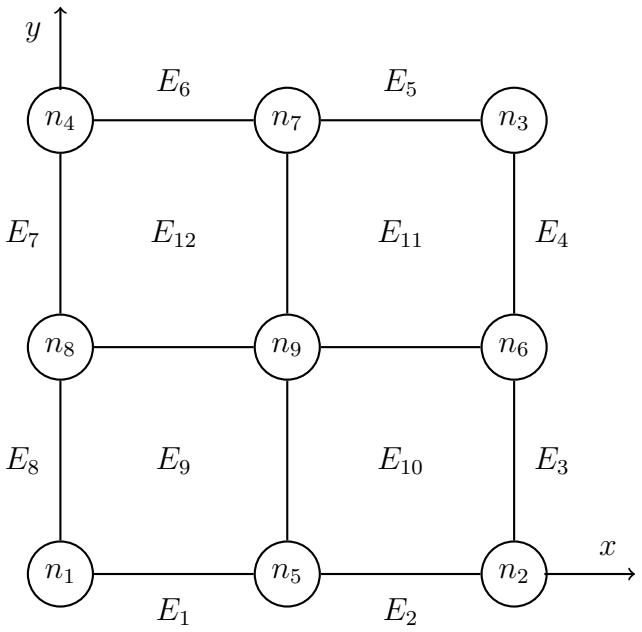
$$a(u, w) = (w, s) + (w, h)_\Gamma. \tag{2.20}$$

2.1.2 Aproximação pelo Método de Galerkin

Pelo Método de Galerkin, o problema contínuo é discretizado em elementos lineares, permitindo a obtenção de um resultado contínuo por partes. Assim, computacionalmente, o domínio é representado através de uma malha, a qual é composta por nós e elementos onde é aplicado o método. A Figura 1 representa um domínio bidimensional quadrado de tamanho 2×2 discretizado em 9 nós e 12 elementos. É definido o conjunto global

de nós $\mathcal{N} = \{n_1, n_2, n_3, n_4, n_5, n_6, n_7, n_8, n_9\}$ e o conjunto de elementos quadrilaterais $\mathcal{E} = \{E_1, E_2, E_3, E_4, E_5, E_6, E_7, E_8, E_9, E_{10}, E_{11}, E_{12}\}$. Além disso, temos o subconjunto $\mathcal{N}_\mathcal{D} \in \mathcal{N}$ de nós que pertencem à porção do domínio de Dirichlet, ou seja, nós não incógnitos, com valor conhecido. Considerando todo contorno do domínio com aplicação de condição do tipo Dirichlet, o conjunto $\mathcal{N}_\mathcal{D} = \{n_1, n_5, n_2, n_6, n_3, n_7, n_4, n_8\}$. O resultado da aproximação por Elementos Finitos é aplicado nos elementos mediante interpolação. Por exemplo, para o elemento E_9 é feita a aproximação nos nós n_1, n_5, n_9 e n_8 e estes valores são interpolados no subdomínio do elemento.

Figura 1 – Exemplo de domínio bidimensional discretizado em 9 nós



Para a aplicação computacional do método, devem ser escolhidas funções de forma para cada nó n do elemento de forma que o valor da função no nó n seja 1 e 0 nos demais. As funções interpoladoras dependem do tipo do elemento utilizado na malha. Esses podem ser quadriláteros ou triangulares para elementos bidimensionais, por exemplo, e tetraédricos ou hexaédricos para elementos tridimensionais.

Dessa forma, para a aplicação do Método de Galerkin, deve-se definir os conjuntos $\mathcal{S}^h$ e $\mathcal{V}^h$, conjuntos de funções em um domínio discretizado de Ω, ou seja, são conjuntos de funções em um espaço finito. Dessa forma, $\mathcal{S}^h$ é um subconjunto de $\mathcal{S}$ e $\mathcal{V}^h$ um subconjunto de $\mathcal{V}$. Além disso, $u^h \in \mathcal{S}^h$ e $w^h \in \mathcal{V}^h$.

Pelo fato do problema necessitar de uma condição de Dirichlet para existir um resultado único, tem-se um conjunto de nós com valores prescritos ($\mathcal{N}_D$). Assim, para a formulação de u^h, tem-se

$$u^h(x) = \sum_{A \in \mathcal{N} - \mathcal{N}_D} N_A(x) u_A + \sum_{A \in \mathcal{N}_D} N_A(x) u_D(x_A), \tag{2.21}$$

onde $N_A(x)$ é a função de forma, u_A o valor nos nós incógnitos e $u_D(x_A)$ é uma função que satisfaz a condição de contorno nos nós da porção do domínio de Dirichlet.

Além disso, a função w^h é dada por

$$w^h = \text{subespaço}\{N_A\}. \tag{2.22}$$

Ou seja, funções que são obtidas através da combinação linear de N_A. Dessa forma, é desejado encontrar $u^h \in \mathcal{S}^h$ tal que obedeça a seguinte equação

$$a(u^h, w^h) = (w^h, s) + (w^h, h)_\Gamma \quad \forall w^h \in \mathcal{V}^h. \tag{2.23}$$

Substituindo a Equação (2.21) e (2.22) na Equação (2.23), tem-se

$$\begin{aligned} \sum_{\mathcal{B}\in\mathcal{N}-\mathcal{N}_\mathcal{D}} a(N_A, N_B)u_B = (N_A, s) \\ +(N_A, h)_\Gamma - \sum_{\mathcal{B}\in\mathcal{N}_\mathcal{D}} a(N_A, N_B)u_D(x_B) \quad \forall A \in \mathcal{N} - \mathcal{N}_\mathcal{D} \end{aligned} \tag{2.24}$$

em que u_B são os únicos termos desconhecidos da equação.

Dessa forma, considerando n o número de nós do elemento, para cada elemento existem n equações e n u_B's, valores desconhecidos. As equações integrais são resolvidas numericamente através da Quadratura Gaussiana, obtendo-se um sistema linear com solução única. O sistema pode então ser representado simplificadamente por $\mathbf{Ku} = \mathbf{f}$, onde

$$\begin{aligned} \mathbf{K} &= \sum_{e=1}^{n_{el}} \mathbf{K}^e, \quad K^e_{ab} = a(N_A, N_B) \\ \mathbf{f} &= \sum_{e=1}^{n_{el}} \mathbf{f}^{\,e}, \quad f^e_a = (N_A, s) + (N_A, h)_\Gamma - a(N_A, N_B)u_D(x_B), \\ \mathbf{u} &= u_B \end{aligned} \tag{2.25}$$

onde e é referente ao elemento processado e os subscritos a e b representam os nós do elemento processado. O modo como esse sistema linear é resolvido será apresentado na Seção 3.2.

2.2 FORMULAÇÃO PARA PROBLEMAS CONVECTIVOS-DIFUSIVOS TRANSIENTES

Os problemas transientes podem ser formulados de duas formas diferentes: a formulação acoplada em tempo-espaço e a discretização semi-discreta. Na formulação acoplada são utilizados elementos com dimensão temporal utilizando funções de forma que variam tanto no espaço quanto no tempo (27). Já na formulação semi-discreta são utilizados elementos que dependem somente do espaço com funções de forma comuns como

as aplicadas nos problemas estacionários. Nesse caso, o termo temporal é tratado pelo Método das Diferenças Finitas, podendo utilizar uma diferença progressiva, regressiva ou uma diferença intermediária como Crank-Nicolson.

Para um problema transiente genérico de convecção-difusão, tem-se a seguinte equação governante

$$\frac{\partial u}{\partial t} + \mathbf{v}\nabla u - \nabla \cdot (\kappa \nabla u) = s \quad \text{em } \Omega \times]0, T_f], \tag{2.26}$$

com u a variável de interesse a ser solucionada, por exemplo, a concentração de um determinado fluido, $\mathbf{v}$ a velocidade de convecção desse fluido, κ o coeficiente de difusão, s o termo fonte, Ω o domínio espacial do problema e T_f o tempo final modelado. Para um problema transiente é necessária a condição inicial no domínio, ou seja, o valor da variável de interesse em todo o domínio para o primeiro passo de tempo. Dessa forma, utilizando condições de contorno genéricas, o problema bem-posto é dado por

$$\begin{cases} \dfrac{\partial u}{\partial t} + \mathbf{v}\nabla u - \nabla \cdot (\kappa \nabla u) = s & \text{em } \Omega \times]0, T_f] \\ u(\,\cdot\,, 0) = u_0 & \text{em } \Omega \\ u = u_D & \text{em } \Gamma_D \\ \mathbf{v}(\mathbf{n} \cdot \nabla) u = h & \text{em } \Gamma_N \end{cases}, \tag{2.27}$$

onde u é a concentração do fluido, u_0 a concentração inicial no domínio, u_D o valor da variável de interesse prescrita no contorno de Dirichlet, $\mathbf{v}$ a velocidade de convecção do fluido, $\mathbf{n}$ o vetor normal à superfície do domínio e h o valor do fluxo de entrada/saída prescrito no contorno de Neumann.

Obtém-se a formulação fraca do problema multiplicando a equação governante pela função peso e integrando no domínio. É desejado encontrar $u \in \mathcal{S}$ para todo $w \in \mathcal{V}$ tal que obedeça a seguinte equação

$$\int_\Omega w \Big(\frac{\partial u}{\partial t} + \mathbf{v}\nabla u - \nabla \cdot (\kappa \nabla u) \Big) \, d\Omega = \int_\Omega w \cdot s \, d\Omega. \tag{2.28}$$

Aplicando o Teorema do Divergente de Gauss no termo difusivo, obtemos

$$\int_\Omega w \cdot \frac{\partial u}{\partial t} \, d\Omega + \int_\Omega w \cdot (\mathbf{v}\nabla u) \, d\Omega + \int_\Omega \nabla w \cdot (\kappa \nabla u) \, d\Omega = \int_\Omega w \cdot s \, d\Omega + \int_\Gamma w \cdot h \, d\Gamma. \tag{2.29}$$

A fim de simplificar a notação será adotado novamente a forma compacta. Dessa forma, a forma trilinear é definida a seguir

$$(\mathbf{v}; w, u) = \int_\Omega w \cdot (\mathbf{v}\nabla u) \, d\Omega. \tag{2.30}$$

Assim, obtém-se da seguinte formulação

$$(w, u_t) + (\mathbf{v}; w, u) + a(u, w) = (w, s) + (w, h)_\Gamma \tag{2.31}$$

Aplicando o Método de Galerkin inserindo o conjunto de funções peso e teste discretizadas

$$(w^h, u_t) + (\mathbf{v}; w^h, u^h) + a(u^h, w^h) = (w^h, s) + (w^h, h)_\Gamma. \tag{2.32}$$

Novamente, substituindo as Equações (2.21) e (2.22) na Equação (2.32), tem-se

$$\begin{aligned} &(N_A, u_t) + \sum_{B \in \mathcal{N} - \mathcal{N}_D} \Big[(\mathbf{v}; N_A, N_B) u_B + a(N_B, N_A) u_B \Big] \\ &= (N_A, s) + (N_A, h)_\Gamma - \sum_{B \in \mathcal{N}_D} \Big[(\mathbf{v}; N_A, N_B) u_D + a(N_B, N_A) u_D \Big] \quad \forall A \in \mathcal{N} \end{aligned} \tag{2.33}$$

reorganizando a equação e deixando o termo u_B em evidência

$$\begin{aligned} &(N_A, u_t) + \sum_{B \in \mathcal{N} - \mathcal{N}_D} \Big[(\mathbf{v}; N_A, N_B) + a(N_B, N_A) \Big] u_B \\ &= (N_A, s) + (N_A, h)_\Gamma - \sum_{B \in \mathcal{N}_D} \Big[(\mathbf{v}; N_A, N_B) u_D + a(N_B, N_A) u_D \Big] \quad \forall A \in \mathcal{N}. \end{aligned} \tag{2.34}$$

A representação matricial é dada por

$$\mathbf{M}\dot{\mathbf{u}} + \Big(\mathbf{C} + \mathbf{K}\Big)\mathbf{u} = \mathbf{f} \tag{2.35}$$

com

$$\begin{aligned} \mathbf{M} &= \sum_{e=1}^{n_{el}} \mathbf{M}^e, \quad M^e_{ab} = (N_A, N_B) \\ \dot{\mathbf{u}} &= \frac{\partial u}{\partial t} \\ \mathbf{C} &= \sum_{e=1}^{n_{el}} \mathbf{C}^e, \quad C^e_{ab} = (\mathbf{v}; N_A, N_B) \\ \mathbf{K} &= \sum_{e=1}^{n_{el}} \mathbf{K}^e, \quad K^e_{ab} = a(N_B, N_A) \\ \mathbf{u} &= u_B \\ \mathbf{f} &= \sum_{e=1}^{n_{el}} \mathbf{f}^e, \quad f^e_a = (N_A, s) + (N_A, h)_\Gamma - \Big[(\mathbf{v}; N_A, N_B) + a(N_B, N_A) \Big] u_D \end{aligned} \tag{2.36}$$

Dessa forma, seguindo o método semi-discreto para o cálculo da derivada temporal e utilizando a formulação de Galerkin para família de métodos θ (46) temos a seguinte aproximação para o termo temporal

$$\frac{\mathbf{u}_i - \mathbf{u}_{i-1}}{\Delta t} = \theta\Big(\frac{\partial \mathbf{u}}{\partial t}\Big)_i + (1-\theta)\Big(\frac{\partial \mathbf{u}}{\partial t}\Big)_{i-1}$$

com as seguintes opções de métodos:

$$\begin{aligned} \theta = 0, &\quad \text{Método de Euler Explícito} \\ \theta = 1/2, &\quad \text{Método de Crank-Nicolson} \\ \theta = 1, &\quad \text{Método de Euler Implícito} \end{aligned}$$

Por fim, aplicando o método θ para o problema estudado é obtido da seguinte equação

$$\begin{aligned} \mathbf{M}\Big(\frac{\mathbf{u}^i - \mathbf{u}^{i-1}}{\Delta t}\Big) + \theta\Big((\mathbf{C}+\mathbf{K})\mathbf{u}^i\Big) + (1-\theta)\Big((\mathbf{C}+\mathbf{K})\mathbf{u}^{i-1}\Big) = \\ \theta\mathbf{f}^i + (1-\theta)\mathbf{f}^{i-1} \end{aligned} \tag{2.37}$$

e, ao isolar os termos em função da incógnita no passo de tempo i tem-se

$$\begin{aligned} \mathbf{M}\mathbf{u}^i + \Delta t \cdot \theta\Big((\mathbf{C}+\mathbf{K})\mathbf{u}^i\Big) = \\ \mathbf{M}\mathbf{u}^{i-1} - \Delta t(1-\theta)\Big((\mathbf{C}+\mathbf{K})\mathbf{u}^{i-1}\Big) + \Delta t(1-\theta)\mathbf{f}^{i-1} + \Delta t \cdot \theta\mathbf{f}^i \end{aligned} \tag{2.38}$$

Problemas do tipo convectivos dominantes, ou seja, problemas em que o termo convectivo tem maior influência que o difusivo no movimento do fluido, devem ser tratados de forma especial devido a oscilações numéricas espúrias encontradas na solução utilizando da formulação de Galerkin apresentada anteriormente. Além da possibilidade de aplicar malhas mais refinadas, é possível adicionar termos estabilizadores na formulação inserindo uma difusão artificial na solução responsável por amenizar as oscilações (12).

2.2.1 Formulação Estabilizada

Devido a oscilações numéricas na resolução do método de Galerkin em problemas convectivos-dominantes é possível aplicar termos estabilizantes na formulação de elementos finitos como: *Streamline Upwind Petrov-Galerkin* (SUPG) detalhado em (12) e *Consistent Approximate Upwind* (CAU) detalhado em (5). Enquanto o termo estabilizador SUPG reduz as oscilações presentes na direção da linha de corrente do fluido, o termo CAU, um Operador de Captura de Descontinuidades (OCD) ameniza as oscilações numéricas presentes nas direções perpendiculares às linhas de corrente devido ao forte gradiente (5). Este termo transforma a formulação estabilizada em uma formulação não-linear.

Com a inserção destes termos estabilizadores, a formulação final para problemas convectivos-difusivos transientes é dada pela seguinte equação

$$\underbrace{\int_{\Omega_e} w^h \Big(\frac{\partial u}{\partial t} + \mathbf{v} \cdot \nabla u^h - \nabla \cdot (\kappa \nabla u^h) \Big) \, d\Omega_e}_{\text{Galerkin}} + \underbrace{\sum_e \int_{\Omega_e} \tau_{\text{SUPG}} \frac{\mathbf{v}^h}{||\mathbf{v}^h||} \nabla w^h \Big(\frac{\partial u}{\partial t} + \mathbf{v} \cdot \nabla u^h - \nabla \cdot (\kappa \nabla u^h) \Big) \, d\Omega_e}_{\text{SUPG}} + \underbrace{\sum_e \int_{\Omega_e} \delta_{\text{CAU}} \nabla w^h \nabla u^h \, d\Omega_e}_{\text{CAU}} = \int_{\Omega_e} (w^h + \tau_{\text{SUPG}} \frac{\mathbf{v}^h}{||\mathbf{v}^h||} \nabla w^h) \cdot s \, d\Omega_e \tag{2.39}$$

onde,

$$\tau_{SUPG} = (4/\Delta t^2 + \mathbf{v}.\mathbf{G}\mathbf{v} + 9\kappa^2 \mathbf{G} : \mathbf{G})^{-0.5} \tag{2.40}$$

com Δt o tamanho da discretização temporal utilizada e $\mathbf{G}$ um tensor métrico de segunda ordem dado por

$$\mathbf{G} = \left(\frac{\partial \xi}{\partial \mathbf{x}} \right)^T \left(\frac{\partial \xi}{\partial \mathbf{x}} \right),$$

onde $\partial \xi / \partial \mathbf{x}$ é o inverso da matriz Jacobiana do elemento mapeado entre o sistema de coordenadas de referência e o físico.

Já, para o termo CAU, conforme apresentado em (5), tem-se

$$\delta_{\text{CAU}} = \frac{1}{2} \alpha_c h^e \frac{|L^h(u^h)|}{||\nabla u^h||} \tag{2.41}$$

onde

$$\alpha_c = \min\Big(\frac{1}{4} Pe^e_{//}, 1 \Big), \tag{2.42}$$

$$Pe^e_{//} = \frac{1}{2} h^e \frac{||\mathbf{v}^e_{//}||^3}{(\mathbf{v}^e_{//})^T \kappa \mathbf{v}^e_{//}}, \tag{2.43}$$

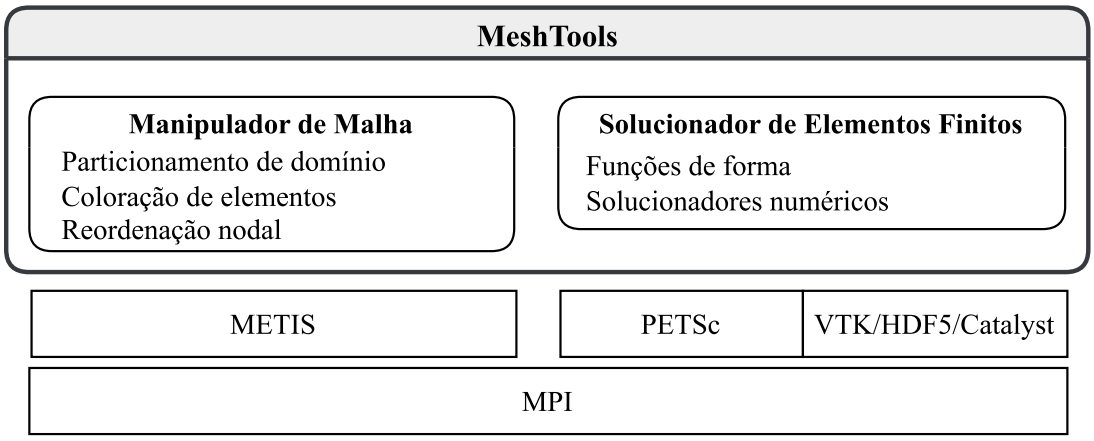
$$\mathbf{v}^e_{//} = \frac{\mathbf{v} \nabla u}{||\nabla u||^2} \nabla u, \tag{2.44}$$

com h^e o tamanho característico do elemento, $\mathbf{v}^e_{//}$ definido pela velocidade do elemento e na direção paralela ao gradiente da solução e $Pe^e_{//}$ o número de *Péclet* correspondente à $\mathbf{v}^e_{//}$. O termo $L^h(u^h)$ é referente ao resíduo da solução u^h interior ao elemento.

3 MESHTOOLS

MeshTools é um arcabouço computacional de código-livre, escrito em C++ que permite manipular uma malha de elementos finitos, preparando-a para o uso em sistemas computacionais híbridos massivamente paralelos. MeshTools é divido em dois componentes: Manipulador de Malha e Solucionador de Elementos Finitos. No componente de manipulação de malhas é feito o pré-processamento da malha a fim de otimizar sua resolução no componente solucionador de elementos finitos. Neste último componente, é possível solucionar uma Equação Diferencial Parcial através do Método de Elementos Finitos de forma paralela. O MeshTools é construído sobre bibliotecas de terceiros como a METIS (29), PETSc (9) e Catalyst (8). que serão definidas e descritas nas próximas seções. A estruturação do arcabouço é ilustrada na Figura 2.

Figura 2 – Estrutura do MeshTools



3.1 MANIPULAÇÃO DE MALHAS

O componente Manipulador de Malha é responsável pela leitura da malha, particionamento em caso de execução com múltiplos processos MPI e coloração dos elementos da malha, possibilitando computações paralelas em memória compartilhada de forma segura contornando condições de corrida, ou seja, linhas de execução, ou *threads*, que acessam simultaneamente um mesmo endereço de memória e geram inconsistência nos cálculos. Em caso de alguma escrita feita por uma *thread*, a outra que já fez a leitura estará trabalhando com um dado obsoleto causando inconsistência nos cálculos. Além da leitura, particionamento e coloração, o componente de manipulação também é responsável por aplicar a reordenação nodal, uma técnica que aprimora a localidade espacial dos dados na memória, e ainda, responsável pela escrita dos dados em disco.

Como entrada, o MeshTools suporta malhas no formato `Gmsh` (versão 2.0), na representação ASCII ou binária. Este formato é referente às malhas geradas no *software* `Gmsh` (21), um *software* de código livre de geração e visualização de malhas de elementos

finitos bidimensionais e tridimensionais, além de disponibilizar um solver acoplado e um módulo de pós-processamento pelo qual é possível analisar o resultado da solução do problema na malha gerada. O `Gmsh` também permite a geração e utilização de modelos *Computer-aided design* (CAD), facilitando o reuso de malhas.

3.1.1 Representação da malha

A malha é representada computacionalmente na classe `Mesh`. Esta classe é composta, além de outros atributos, por três principais *arrays*: coordenadas, conectividades e *offset*. O vetor de coordenadas, *coord*, armazena as coordenadas nodais (x, y, z) de cada ponto da malha no espaço. O vetor de conectividades, *conn*, armazena os nós que delimitam cada elemento da malha, ou seja, as conectividades de cada elemento. E por fim, o vetor *offset* mapeia a posição inicial de cada elemento no vetor de conectividades. A utilização dessas estruturas é exemplificada na Figura 3 em uma malha representativa de uma placa aplicada a um problema térmico.

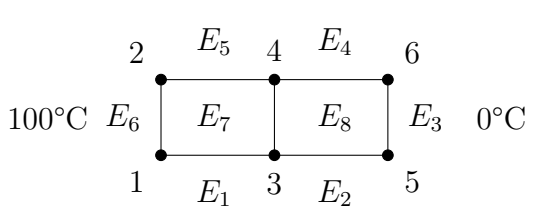
Uma malha é formada por elementos internos, os quais representam a região interna do domínio, e elementos de superfície, esses que delimitam o extremo do domínio. Tome, por exemplo, a Figura 3, a representação de uma placa com superfície a esquerda a 100ºC e à direita a 0ºC. Considere uma numeração nodal e elementar se iniciando de 1. Na Figura 3 os nós 1, 3, 4 e 2 delimitam o elemento E_7, um elemento interno, e os nós 1 e 2 delimitam o elemento E_6, um elemento de superfície. Os elementos de superfícies também são armazenados na classe `Mesh` de forma a facilitar a aplicação das condições de contorno do problema a ser solucionado e para aplicação de integrais de superfície.

Ainda na Figura 3, observe que o nó 1, representado por um círculo preto, tem suas coordenadas x_1, y_1, z_1 armazenadas nas três primeiras posições do vetor *coord*. Já o nó 2 tem as suas coordenadas armazenadas nas três próximas posições. Para a representação dos elementos da malha, o *arrays conn* e *offset* são utilizados em conjunto. Então, para um elemento i é possível obter a índice inicial dos seus nós delimitantes no vetor de conectividades a partir da sua posição relativa no arranjo *offset*, ou seja, na posição i. Com o índice inicial no vetor de conectividades, os nós do elemento são representados até a posição mapeada pelo índice $i + 1$ do vetor *offset*, ou seja, até a posição inicial do próximo elemento no vetor de conectividades. Essa forma de estruturação dos dados facilita o armazenamento de malhas com elementos de diferentes tipos.

Para exemplificar, observe o vetor *offset* na Figura 3. O primeiro elemento, E_1 tem sua conectividade nodal armazenada na posição 0 (offset[0]) à posição 2 (offset[1]) do vetor *conn*. Agora, para o elemento E_2, suas conectividades estão armazenadas da posição 2 (offset[1]) à posição 4 (offset[2]).

Porém, além de armazenar informações dos nós e elementos, também é necessário armazenar informações dos grupos físicos que compõe as diferentes regiões da malha.

Figura 3 – Malha de elementos finitos de uma placa e sua representação computacional.



coord: $[x_1, y_1, z_1, x_2, y_2, z_2, ..., x_6, y_6, z_6]$

conn: [1, 3, 3, 5, 5, 6, 6, 4, 4, 2, 2, 1, 1, 3, 4, 2, 3, 5, 6, 4]

offset: [0, 2, 4, 6, 8, 10, 12, 16, 20]

A criação desses grupos é necessária a fim de aplicar condições de contorno, condições iniciais, propriedades de material que compõe o domínio e cálculo de integrais de superfície. Estas informações físicas devem ser definidas no `Gmsh` e dessa forma são importadas pelo MeshTools.

Além dos *arrays* já apresentados, a fim de permitir integrações de superfície, é necessário ter a relação de adjacência entre os elementos de superfície e os elementos internos da malha. Essa relação possibilita determinar a direção do vetor normal à superfície. A construção dessa relação é apresentada no Algoritmo 1. A ideia básica do algoritmo é inserir todos os elementos de superfície em uma tabela *hash* calculada através da função Cantor Pairing (35). Nesta função é gerado um inteiro único a partir de uma tupla de inteiros. Ou seja, para um dado elemento é possível obter um valor único a partir da numeração das suas conectividades. Em seguida, após o armazenamento das *hashs* de cada elemento de superfície na tabela, os elementos internos da malha são percorridos identificando os que possuem alguma de suas faces inserida na tabela. Caso, a partir de um elemento interno, seja identificado uma face na tabela significa que o elemento de superfície referente à essa face é adjacente ao elemento interno em questão.

3.1.2 Reordenação nodal

A reordenação nodal se trata de uma técnica de reorganizar os nós da malha de forma que os elementos tenham conectividades numericamente mais próximas, aumentando a eficiência da busca de informações na memória principal pelo aprimoramento da localidade dos dados, reduzindo assim, o erro de acesso a *cache* e melhorando o desempenho global da aplicação. Para a aplicação da reordenação são disponibilizados no MeshTools dois algoritmos: Reverse Cuthill McKee (RCM) (15) e METIS_ND (28), disponibilizado pela biblioteca METIS, em que é aplicado o algoritmo Multilevel Nested Dissection.

A técnica de reordenação tenta explorar dois princípios da hierarquia de memória: o princípio da localidade espacial, onde prevê que dados próximos às informações requisitadas pelo processador serão provavelmente acessadas em breve. E também, o princípio da localidade temporal, o qual prevê que dados já acessados pelo processador serão provavelmente acessados novamente em breve. Dessa forma, o sistema de memória garante a localidade espacial colocando as informações em um bloco com os outros dados

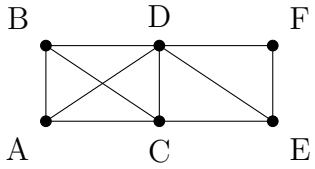
Algoritmo 1: Mapeamento entre elementos de superfície e internos

Entrada: $Mesh$ - Malha de elementos finitos
Saída: $Mesh$ - Malha de elementos finitos atualizada com relação entre elementos de superfície e internos

Instanciar vetor $face\_to\_element$ com tamanho $Mesh.n\_face\_elem$
Instanciar hash $face\_elements\_hash$ com tamanho $Mesh.n\_face\_elem$
Para $\forall fe_i \in Mesh.face\_elems$ **Faça**
  Gerar valor único $face\_hash$ com os índices dos nós de fe_i
  $face\_elements\_hash[face\_hash] \leftarrow fe_i$
Fim-Para
Para $\forall e_i \in Mesh.internal\_elems$ **Faça**
  Instanciar variável $nfaces$ com número de faces em e_i
  Para $\forall face \in nfaces$ **Faça**
    Gerar valor único $face\_hash\_e_i$ com índices dos nós de $face$
    Se $face\_hash\_e_i \in face\_elements\_hash$ **Então**
      $surface\_elem\_id \leftarrow face\_elements\_hash[face\_hash\_e_i]$
      $face\_to\_element[surface\_elem\_id] \leftarrow e_i$
    Fim-se
  Fim-Para
Fim-Para
Mesh.face_to_element = face_to_element

contínuos na memória. Este bloco é a menor unidade de dados transferidos entre níveis de memória. Já a localidade temporal é assegurada por continuamente trocar blocos utilizados menos recentemente na *cache* por outros atualmente requisitados pelo processador.

Figura 4 – Grafo nodal representante de uma malha de elementos finitos com 6 nós e 6 elementos.



Em ambos os algoritmos é necessário converter a malha de elementos finitos em um grafo nodal. O algoritmo RCM é apresentado no Algoritmo 2 onde uma estrutura de pilha foi utilizada a fim de substituir a ideia de recursão. Nele, a renumeração dos nós é iniciada a partir do nó de menor grau. O grau de um nó é definido como seu número de adjacências. Por exemplo, no grafo nodal representado pela Figura 4 o nó A tem grau 3 e o nó D tem grau 5. No caso do grafo apresentado na Figura 4, o nó de menor grau é o F com 2 nós adjacentes, dessa forma, o algoritmo seria inicializado nele. Caso exista mais de um nó com grau mínimo, o autor não especifica em qual o processo é iniciado. Após a numeração do primeiro nó, a numeração do resto do grafo é realizada recursivamente visitando e numerando nós adjacentes ao nó visitado em ordem crescente de grau. Ao final, observou-se que a reversão da ordem de *Cuthill-McKee* produz uma numeração melhor

para problemas de reordenamento de matrizes. Esse algoritmo é detalhado por (15) e sua versão original sem a reversão definida por (17).

Algoritmo 2: Reordenação *Reverse Cuthill-McKee*

Entrada: G - Grafo com V vértices e E arestas
Saída: n - Arranjo com a numeração dos vértices

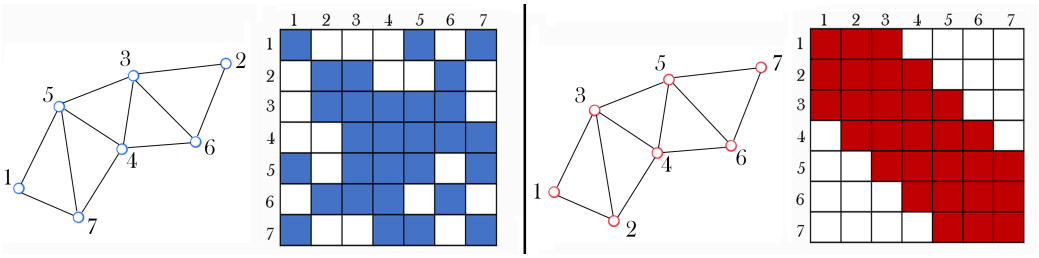
Encontrar x, nó de menor grau de V
$n \leftarrow x$
Instanciar pilha P
Empilhar x em P
Enquanto $P \neq \emptyset$ **faça**
  $x \leftarrow$ Desempilhar P
  Obter conjunto V, nós vizinhos de x ainda não numerados
  Ordenar V de menor a maior grau
  Para todo $v_i \in V$ **Faça**
    Empilhar P com v_i
    $n \leftarrow n + v_i$
  Fim-Para
fim-enquanto
Reverter ordem de n
Retorne n

Já no algoritmo *Multilevel Nested Dissection* (28), o grafo nodal G é dividido em dois subgrafos disjuntos a partir de um nó separador calculado a partir de uma aresta separada, a qual minimiza a cobertura de vértices de G. Dessa forma também é minimizado o conjunto de vértices separadores. A partir desses conjuntos complementares, cada subgrafo é renumerado aplicando esse algoritmo recursivamente.

Como um indicador de desempenho para a técnica de reordenação nodal, é possível analisar a matriz de adjacência do grafo que representa a malha de elementos finitos. Considere a_{ij} um elemento pertencente à matriz cuja posição é definida pela linha i e coluna j. Em um grafo, a ordenação dos nós é representada por uma matriz simétrica esparsa onde valores não nulos indicam adjacência entre os vértices i e j. Devido à organização dos computadores atuais, uma matriz esparsa com valores não-nulos mais concentrados (uma matriz com padrão de banda), rende um maior desempenho computacional. Isto decorre do fato de existir um melhor trecho na hierarquia de memórias, em que a maioria desses dados fica armazenado na *cache*, a qual é mais próxima fisicamente do processador, onde os cálculos são feitos. Portanto, a fim de alcançar uma maior eficiência na solução de Elementos Finitos, no MeshTools é disponibilizado a aplicação de técnicas atuais de programação como a reordenação nodal, através dos algoritmos RCM e METIS_ND apresentados nesta seção, a qual cria um novo arranjo aos nós do grafo, objetivando uma matriz com elementos não-nulos mais condensados ao entorno da diagonal principal.

Na Figura 5 é apresentado um exemplo da otimização resultante da técnica em uma malha composta por 7 nós. A numeração apresentada na malha da esquerda, com nós marcados em azul, resulta em uma matriz de adjacência com largura de banda de tamanho 7 e, após a aplicação de um algoritmo de reordenação nodal, é obtida a numeração dos nós apresentada na malha da direita, com nós marcado em vermelho, cuja matriz de adjacência referente tem largura de banda de tamanho 3. Esta redução na largura de banda indica numerações próximas dos nós pertencentes aos elementos, o que otimiza a utilização da hierarquia de memória no acesso dos elementos da malha.

Figura 5 – Representação da matriz de adjacência referente à uma malha antes e após a aplicação de um algoritmo de reordenação nodal.



3.1.3 Coloração de elementos

O advento de sistemas computacionais com múltiplos núcleos possibilitou o desenvolvimento de diversas aplicações que aproveitassem o paralelismo em memória compartilhada embarcado nestes sistemas. Neste tipo de paralelismo, a memória pode ser acessada simultaneamente por diferentes *threads*, fluxos de execução, de uma dada aplicação, se tornando essencial contornar a condição de corrida.

No MeshTools, para a aplicação do paralelismo em memória compartilhada, é utilizado a interface de programação OpenMP (18). Esta é uma interface para programação *multithread* em sistemas de memória compartilhada. O OpenMP é organizado em diretivas de compilação, biblioteca e variáveis de ambiente. Utilizando do modelo *Fork/Join*, o OpenMP viabiliza a criação e destruição de *threads*, ou seja, execução do código em seções paralelas ou seriais. Inicialmente a *threads master* executa o programa serial enquanto não encontra uma diretiva de paralelização, quando essa é alcançada é feito o *fork* criando-se um time de *threads* os quais irão executar determinada parte do código em paralelo. Ao fim da diretiva de paralelização, ou da estrutura de repetição paralelizada, ocorre o *join* quando o time de *threads* é destruído, voltando o código a ser executado em série pela *threads master*.

Em uma aplicação de elementos finitos, percorrer a lista de elementos é uma operação frequente, assim, otimizar esse processo é algo desejado. Entretanto, a utilização do paralelismo de memória compartilhada requer certos cuidados, uma vez que elementos vizinhos compartilham nós, ou seja, compartilham dados em mesmas posições de memória.

Uma das formas de contornar tal situação é a aplicação da técnica de coloração de grafos (22). Nessa técnica, uma malha de elementos finitos é tratada como um grafo onde os elementos são representados por vértices e suas arestas seriam a vizinhança entre esses elementos. Os vértices então são coloridos a partir das cores dos seus vértices adjacentes. Essa cor é representada por um valor inteiro. Com a finalização do algoritmo, vértices vizinhos tem cores distintas, possibilitando então a paralelização de elementos com mesma cor e, dessa forma, contornando o gargalo ocasionado pela condição de corrida. Neste momento, os arranjos *offset* e de conectividades, que representam os elementos, são reordenados posicionando elementos de mesma cor de forma consecutiva, aprimorando a utilização da memória *cache*.

O problema de coloração de malhas é classificado como um problema NP-Completo (20). Entretanto, existem inúmeras heurísticas que permitem obter uma solução não ótima em tempo polinomial. Um dos métodos mais simples é o algoritmo conhecido como *Greedy* (41). Neste esquema, o método visita os vértices do grafo atribuindo a menor cor possível, isto é, a menor cor ainda não atribuída aos vértices vizinhos.

Na Figura 7 é representado um domínio quadrado com dimensões unitárias $[0,1] \times [0,1]$ discretizado por uma malha estruturada de elementos finitos com 3 subdivisões nos eixos x e y, totalizando nove elementos do tipo quadrilátero e seu respectivo grafo dual. A Figura 6 ilustra a coloração por etapas utilizando o algoritmo *Greedy* da malha apresentada na Figura 7.

Algoritmo 3: Coloração *Greedy*

Entrada: G - Grafo com V vértices e E arestas

Saída: c - Arranjo com a coloração dos vértices

Para todo $v_i \in V$ **Faça**
  $C \leftarrow$ {Cores de todos os vértices coloridos $v_j \in adj(v_i)$} ;
  $c(v_i) \leftarrow$ {menor cor $\notin$ C} ;
Fim-Para

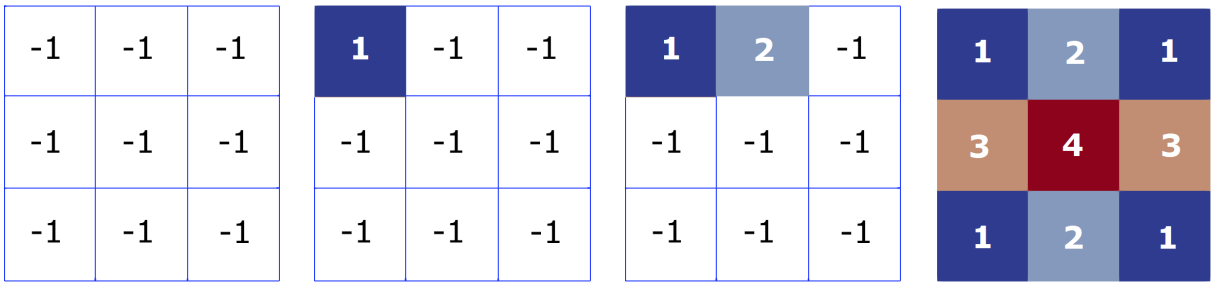
retorne c;

O pseudo-código do algoritmo de coloração *Greedy* é apresentado no Algoritmo 3. O algoritmo tem como entrada um Grafo G com V vértices e E arestas e retorna uma lista c contendo as cores atribuídas para todos os vértices de G. O mesmo é iniciado atribuindo para cada elemento (vértice no grafo dual) um valor que indica a ausência de cor (-1), como pode ser visto na Figura 6a. As cores que podem ser atribuídas para um elemento variam entre 1 até, no máximo, o número de elementos da malha. Para o primeiro elemento, Figura 6b, verifica-se as cores dos seus elementos adjacentes que, neste caso, são elementos que não têm cor. Assim, é escolhido a menor cor possível, representado pelo valor 1. Para o segundo elemento, Figura 6c, é novamente verificado as cores dos seus elementos adjacentes, sendo um com cor 1 e os outros sem cor, logo a menor cor possível

para esse elemento é representado pelo valor 2. Note que o valor 1 não seria possível, pois quebraria a regra de não ter elementos vizinhos com a mesma cor. No final da execução do algoritmo, obtém-se a malha colorida representada pela Figura 6d.

Também foi implementada a versão *Blocked* do algoritmo *Greedy* onde o número de elementos por cor é limitado. A limitação é estipulada pelo usuário a fim de possibilitar o armazenamento dos elementos de mesma cor na memória *cache* no momento da paralelização com OpenMP, aprimorando seu uso.

Figura 6 – Malha bidimensional exemplar.



(a) (b) (c) (d)

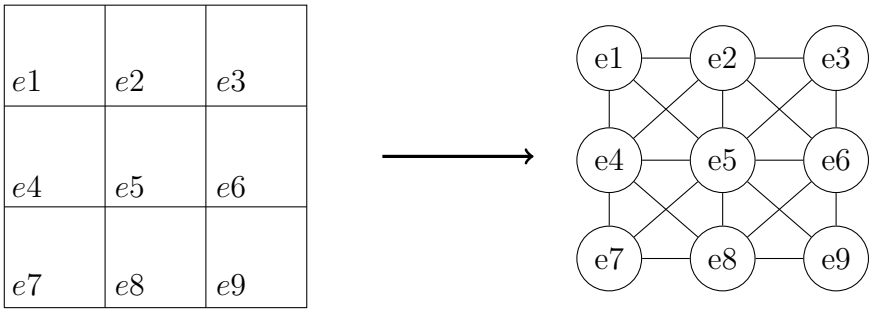
3.1.4 Particionamento de domínio

Com a necessidade de utilizar malhas cada vez mais refinadas, visando maior acurácia dos resultados, a memória se torna limitante para a aplicação do Método dos Elementos Finitos em sistemas seriais. Dessa forma, torna-se necessário o uso de técnicas de paralelismo do método em sistemas de memória distribuída. Assim, o domínio original precisa ser particionado e enviado a processos distintos, possibilitando a resolução do método de forma concorrente. Nesse sentido, visando um desempenho adequado do paralelismo, o particionamento deve ser feito de forma a balancear ao máximo a carga entre os múltiplos processos, assegurando assim, um número homogêneo de nós entre as malhas distribuídas. A fim de minimizar a comunicação entre processos MPI, deve ser visado, também, a minimização do número de nós de interface entre os domínios particionados. Comunicação excessiva entre processos é um gargalo no desempenho da paralelização em memória distribuída.

Para a realização do particionamento de domínio é utilizada a biblioteca METIS (29). METIS é uma biblioteca que oferece métodos para particionamento de grafos, particionamento de malhas de elementos finitos e redução da largura de banda de matrizes esparsas. Esta biblioteca destaca-se por utilizar algoritmos otimizados que permitem o particionamento de malhas não estruturadas, além de possuir rotinas para a otimização de largura de banda em matrizes esparsas. A biblioteca METIS trata o particionamento de malha como um problema de particionamento de grafo. Duas metodologias podem ser empregadas: particionamento do grafo nodal e particionamento do grafo dual. Na primeira

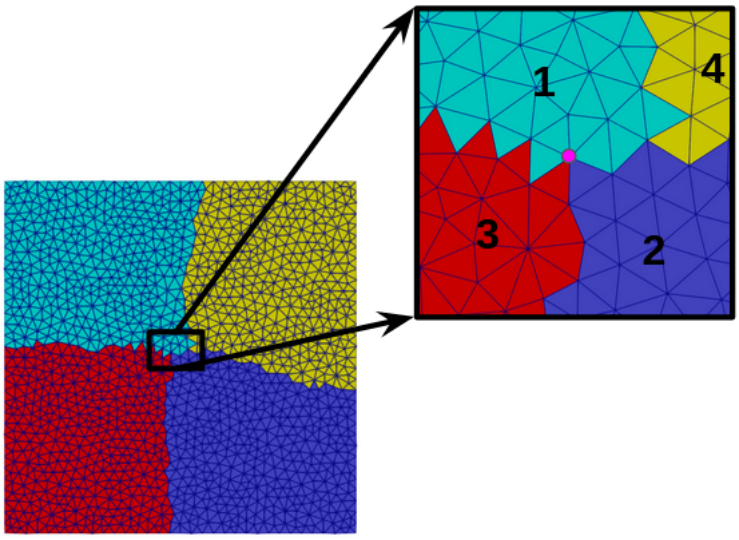
técnica, cada nó da malha torna-se um vértice do grafo enquanto que, na segunda técnica, cada elemento da malha torna-se um vértice do grafo, como representado na Figura 7. O MeshTools emprega o particionamento utilizando de um grafo dual.

Figura 7 – Malha de elementos finitos e sua representação em um grafo dual



Esta etapa de decomposição do domínio é executada pelo processo raiz, o de *rank* 0, através da classe `MeshPartition` a qual manipula as estruturas e aplica os métodos disponibilizados pela biblioteca METIS. Em seguida, os dados da malha particionada são distribuídos, por meio de mensagens MPI, aos outros processos executados em paralelo. Na Figura 8 é ilustrada a decomposição de uma malha em quatro partições aplicadas através da classe `MeshPartition`. A malha particionada referente à cada processo é organizada na classe `ParallelMesh`. Esta classe herda da classe `Mesh` e contém informações adicionais referente ao particionamento como o *rank* dos processos vizinhos, nós de interface e mapeamento da numeração local e global dos nós. Ao final da distribuição, determina-se o mapa de comunicação entre as partições.

Figura 8 – Exemplo de particionamento de malha. Os números indicam o número referente ao processo paralelo.



3.1.4.1 Mapa de comunicação

A etapa de construção do mapa de comunicação preenche atributos da classe `ParallelMesh` responsáveis por armazenar a relação de comunicação entre partições vizinhas e identificando o dono de um nó de interface, ou seja, o processo responsável por computá-lo e compartilhar seu resultado entre os processos MPI vizinhos. Esta etapa é demonstrada no Algoritmo 4.

Inicialmente é preenchido um arranjo, cujo tamanho é o número de nós da malha particionada, com o *rank* do processo executante. Neste arranjo, é armazenado para cada nó, o identificador do processo com maior *rank* entre os processos em que ele é compartilhado. Essa informação sendo preenchida, é feito um loop pelos processos vizinhos e se o *rank* do processo executante for menor que o processo vizinho iterado, os nós compartilhados entre eles serão inseridos no *buffer* de recebimento. Caso contrário, os nós são armazenados no *buffer* de envio. Então dessa forma, para cada nó de interface, o processo de maior *rank* que o pertence será o líder e os outros processos, que compartilham também o mesmo nó, serão os subalternos. Assim, é possível estabelecer a relação de comunicação entre os processos apresentada em (43).

Algoritmo 4: Construção do mapa de comunicação

Entrada: $PMesh$ - Malha paralela de elementos finitos

Saída: $PMesh$ - Malha paralela de elementos finitos atualizada com mapa de comunicação

```
Instanciar vetor maior_proc_vizinho com tamanho PMesh.n_nós
Preencher maior_proc_vizinho com o rank do processo executante, my_rank
Para ∀p ∈ proc_vizinhos Faça
    Para ∀nó_i ∈ p.nós_compartilhados Faça
        Se p > my_rank Então
            nó_i.maior_proc_vizinho ←p;
        Fim-se
    Fim-Para
Fim-Para

Para ∀p ∈ proc_vizinhos Faça
    // my_rank é subalterno do processo p
    Se my_rank < p Então
        Para ∀nó_i ∈ p.nós_compartilhados Faça
            Se nó_i.maior_proc_vizinho = p Então
                Insere nó_i ao buffer de recebimento relativo ao processo p
            Fim-se
        Fim-Para
    Fim-se
    // my_rank é líder do processo p
    Se my_rank > p Então
        Para ∀nó_i ∈ p.nós_compartilhados Faça
            Se nó_i.maior_proc_vizinho = p Então
                Insere nó_i ao buffer de envio relativo ao processo p
            Fim-se
        Fim-Para
    Fim-se
Fim-Para
Armazenar buffers em PMesh
Retornar PMesh
```

Na Figura 8, a imagem ampliada representa uma região em comum às partições onde existem nós de interface. Nesse caso, o nó de exemplo em rosa é compartilhado pelos processos 1, 2 e 3. Dessa forma, através do Algoritmo 4, o processo 3, de maior *rank*, é definido como o líder e os processos 1 e 2, seus subalternos.

Através desta estratégia, é possível cada processo definir seus *buffers* de envio e recebimento de mensagens. Ao final da criação do mapa de comunicação, é calculado

e armazenado na malha paralela de cada processo a relação entre sua numeração local e global dos nós. Essa relação é utilizada pelo componente Solucionador de Elementos Finitos do MeshTools para a resolução do sistema linear em paralelo.

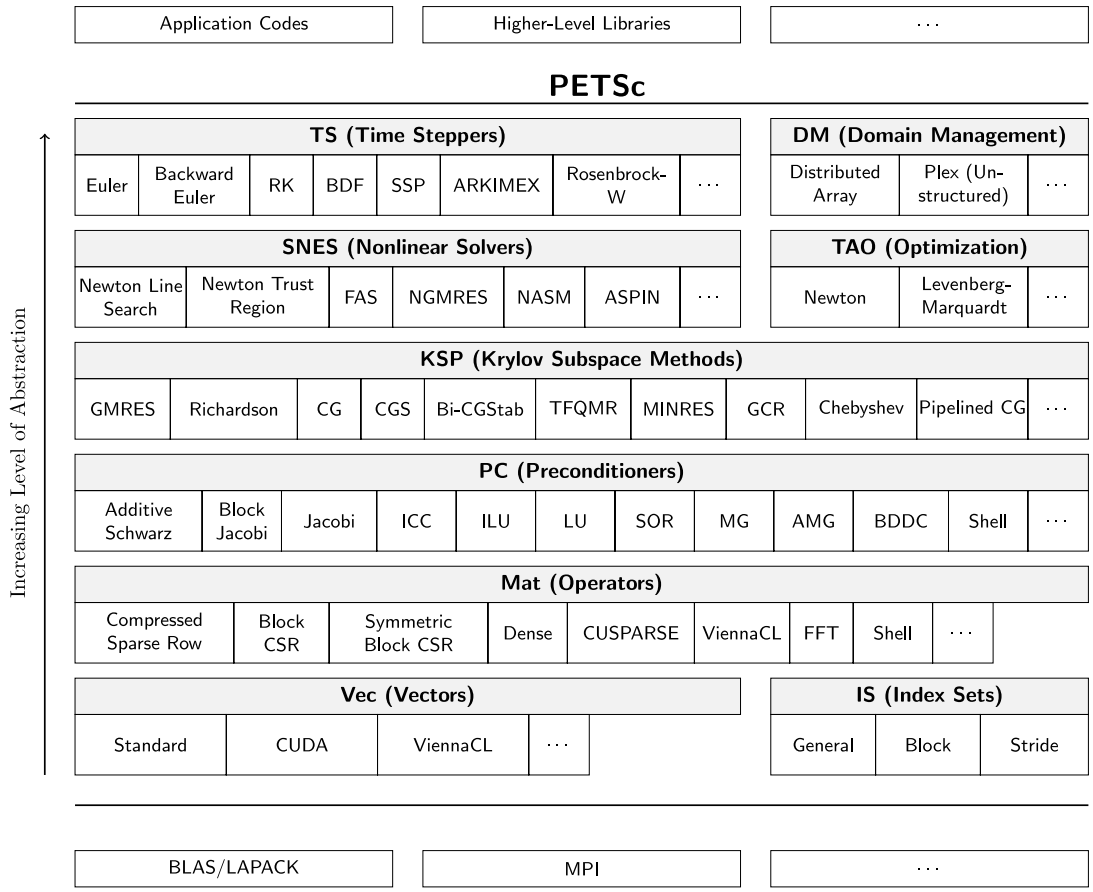
3.2 SOLUCIONADOR DE ELEMENTOS FINITOS

Neste componente é realizada a solução do Método dos Elementos Finitos em paralelo. O Método dos Elementos Finitos é um método numérico para resolução de Equações Diferenciais Parciais. Neste método, a solução destas equações é realizada através da solução de um sistema de equações lineares do tipo

$$Ax = b,$$

onde A é uma matriz de tamanho $n \times n$, x o vetor incógnito e b o vetor independente, ambos com tamanho n. Sendo n o número de graus de liberdade vezes o número de nós da malha.

Figura 9 – Estrutura da biblioteca PETSc. Imagem obtida do manual da biblioteca.



Para solucionar este sistema de equações no MeshTools, é utilizada a biblioteca PETSc (9). PETSc é um arcabouço computacional que possibilita a solução de aplicações modeladas por Equações Diferenciais Parciais de maneira escalável. Além da possibilidade de solucionar estas aplicações em sistemas paralelos, PETSc também provê armazenamento

de matrizes esparsas e diversos métodos para solução de sistemas lineares como Gradiente Conjugado (45), GMRES (42) e Chebyshev (23) conforme representado na Figura 9. A convergência da solução do sistema por meio de métodos iterativos pode ser aprimorada com a aplicação de pré-condicionadores disponibilizados na PETSc entre eles Jacobi, *Incomplete* LU (ILU) e LU como apresentado também na Figura 9. Também é possível escolher a tolerância e o tipo de norma para teste de convergência. Estas opções podem ser selecionadas através dos argumentos de execução do MeshTools.

A biblioteca PETSc é acoplada ao componente de resolução do Método de Elementos Finitos a fim de usufruir de suas ferramentas. Assim, o MeshTools é definido sobre as estruturas da biblioteca PETSc utilizando seus objetos para armazenamento da matriz e vetor independente do sistema de equações e também o solucionador linear `KSP`.

3.2.1 Estruturas PETSc

Para a resolução do método numérico, é necessário armazenar a matriz, o vetor independente e o solucionador do sistema. Estas estruturas são armazenadas, respectivamente, nas estruturas `Mat`, `Vec` e `KSP`. Para o armazenamento de matrizes, PETSc disponibiliza diversos tipos de estruturas como apresentado na Figura 9. MeshTools utiliza a matriz tipo `MATAIJ` para matrizes esparsas sequenciais ou paralelas armazenadas em formato *Compressed Sparse Row* (CSR). Neste tipo de formato, é armazenado em um arranjo os valores não-nulos de cada linha da matriz, em outro arranjo são armazenadas suas posições nas respectivas linhas e em um terceiro arranjo é armazenado o índice de início de cada linha do primeiro e segundo arranjos. Para o armazenamento de vetores o principal tipo de estrutura é o `Vec Standard`, havendo também a disponibilidade na PETSc de estruturas voltadas para arquitetura de GPU com CUDA e ViennaCL, como também apresentado na Figura 9.

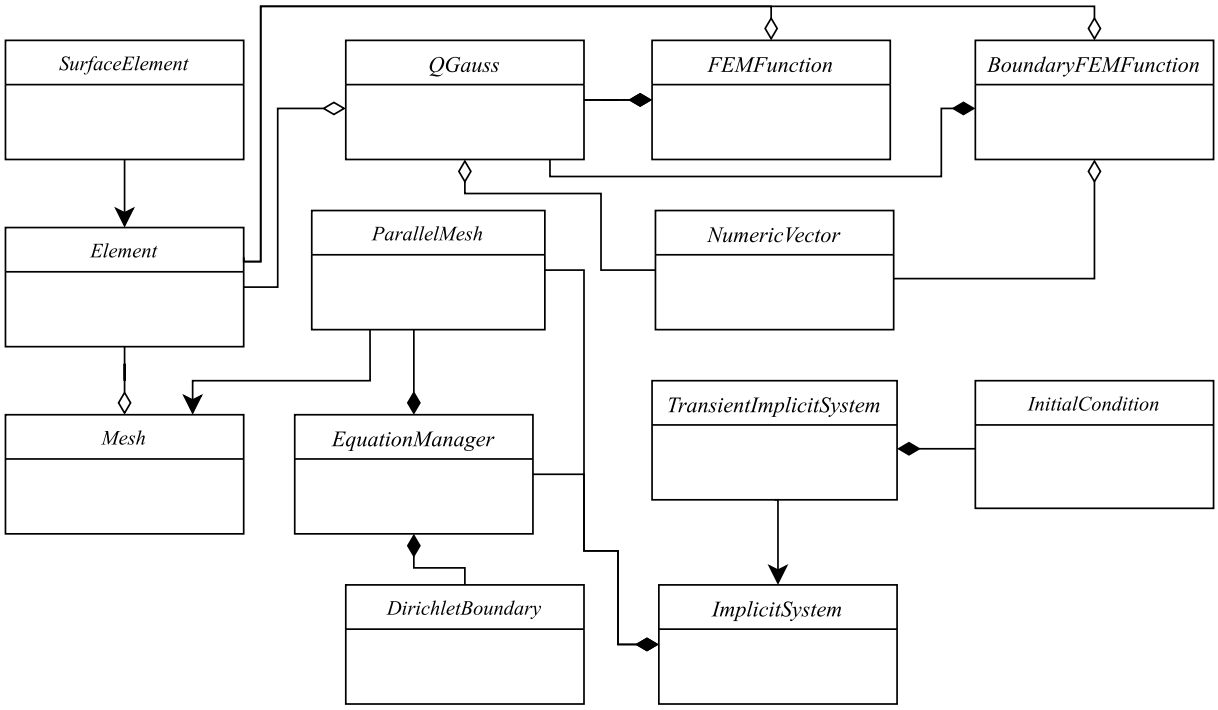
Além da utilização de estruturas para armazenar a matriz e vetor independente do sistema de equações, também é necessário instanciar o objeto responsável pela solução do sistema, o `KSP`. Este resolve o sistema em série ou em paralelo, além de possibilitar encontrar a solução via métodos iterativos ou diretos. Para utilizar o `KSP` é necessário instanciar sua classe indicando o comunicador MPI, caso esteja sendo executado em paralelo, e em seguida indicar a matriz A do sistema e também a matriz pela qual o pré-condicionador será criado. No MeshTools, a matriz utilizada para o cálculo do pré-condicionador também é a matriz A do sistema.

3.2.2 Classes do componente Solucionador de Elementos Finitos

Na Figura 10 é apresentado o relacionamento das classes implementadas no MeshTools para solução de elementos finitos. Os atributos e métodos foram suprimidos a fim de apresentar somente a relação entre as classes, dessa forma, nas subseções as principais

classes são apresentadas com maiores detalhes.

Figura 10 – Diagrama das classes implementadas no MeshTools utilizadas para solução de Elementos Finitos



3.2.2.1 Condição de contorno

Em uma Equação Diferencial estacionária, as condições de contorno garantem uma solução única para o problema. Para a aplicação de condições de contorno do tipo Dirichlet é implementada a classe `DirichletBoundary`. O diagrama desta classe é apresentado na Figura 11. Neste tipo de condição, o valor da variável de interesse é estipulado no contorno do domínio. Além deste tipo de condição de contorno, também é possível aplicar a condição do tipo Neumann em que o valor da derivada da variável de interesse é estipulado. Para este caso é necessário acrescentar, na formulação de elementos finitos, o termo referente à integração de superfície fazendo uso das funções de forma referentes presentes na classe *BoundaryFEMFunction*, apresentada na Seção *3.2.2.4*.

Figura 11 – Diagrama da classe *DirichletBoundary*.

DirichletBoundary
-funcion_parser: FunctionParser -_dof_id: int -_boundary_surface: int
+DirichletBoundary(int boundary_surface, int dof_id, string function, string vars): DirichletBoundary +get_value(double x, double y, double z, double t): double +get_dof_id(): int +get_boundary_id(): int

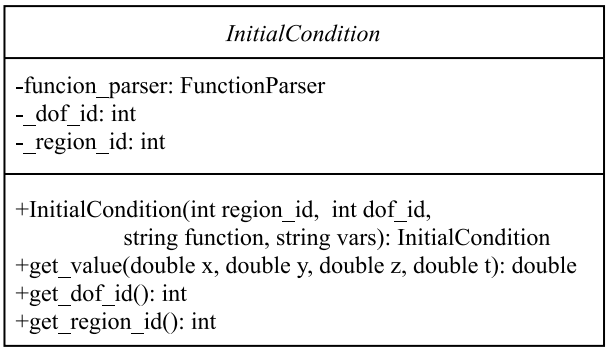
Para a interpretação da função matemática aplicada no contorno é utilizada a classe `FunctionParser` da biblioteca FParser (38). Através desta classe é possível escrever uma função em formato texto e fazer sua avaliação em um determinado ponto do domínio.

É necessário indicar para a classe `DirichletBoundary` o identificador da condição de contorno definido como uma entidade física pelo `Gmsh` e também o identificar do grau de liberdade relacionado à condição de contorno definida. Além disso, explicitar a função a ser aplicada neste contorno e as variáveis desta função também é necessário.

3.2.2.2 Condição inicial

Em problemas transientes, representados por meio de equações com termo temporal, é necessário que a condição inicial do domínio seja especificada. Para isso é disponibilizada a classe *InitialCondition* com seu diagrama apresentado na Figura 12.

Figura 12 – Diagrama da classe *InitialCondition*.



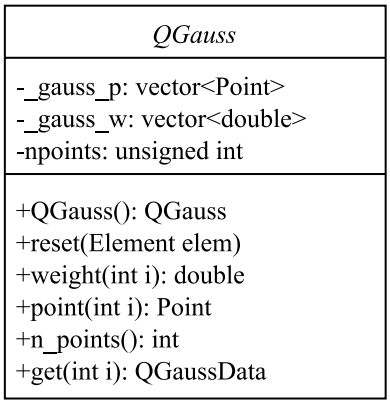
Esta classe é estruturada de forma similar à classe *DirichletBoundary*, responsável por armazenar as condições de contorno do tipo Dirichlet, apresentada na Seção *3.2.2.1*. Isso ocorre pelo fato de ambas as classes estipularem o valor de uma função matemática em uma região pré-determinada do domínio. Analogamente à condição de contorno, a condição inicial é referente a um grau de liberdade, que deve ser indicada na instância da classe, porém esta é estipulada em uma região e não somente em uma superfície de contorno. Dessa forma, na classe *InitialCondition* é armazenado, também, o identificador da região em que a função será aplicada.

3.2.2.3 Integração numérica

A integração numérica é manipulada através da classe de Quadratura Gaussiana. A Figura 13 representa o diagrama da classe *QGauss*, a qual armazena informações necessárias para o cálculo da quadratura gaussiana para os diferentes tipos de elementos suportados pelo MeshTools.

O objeto desta classe é reutilizado durante a solução de elementos finitos utilizando o método `reset`. Para este método, é informado por parâmetro o elemento em que a

Figura 13 – Diagrama da classe *QGauss*.

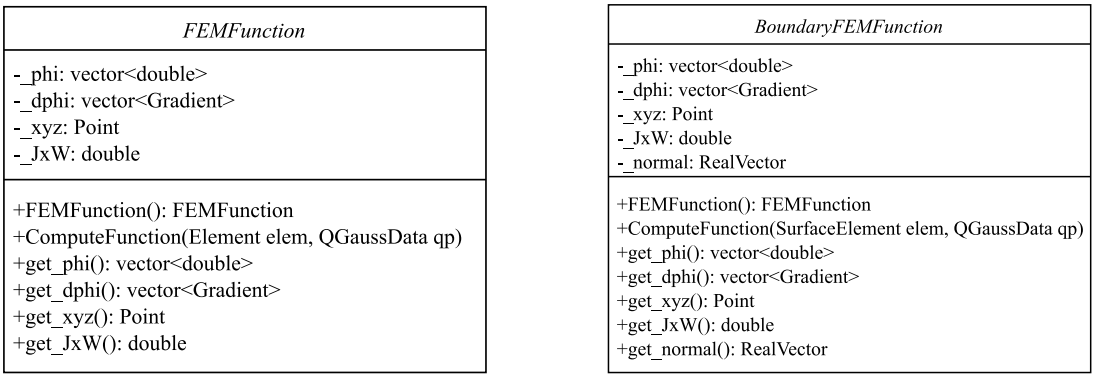


integração numérica será calculada e dessa forma, feita a identificação do tipo do elemento, possibilitando o armazenamento dos seus pontos de integração e seus respectivos pesos.

3.2.2.4 Funções de forma

O cálculo das funções de forma são definidos nas classes *FEMFunction* e *BoundaryFEMFunction* respectivamente para elementos internos da malha e para elementos de superfície. Os diagramas de ambas as classes são apresentados nas Figuras 14a e 14b.

Figura 14 – Diagramas das classes (a) *FEMFunction* e (b) *BoundaryFEMFunction*



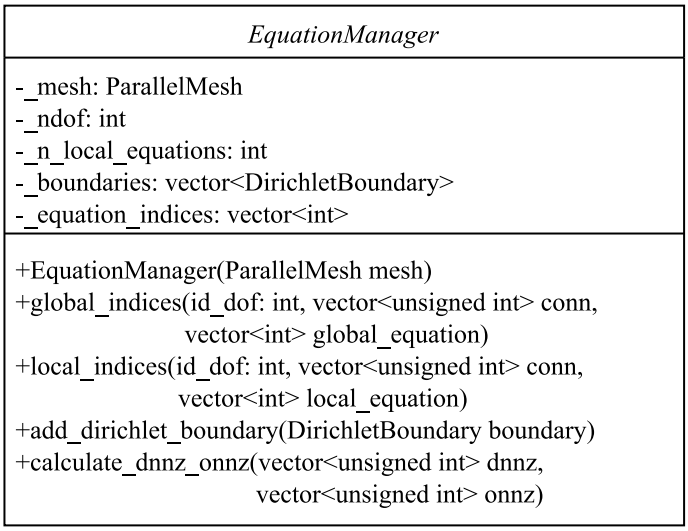
(a) (b)

Através do método `ComputeFunction` é calculada a função de forma do elemento passado como parâmetro para o ponto de integração estipulado. Em seguida, são preenchidas as variáveis que armazenam o valor da função de forma para o ponto de integração determinado e o valor das suas derivadas em relação a cada eixo do sistema de coordenadas. Além disso, é calculado a matriz Jacobiana a qual mapeia o sistema de coordenadas físicas no sistema de coordenadas de referência.

3.2.2.5 Gerenciamento de equações

Através da classe *EquationManager* é possível realizar o gerenciamento das equações da malha. O diagrama desta classe é representada na Figura 15.

Figura 15 – Diagrama da classe *EquationManager*.



Para a instância desta classe é necessário informar a malha de elementos finitos paralela via parâmetro. Com a malha de elementos finitos e o número de graus de liberdade do problema sendo solucionado, é feita a numeração das equações locais e globais utilizadas na PETSc. Com esta classe é possível realizar o mapeamento entre cada nó da malha e grau de liberdade para a numeração das equações.

Além disso, a biblioteca PETSc provê estruturas para otimizar o armazenamento da matriz de elementos finitos. Para isto é necessário realizar um pré-processamento da matriz calculando o número de elementos não-nulos por linha que se encontram dentro do bloco diagonal e os que se encontram fora do bloco diagonal. Esta otimização da matriz do sistema linear é implementada pelo método `calculate_dnnz_onnz`, apresentado na Figura 15.

Tendo em vista que a matriz paralela é armazenada por linhas na memória, utilizando do formato CSR como explicado na Seção **3.2.1**, se torna importante obter o número de elementos não-nulos fora do bloco diagonal da matriz paralela por conta da resolução do sistema linear, o qual necessita calcular diversos produtos entre matriz e vetor. No bloco diagonal da matriz estão armazenados os dados que serão multiplicados pelo vetor pertencente ao processo executante. Já os valores posicionados fora do bloco diagonal da matriz, serão multiplicados por dados de vetores pertencentes aos processos concorrentes. Dessa forma, é necessário haver a troca de mensagens entre os processos para possibilitar este cálculo. A biblioteca PETSc então realiza uma troca de mensagens entre os processos comunicando os elementos não-nulos da matriz pertencentes fora do bloco diagonal.

Na Figura 16 é representada uma matriz paralela de tamanho 8×8 particionada com as três primeiras linhas pertencentes ao processo P_1, as três próximas pertencentes ao processo P_2 e as duas últimas pertencentes ao processo P_3. Processando a otimização da

alocação desta matriz, para a primeira linha do processo P_1 existem 2 elementos não-nulos pertencentes ao bloco diagonal, os elementos 1 e 2, e existem também 2 elementos não-nulos pertencentes ao bloco não diagonal, os elementos 3 e 4. Para a segunda linha do processo P_1 tem o mesmo número de elementos não-nulos tanto no bloco diagonal quanto no bloco não diagonal. Já para a segunda linha pertencente ao processo P_2, que seria a quinta linha da matriz global, existem 1 elemento no bloco não diagonal, de valor 18, e 3 no bloco diagonal, de valores 19, 20 e 21. Além das linhas da matriz, as colunas também são particionadas tendo em vista a decomposição do vetor no produto matriz vetor realizados durante a solução do sistema linear. Dessa forma, os elementos da matriz armazenados no processo P_1 são multiplicados por elementos do vetor armazenados no processo P_1, P_2 e P_3.

Figura 16 – Representação de uma matriz paralela particionada em 3 processos, P_1, P_2 e P_3.

$$
\begin{array}{c}
 \\
P_1 \\
 \\
 \\
P_2 \\
 \\
 \\
P_3 \\
 \\
\end{array}
\begin{array}{c}
\begin{array}{ccc} P_1 & & \end{array} \\
\left(\begin{array}{ccc|ccc|cc}
1 & 2 & 0 & 0 & 3 & 0 & 0 & 4 \\
0 & 5 & 6 & 7 & 0 & 0 & 8 & 0 \\
9 & 0 & 10 & 11 & 0 & 0 & 12 & 0 \\
\hline
13 & 0 & 14 & 15 & 16 & 17 & 0 & 0 \\
0 & 18 & 0 & 19 & 20 & 21 & 0 & 0 \\
0 & 0 & 0 & 22 & 23 & 0 & 24 & 0 \\
\hline
25 & 26 & 27 & 0 & 0 & 28 & 29 & 0 \\
30 & 0 & 0 & 31 & 32 & 33 & 0 & 34
\end{array}\right)
\end{array}
\begin{array}{c}
 \\
 \\
 \\
 \\
 \\
 \\
 \\
 \\
 \\
\end{array}
\begin{array}{ccc} & P_2 & \end{array}
\begin{array}{cc} & P_3 \end{array}
$$

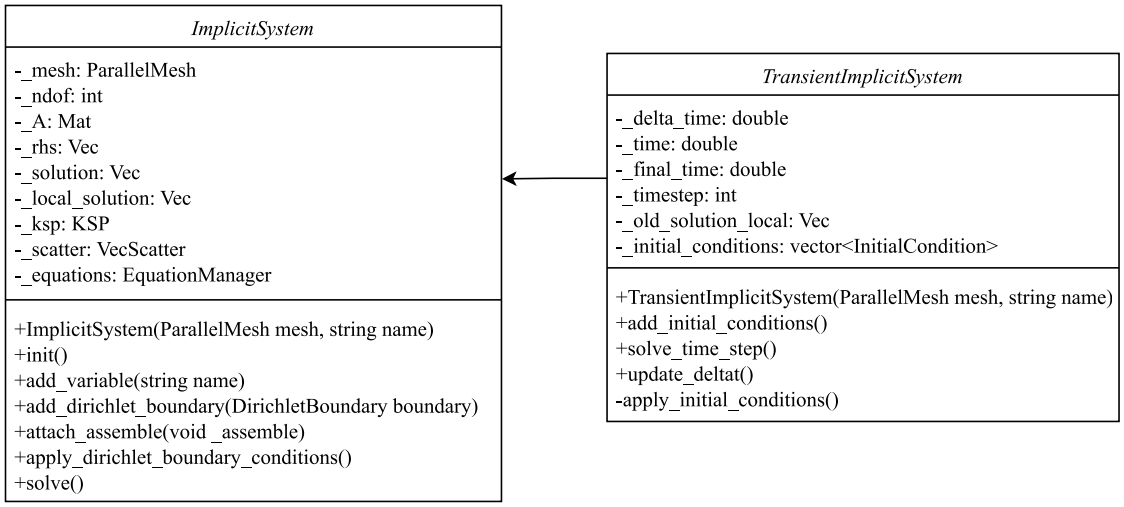
No contexto de uma matriz gerada por meio de uma Malha de Elementos Finitos particionada, o bloco diagonal é referente aos elementos da matriz, ou equações da malha de elementos finitos, que pertencem ao processo executante, e o bloco não-diagonal é referente aos elementos da matriz, ou equações da malha de elementos finitos, que pertencem ao processo vizinho líder. Ou seja, esses elementos pertencentes ao bloco não diagonal seriam referente aos graus de liberdade dos nós de interface que pertencem ao processo líder do processo executante, como explicado na Seção *3.1.4.1*.

Esta classe *EquationManager* compõe as classes solucionadoras de sistema de equações para mapear os nós da malha para seus respectivos índices das equações e realizar a otimização no armazenamento da matriz do sistema linear.

3.2.2.6 Tipos de sistemas implementados

Para a solução do problema de Elementos Finitos, MeshTools disponibiliza dois tipos de sistema: implícito e transiente. O tipo de sistema implícito é implementado na classe *ImplicitSystem* utilizado para resolver problemas estacionários e o sistema transiente

Figura 17 – Diagramas das classes *ImplicitSystem* e *TransientImplicitSystem*.



implementado na classe *TransientImplicitSystem*. O diagrama das classes são apresentadas na Figura 17.

As duas classes responsáveis por solucionar diferentes tipos de sistemas são organizadas utilizando relação de herança, um princípio da Programação Orientada a Objetos. Na classe mãe, *ImplicitSystem* são armazenadas informações como a malha manipulada no componente anterior, as condições de contorno, o número de graus de liberdade e o nome das variáveis a serem solucionadas para escrita no arquivo de saída. E, a classe *TransientImplicitSystem* herda da classe *ImplicitSystem*, armazenando também a condição inicial da equação, o tamanho do passo temporal e o passo de tempo final a ser resolvido. Para a montagem do sistema de equações é necessário acoplar uma função que realize esta etapa do método.

Por meio de ambas as classes é possível resolver um sistema de equações não-linear. Para isto deve ser estipulado a tolerância do erro da resolução do sistema não-linear e o número máximo de iterações para resolução deste sistema. Para a resolução do sistema não-linear é utilizado o Método de Picard (40). Neste método, a solução encontrada na iteração anterior da solução do sistema não-linear é utilizada para calcular a próximo iteração. No momento em que a diferença entre estes valores for menor que um valor, definido pela tolerância, significa que o método convergiu para a solução.

3.2.2.6.1 Classe *ImplicitSystem*

Através desta classe, apresentada na Figura 17, é possível solucionar um problema estacionário definido em um domínio representado pela malha paralela informada por parâmetro na instância do objeto. Na inicialização do sistema implícito são alocadas as estruturas da biblioteca PETSc como a matriz do problema, o vetor independente e o objeto `KSP`, o solucionador do sistema, apresentadas na Seção **3.2.1**. Também, na

inicialização é feito o cálculo para a otimização no armazenamento da matriz identificando o número de elementos não-nulos pertencentes e não pertencentes ao bloco diagonal. Além disso, é preenchido o gerenciador de equações definido na Seção *3.2.2.5* com a relação das equações pertencentes à alguma condição do tipo Dirichlet para posterior aplicação da respectiva condição na resolução do sistema.

O sistema linear global é construído acumulando matrizes e vetores elementares da malha de elementos finitos. Para as equações destes elementos, pertencentes à condição de contorno do tipo Dirichlet, deve ser definida uma técnica para a obtenção do valor estipulado no vetor solução. Para isto, é possível desconsiderar nós pertencentes à condição de contorno no sistema, reduzindo assim o tamanho da matriz e do vetor independente, e inserindo manualmente o valor destas equações no vetor solução. Porém, através desta estratégia, com a existência de processos com mais equações pertencentes à condição de contorno do tipo Dirichlet, haveria um desbalanceamento de carga entre os processos paralelos, otimizado pela biblioteca METIS no particionamento do domínio como detalhado na Seção **3.1.4**. Então, no MeshTools, todas as equações são consideradas nas matrizes e vetores elementares, porém, ao final da montagem da matriz global, estas equações pertencentes à condição de Dirichlet têm sua linha zerada com inserção do valor 1 na diagonal principal e é substituído o valor da condição de contorno no vetor independente, forçando a equação a adquirir o valor estipulado. Foi optado por esta estratégia a fim de garantir o balanceamento de dados calculado pela biblioteca METIS no particionamento de domínio da malha.

A solução do sistema linear é armazenado em um vetor referente à numeração global das equações. Para a obtenção dos resultados em relação à numeração local da malha paralela é necessário fazer o *scattering* dos dados, a distribuição entre os processos executantes. Para isso é utilizada a estrutura `VecScatter` da biblioteca PETSc, um objeto para comunicação de dados entre vetores paralelos. A fim de instanciar o `VecScatter`, é necessário informar a estrutura `IndexSet`, a qual armazena a numeração das equações envolvidas no processo. Deve ser informado o `IndexSet` referente à numeração local e global das equações do processo. Estas numerações são obtidas da classe *EquationManager* da MeshTools apresentada na Seção *3.2.2.5*.

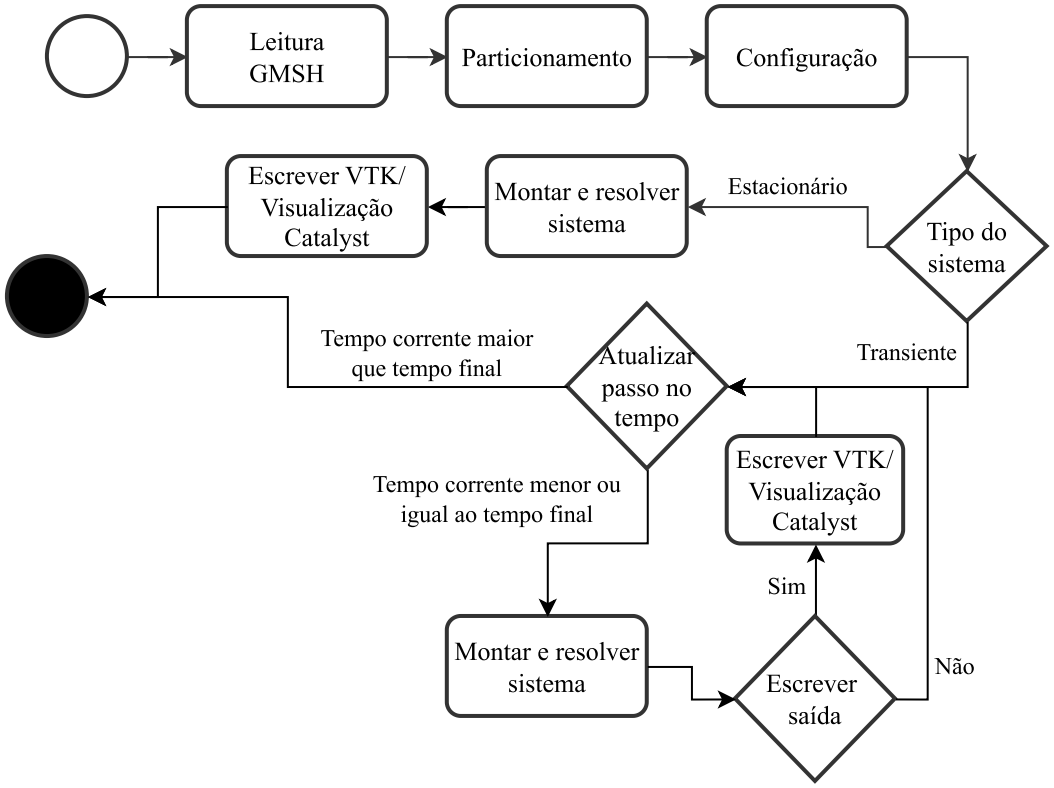
3.2.2.6.2 Classe *TransientImplicitSystem*

Na inicialização do sistema transiente, além dos cálculos realizados no sistema implícito apresentados na Seção 3.2.2.6.1, também é instanciado o vetor solução no passo de tempo anterior, necessário para a solução de um problema transiente. Além disso, a condição inicial necessária para solução de um problema transiente também é preenchida na inicialização desta estrutura. A condição inicial é aplicada no vetor solução do sistema para todas as equações da malha de elementos finitos.

3.2.3 Fluxo de código para solução de um problema por elementos finitos

Nesta seção será abordado o fluxo seguido pelo MeshTools na solução de um problema através do Método dos Elementos Finitos em paralelo.

Figura 18 – Diagrama com fluxo de execução do componente Solucionador de Elementos Finitos no MeshTools



Na Figura 18 é representado o diagrama com o fluxo de execução do MeshTools. O código para resolver uma equação transiente utilizando o MeshTools é apresentado na Figura 19. Neste código de exemplo é resolvido um sistema com variável nomeada u, condição de contorno valendo 0.0 aplicada no grupo físico 1 da malha e condição inicial valendo 1.0 aplicada no grupo físico 2 da malha. O problema é resolvido com discretização temporal 0.005 e tempo final 8.0.

Para resolver um problema de elementos finitos no MeshTools, como apresentado na Figura 18, é inicialmente feita a leitura da malha em formato `Gmsh`. Esta etapa é realizada através da instância da classe `Mesh` indicando o caminho para a malha de elementos finitos, a qual representa a linha 15 de código da Figura 19. Em seguida, são aplicadas as técnicas apresentadas na Seção 3.1 para otimização e possibilidade de resolver o método numérico em paralelo que representam as linhas 18 e 20 de código da Figura 19. Neste caso é aplicado somente a técnica de particionamento de domínio possibilitando distribuir a malha entre processos MPI.

Após a aplicação destas técnicas, deve ser feita a configuração do MeshTools para solução de elementos finitos. Esta etapa de configuração representa as linhas do intervalo

Figura 19 – Exemplo de código para solução de um problema de elementos finitos no MeshTools

```
int main(int argc, char *argv[])
{
    MeshTools::Init(argc,argv);
    CatalystAdaptor::Initialize(argc, argv);

    MeshPartition *parts = new MeshPartition();
    Mesh *mesh;
    ParallelMesh *pmesh;

    int processor_id = MeshTools::processor_id();
    int n_processors = MeshTools::n_processors();

    if (processor_id == 0)
    {
        mesh = new Mesh("caminho/para/malha.msh");

        if (n_processors > 1)
            parts->ApplyPartitioner(mesh, n_processors);
    }
    pmesh = parts->DistributedMesh(mesh);

    TransientImplicitSystem *system = new TransientImplicitSystem(
        *pmesh, "sistema_exemplo"
    );
    system->add_variable("u");
    DirichletBoundary  bc(1,0,"0.0","x,y,z");
    InitialCondition  ic(2,0,"1.0","x,y,z");
    system->add_dirichlet_boundary(bc);
    system->add_initial_condition(ic);
    system->attach_assemble(assemble_method);

    system->init();
    system->set_final_time(8.0);
    system->set_deltat(0.005);
    unsigned int write_vtk_interval = 50;
    unsigned int catalyst_interval = 25;


    while(system->get_time() < system->get_final_time()){
        system->solve_time_step();

    if(system->get_time_step()%write_interval == 0 )
        system->write_result("nome_da_saida");

    if(system->get_time_step()%catalyst_interval == 0 )
        CatalystAdaptor::Execute(system->get_time_step(),
             system->get_time(), static_cast<ImplicitSystem*>(system));
    }

    delete system;
    if (MeshTools::processor_id() == 0)
        delete mesh;
    delete pmesh;
    delete parts;

    CatalystAdaptor::Finalize();
    MeshTools::Finalize();
}
```

22 à 36 de código da Figura 19. Nesta etapa de configuração, a classe para solucionar um sistema de equações, *ImplicitSystem* ou *TransientImplicitSystem*, deve ser instanciada para solução do método numérico. Também, nesta etapa da configuração são adicionadas as variáveis do sistema. No MeshTools é possível solucionar equações com mais de um grau de liberdade, como, por exemplo, na resolução da equação de Navier-Stokes (16) tridimensional em que existem 4 graus de liberdade: pressão e componentes x, y e z da velocidade. Além das variáveis, nesta etapa também devem ser adicionadas as condições de contorno, fazendo uso da classe *DirichletBoundary*, e no caso da solução de um sistema transiente também deve ser feita a adição da condição inicial do sistema través da classe *InitialCondition*. O acoplamento da condição de contorno ao solucionador de sistema de equações é feita através do método `add_dirichlet_boundary` e o acoplamento da condição inicial é feita através do método `add_initial_condition`.

Além da inserção das variáveis, condições de contorno e condição inicial, é necessário indicar ao objeto solucionador de sistema de equações o método de montagem do sistema de elementos finitos. Para isso deve ser implementada uma função, como a apresentada no Algoritmo 5. O acoplamento desta função no objeto solucionador de sistema de equações é feito na linha 30 de código da Figura 19. A escrita dos resultados é realizada tanto em arquivo VTK quanto através da interface Catalyst, a qual será mais detalhada na Seção 3.3. Para isso foram estipulados intervalos de escrita dos dados, sendo os dados brutos escritos a cada 50 passos no tempo e os dados processados pelo ParaView Catalyst escritos a cada 25 passos no tempo, como apresentado nas linhas 35 e 36 na Figura 19. A escrita dos dados em formato VTK e dos dados processados pelo Catalyst são realizadas respectivamente nas linhas 43 e 46 do código apresentado na Figura 19.

Algoritmo 5: Montagem do sistema de elementos finitos

Entrada: solucionador do sistema de equações
$qrule \leftarrow QGauss$
$fem \leftarrow FEMFunction$
Para $iel \in nelem$ **Faça**
  Instanciar a matriz K_e e o vetor independente F_e do elemento iel
  Obter pontos de integração e respectivos pesos da Quadratura Gaussiana
  Obter a função de forma para cada nó do elemento iel com a variável fem
  Obter a derivada da função de forma para cada nó do elemento iel com a variável fem
  Para $qponto \in qrule.pontos$ **Faça**
    Calcular parâmetros como velocidade, viscosidade, permeabilidade, ...
    Calcular o valor do termo de estabilização SUPG (para problemas convectivos dominantes)
    Para $i \in iel\_nós$ **Faça**
      Atualizar vetor F_i
      Para $j \in iel\_nós$ **Faça**
        Atualizar matriz K_{ij}
      Fim-Para
    Fim-Para
  Fim-Para
  Adicionar matriz elementar K_e na matriz global
  Adicionar vetor elementar F_e no vetor global
Fim-Para

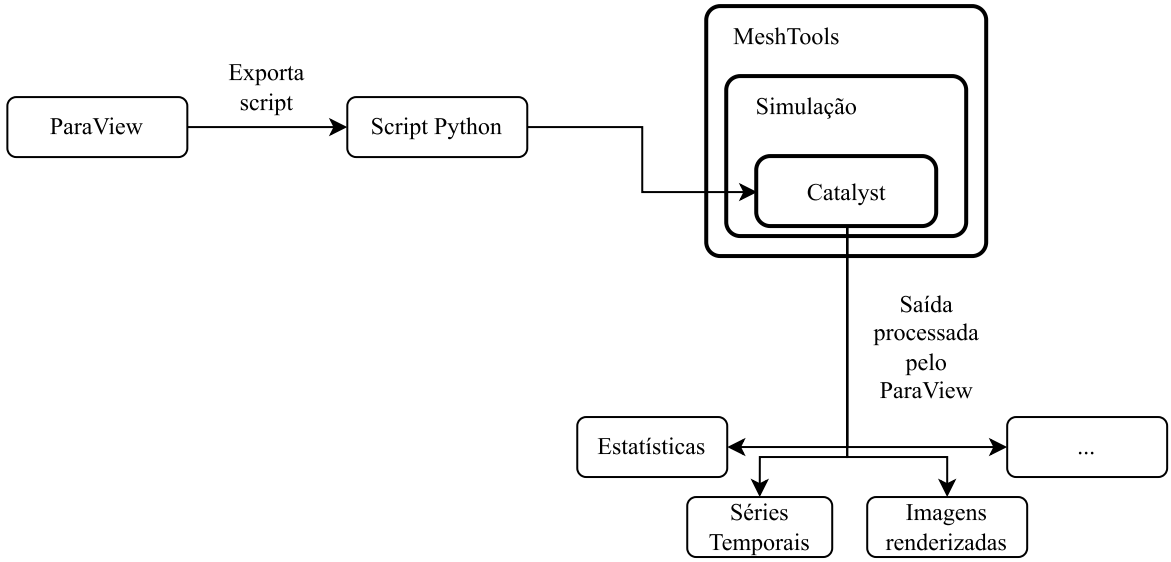
3.3 ESCRITA E VISUALIZAÇÃO IN SITU

MeshTools faz escrita dos resultados em paralelo utilizando o formato VTK (32). Para realizar esta escrita em paralelo, cada processo MPI escreve em disco o respectivo arquivo de dados de saída, no fomato `vtu`, referente à sua malha paralela. E também, o processo líder, de *rank* zero, além de escrever o arquivo de dados, escreve o arquivo de metadados, no formato `vtp`. O formato `vtu` é utilizado para dados pertencentes a um domínio não-estruturado, e o formato `vtp` utilizado na organização dos dados escritos em paralelo. VTK é um tipo de formato implementado pelo Visualization Toolkit (44), um *software open source* para manipulação e exibição de dados científicos, o qual fornece opções de escrita e leitura de dados em série ou paralelo para domínios estruturados, ou não-estruturados. É utilizado para a escrita dos resultados o formato para uma grade não-estruturada, baseado na sintaxe XML, uma sintaxe voltada para formatos de conjunto de dados topologicamente irregulares tendo em vista a irregularidade dos conjuntos de pontos e elementos em uma malha de elementos finitos.

O formato VTK é suportado nativamente pelo ParaView (2), um *software* de código aberto para visualização científica e processamento de dados. Tendo em vista a utilização do ParaView para visualização dos resultados obtidos através do MeshTools, o arcabouço implementado além de prover escrita de dados em paralelo, também tem

acoplado a si a interface de programação ParaView Catalyst (8). Através do ParaView Catalyst é possível realizar visualização *in situ*, ou seja, gerar visualizações para o dado de interesse no momento em que este é calculado pelo *software*. Para isto, o usuário deve exportar um *script* Python através do ParaView, como apresentado na Figura 20. Neste *script* é indicado como deseja-se visualizar os dados. Existem diversas formas de visualizar os dados brutos através dos diferentes filtros disponíveis no ParaView.

Figura 20 – Esquematização do funcionando da API ParaView Catalyst.



Dessa forma, com o ParaView Catalyst é possível visualizar os dados de saída do MeshTools, ou seja, os resultados da simulação de Elementos Finitos, em tempo de execução. Este tipo de visualização tem como vantagem a redução da quantidade de informações escritas no disco dado a possibilidade de aumentar o intervalo de escrita de dados em formato VTK substituindo pelo coprocessamento fornecido pelo ParaView Catalyst. Este tipo de visualização possibilita a escrita direta de imagens, tabelas, gráficos, entre outros, dispensando a necessidade de escrita recorrente do dado bruto da simulação. Dessa forma, além de reduzir o requisito de espaço em disco necessário para escrita dos dados também é decrementado o tempo operações de escrita em disco, o que é um gargalo em aplicações de alto desempenho (24).

Para o devido acoplamento da interface de programação ParaView Catalyst no MeshTools foi necessário implementar um adaptador responsável pela comunicação entre as estruturas de dados do MeshTools com as estruturas do ParaView. Esse adaptador implementa os métodos de inicialização da interface, o método para aplicação de execução e obtenção da visualização do dado e por fim o método de finalização da interface. Na etapa de inicialização do Catalyst é necessário indicar o *script* Python contendo informações de como os dados serão processados e visualizados. No método de execução, executa-se o *script* Python definido anteriormente nos dados gerando as visualizações. Nesta etapa deve ser indicado o passo no tempo a ser visualizado, as informações dos nós dos elementos

com suas coordenadas, as conectividades e tipo dos elementos utilizando a convenção de hierarquias Conduit Mesh Blueprint (1), além das variáveis a serem visualizadas. No método de finalização da interface é executada somente a função de finalização da biblioteca do Catalyst.

Na Figura 21 é apresentado trecho do método de execução do adaptador que acopla a estrutura de dados do Meshtools com o Catalyst. Das linhas 8 a 20 da Figura 21 ocorre o mapeamento das informações das coordenadas da malha do MeshTools para o Catalyst através do protocolo Conduit (33), a qual permite o acoplamento de dados entre pacotes in-core, serialização e tarefas de E/S. Nesta etapa são informados separadamente a posição x, y e z de cada nó da malha obtidos pelo *array* de coordenadas. Em seguida, nas linhas 22 a 32 são armazenados na estrutura do Catalyst a conectividade dos elementos da malha fazendo uso do arranjo de conectividades, além do tipo dos elementos e o número de nós delimitantes dos mesmos. Finalmente, da linha 34 a 44 são comunicados os resultados a serem visualizados. Para isso devemos fazer uso do sistema de equações do MeshTools a fim de obter o nome das variáveis e seus valores, possibilitando o armazenamento na estrutura do Catalyst. Importante destacar que nenhuma cópia de dados será realizada neste processo, mas apenas o mapeamento direto de memória.

Figura 21 – Exemplo do código referente ao método de execução do adaptador do Catalyst implementado no MeshTools

```
1  void Execute(int cycle, double time, ImplicitSystem *system)
2  {
3    conduit_cpp::Node exec_params;
4    auto channel = exec_params["catalyst/channels/grid"];
5    auto mesh = channel["data"];
6    ParallelMesh& meshtools_mesh = system->get_mesh();
7
8    double *coords_ptr   = meshtools_mesh.getCoordinatesData();
9    unsigned int nnodes = meshtools_mesh.get_n_nodes();
10
11   mesh["coordsets/coords/type"].set("explicit");
12   mesh["coordsets/coords/values/x"].set_external(
13         coords_ptr, nnodes, 0, 3*sizeof(double)
14   );
15   mesh["coordsets/coords/values/y"].set_external(
16         coords_ptr, nnodes, sizeof(double), 3*sizeof(double)
17   );
18   mesh["coordsets/coords/values/z"].set_external(
19         coords_ptr, nnodes, 2*sizeof(double), 3*sizeof(double)
20   );
21
22   auto ElemType = meshtools_mesh.get_mesh_element_type();
23   auto paraview_elem_type = ElemType->get_paraview_element_type();
24
25   mesh["topologies/mesh/elements/shape"].set(paraview_elem_type);
26   int nnoel = ElemType->get_n_nodes();
27
28   unsigned int *conn_ptr = meshtools_mesh.getElementConnectivityData();
29   unsigned int ncells = meshtools_mesh.get_n_elements();
30   mesh["topologies/mesh/elements/connectivity"].set_external(
31        conn_ptr, nnoel * ncells
32   );
33
34   auto fields = mesh["fields"];
35   int nvar = system->get_equation_manager().get_n_dofs();
36   for(int nv = 0; nv < nvar; ++nv)
37   {
38     std::string var_name = system->get_variable_name(nv);
39     fields[var_name + "/association"].set("vertex");
40     fields[var_name + "/topology"].set("mesh");
41     fields[var_name + "/volume_dependent"].set("false");
42     fields[var_name + "/values"].set_external(solution, nnodes,
43               nv*sizeof(double), nvar * sizeof(double));
44   }
45 }
```

4 RESULTADOS

Os resultados são apresentados para os dois componentes do MeshTools: Manipulador de Malha e Solucionador de Elementos Finitos. Para o componente de manipulação de malhas são apresentados resultados referentes à decomposição do domínio, coloração dos elementos e reordenação dos nós. Já para o componente responsável pela solução do Método dos Elementos Finitos são apresentados resultados que demonstram a corretude da solução do método numérico e sua escalabilidade paralela.

4.1 MANIPULAÇÃO DE MALHAS

Serão demonstradas nesta seção os resultados relacionados ao particionamento de domínio, coloração dos elementos e reordenação dos nós da malha.

Para apresentar os resultados obtidos no componente de manipulação de malhas são utilizadas duas malhas tridimensionais aplicáveis no estudo de dinâmica de fluidos. A primeira é ilustrada na Figura 22, a qual representa a estrutura de uma artéria. Esta malha é composta por 188.273 nós e 1.005.568 elementos tetraédricos. A segunda malha é apresentada na Figura 23. Nela é definido um volume no entorno da superfície de um carro, possibilitando, por exemplo, o estudo da aerodinâmica deste automóvel. Esta malha é composta por 287.724 nós e 1.442.816 elementos tetraédricos.

Figura 22 – Malha para estudos de dinâmica de fluidos em artéria.

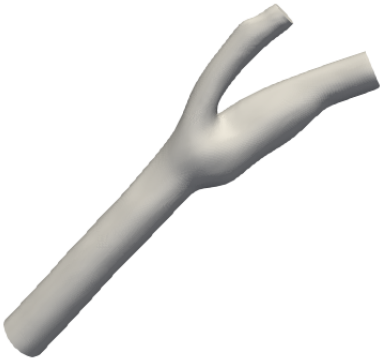
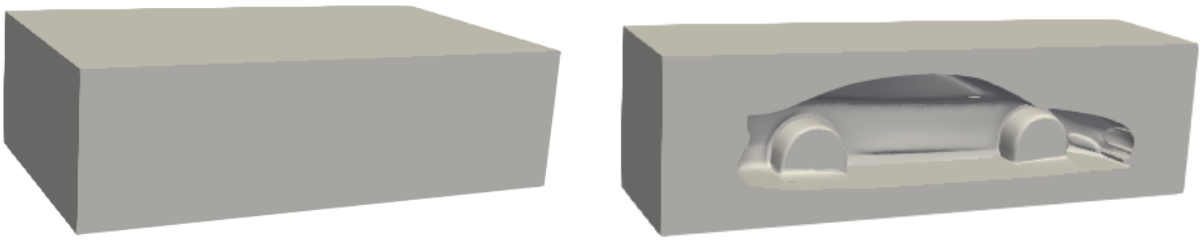


Figura 23 – Malha para estudos de aerodinâmica em um carro. Em (a) a geometria completa e em (b) a geometria seccionada ao meio.



(a) (b)

4.1.1 Particionamento de domínio

Referente à técnica de particionamento de domínio foram aplicadas 4, 8 e 12 partições nas malhas a fim de demonstrar o resultado da decomposição realizada pela biblioteca METIS. O número de partições é equivalente ao número de processos MPI em que o MeshTools é executado. Na Figura 24 é apresentado os diferentes particionamentos aplicados na malha da artéria. Já na Figura 25 são apresentadas as diferentes partições para a malha do volume no entorno do carro. Os resultados estão conforme o esperado ao obter o número de partições iguais ao definido nas execuções de teste.

Figura 24 – Diferentes partições para a malha da artéria. Em (a) aplicada 4 partições, (b) 8 partições e (c) 12 partições

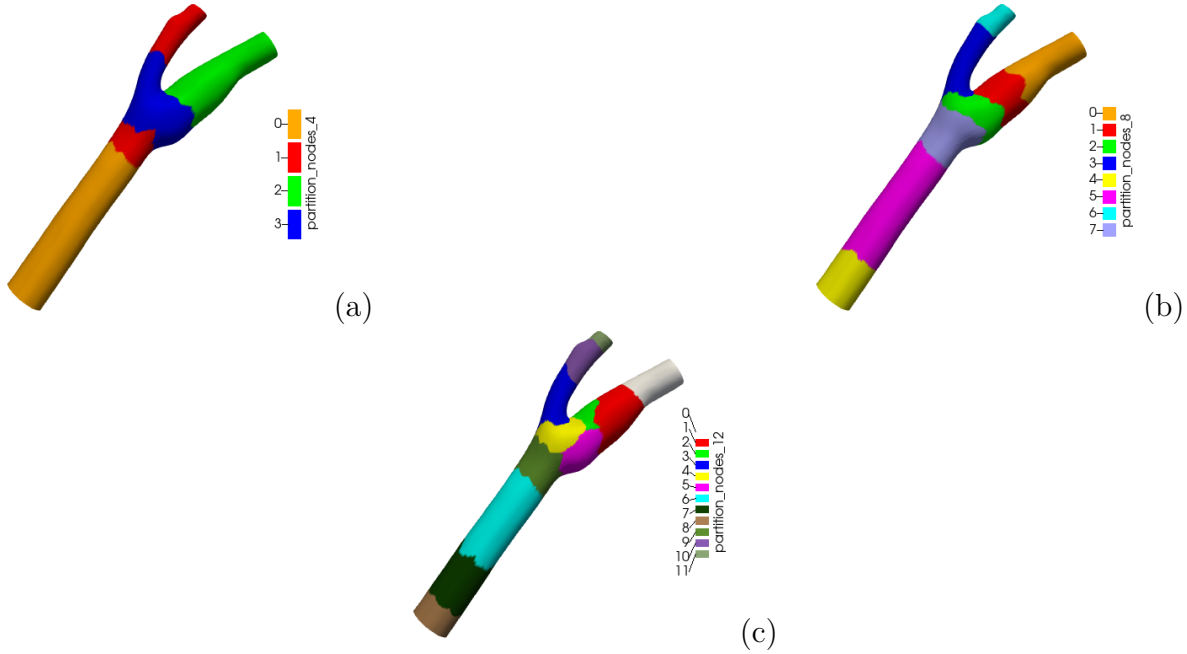
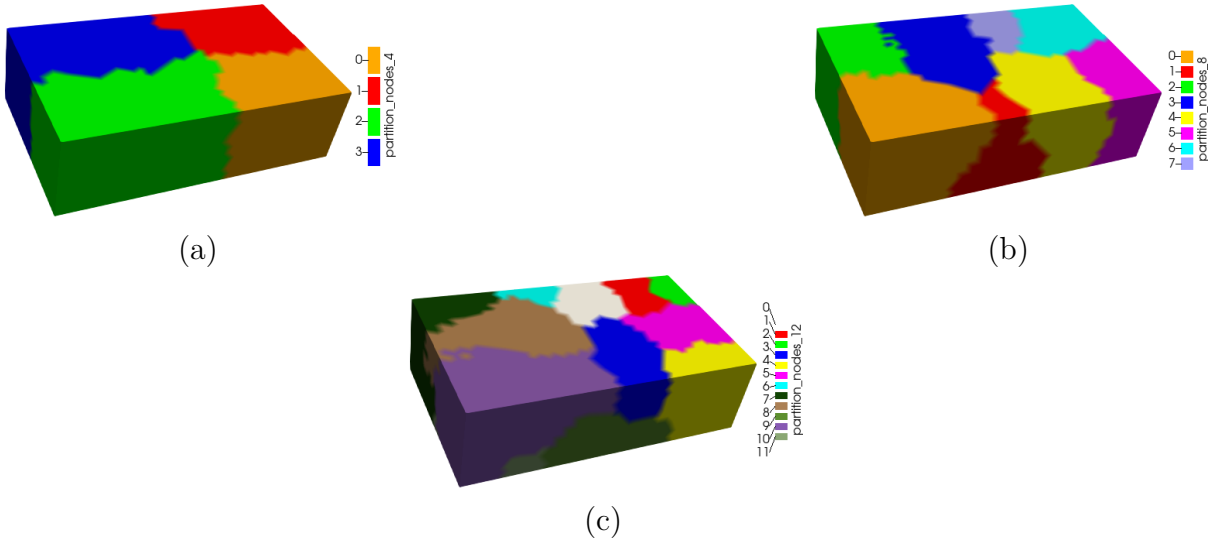


Figura 25 – Diferentes partições para a malha do volume ao entorno do carro. Em (a) aplicada 4 partições, (b) 8 partições e (c) 12 partições



4.1.2 Coloração de elementos

Para os resultados da técnica de coloração dos elementos são apresentadas imagens para visualização das cores dos elementos de ambas as malhas de exemplo nas Figuras 26 e 27. Neste caso as imagens servem somente para ilustrar as diferentes cores presentes nos elementos das malhas. É possível visualizar que não existem elementos adjacentes com mesma cor, o que foi verificado também por meio de algoritmos.

Figura 26 – Visualização da malha de uma artéria com seu esquema de 83 cores calculado através do algoritmo *Greedy*.

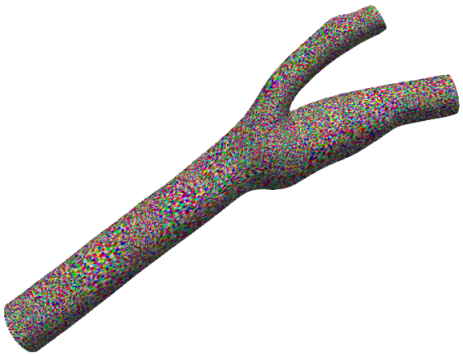
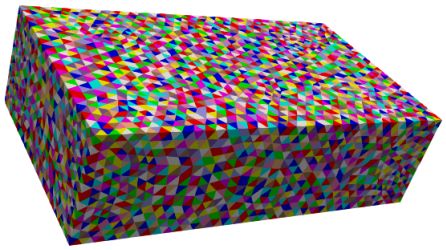


Figura 27 – Visualização da malha do volume ao entorno de um carro com seu esquema de 78 cores calculado através do algoritmo *Greedy*.



Além disso, na Tabela 1 são apresentados resultados indicando o número de elementos por cor para os algoritmos *Greedy*, e *Blocked* com um limite de 16384 elementos por cor para a malha da Artéria. Neste caso, o algoritmo *Greedy* calculou 83 cores e o algoritmo *Blocked* 111 cores. Para o resultado de ambos os algoritmos são indicados o número de elementos por cor em um conjunto específico de cores para melhorar a visualização dos dados. Os resultados foram apresentados de 10 em 10 cores a fim de ilustrar devidamente todas as cores calculadas no texto.

Tabela 1 – Número de elementos por cor para os algoritmos *Greedy* e *Blocked* aplicados na malha da Artéria.

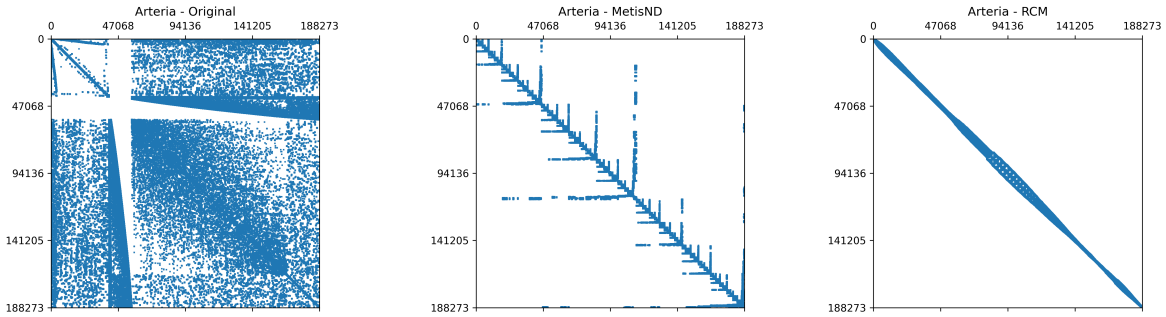
Cor	1	11	21	31	41	51	61	71	81	91	101	111
Greedy	39135	35559	30342	13273	343	89	51	29	8	-	-	-
Blocked	16384	16384	16384	16384	16384	16384	16384	224	84	51	23	4

Pode-se perceber que o algoritmo *Greedy* gera uma coloração não uniforme, resultando em um número superior de elementos nas primeiras cores em detrimento às últimas cores. Isto ocorre por conta do conceito do algoritmo em que este tenta colorir os elementos com a cor mínima. Já, no algoritmo *Blocked* pode-se visualizar que o número de elementos por cor é mais uniforme em relação ao algoritmo *Greedy*, porém resultando em um número total de cores maior. Através do algoritmo *Blocked* é possível indicar o número máximo de elementos por cor como o número de elementos armazenáveis na memória *cache* aprimorando a eficiência do uso da hierarquia de memória.

4.1.3 Reordenação nodal

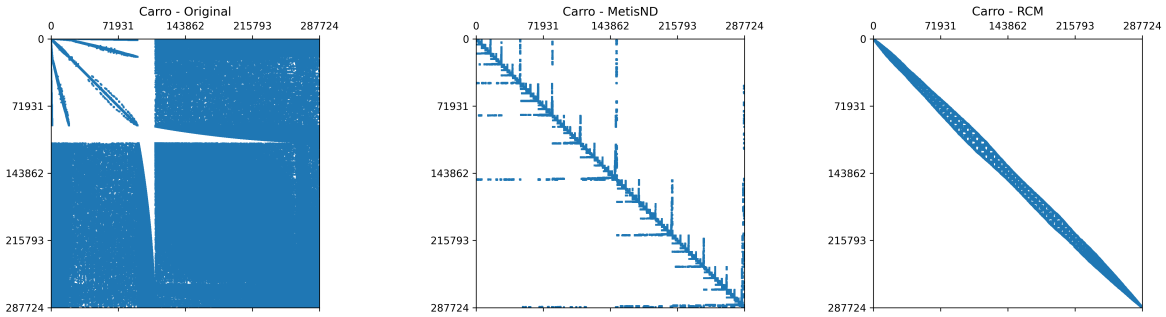
Como métrica de desempenho da técnica de reordenação dos nós da malha foi analisada a largura de banda das matrizes de adjacência das malhas de teste. Nas Figuras 28 e 29 temos as matrizes de adjacências das malhas da Artéria e do Carro, respectivamente. Nestas figuras são apresentadas a matriz de adjacências original das malhas, a matriz após a aplicação do algoritmo de reordenação nodal METIS_ND e também após a aplicação da reordenação RCM.

Figura 28 – Matrizes de adjacência das malhas da Artéria em (a) original, (b) após aplicação do algoritmo de reordenação nodal METIS_ND e (c) após aplicação do algoritmo RCM



(a) (b) (c)

Figura 29 – Matrizes de adjacência das malhas do Carro em (a) original, (b) após aplicação do algoritmo de reordenação nodal METIS_ND e (c) após aplicação do algoritmo RCM



(a) (b) (c)

Na Tabela 2 são apresentadas as larguras de banda referentes às matrizes de adjacência de ambas as malhas exemplares com a aplicação dos diferentes algoritmos de reordenação além da largura de banda referente à matriz original.

Tabela 2 – Larguras de banda para ambas malhas exemplares na ordenação original e após aplicação dos algoritmos de reordenação nodal METIS_ND e RCM.

	Original	METIS_ND	RCM
Artéria	187095	141368	5089
Carro	287124	237572	10128

Através das Figuras 28 e 29 e da Tabela 2 pode-se visualizar que a aplicação dos algoritmos de reordenação reduziu de forma significativa a largura de banda das matrizes de adjacência referentes à ambas as malhas teste. Além disso, pode-se concluir que, em especial, o algoritmo RCM performou melhor em relação ao algoritmo METIS_ND. De uma forma geral, para as duas malhas teste o algoritmo RCM reduziu a largura de banda em cerca de 97% e o algoritmo METIS_ND reduziu em cerca de 20%.

4.2 SOLUCIONADOR DE ELEMENTOS FINITOS

Os resultados referentes ao componente solucionador de elementos finitos foram obtidos através da execução de testes padrão bem estabelecidos na literatura. Para isto foram executados testes estacionários e transientes bidimensionais e tridimensionais como casos de teste. Para o teste estacionário foi solucionada a equação de Poisson apresentada na Equação (2.5) e, para os testes transientes foram executados problemas modelados através da equação de movimento convectivo-difusivo apresentado na Equação (2.27). Para esta equação são realizados três testes: rotação de um pulso gaussiano, apresentado em (49), alongamento de um disco, definido por (10) e alongamento de uma esfera (13).

Em todas as soluções foram utilizados o solucionador linear e pré-condicionador padrões da biblioteca PETSc. Ou seja, a solução do sistema linear é realizada através do método GMRES reinicializado (42) e o pré-condicionador utilizado é o LU incompleto (19) a fim de melhorar a taxa de convergência do método. Para o método GMRES são utilizados 35 vetores na base de Krylov, o número padrão da biblioteca PETSc. Além disso, para a solução do sistema linear foi limitado o número máximo de iterações em 10^5 e aplicada uma tolerância da solução do sistema linear de 10^{-8}. Para os sistemas não-lineares, no caso dos problemas de alongamento de um disco e alongamento de uma esfera, foi limitado o número máximo de iterações não-lineares em sete e aplicada tolerância de 10^{-4}.

Estes testes foram executados em um *Workstation* com processador Intel(R) Core(TM) i7-12700KF de 12ª geração e 64GB de memória RAM. Para os estudos de escalabilidade paralela são utilizadas as métricas de desempenho Speedup e Eficiência, sendo definidas na Equação (4.1) e Equação (4.2) respectivamente.

$$S_p = \frac{T_s}{T_p} \tag{4.1}$$

$$E_p = \frac{S_p}{p} \tag{4.2}$$

Ou seja, o Speedup (S_p) é definido como o tempo de execução em série (T_s) dividido pelo tempo de execução em paralelo (T_p). E a Eficiência (E_p) definida como a divisão entre o Speedup e o número de processos paralelos (p).

4.2.1 Equação de Poisson

A fim de obter resultados em um teste estacionário, foi resolvida a equação de Poisson, dada pela Equação (2.5) com $\kappa = 1.0$ e aplicação de contorno do tipo Dirichlet. Este teste foi realizado para a verificação da corretude nos resultados para os três tipos de elementos implementados no MeshTools: elementos bidimensionais TRI3, QUAD4 e elemento tridimensional TET4. Dessa forma, o teste foi executado em um domínio bidimensional e tridimensional. Para o estudo foi utilizado um tipo de solução manufaturada em um domínio espacial $\Omega \in [0,1] \times [0,1]$, de forma que desejamos obter

$$u(x,y) = 100xy(1-x)(1-y), \tag{4.3}$$

então, a solução exata deve ser o laplaciano de u, ou seja,

$$s(x,y) = 200y(y-1) + 200x(x-1), \tag{4.4}$$

e o valor de u_D segue a solução exata.

O resultado da solução da equação bidimensional é apresentada na Figura 30. Este resultado está conforme a solução u esperada apresentada na Equação (4.3), pode-se verificar que o máximo da equação, obtida em $(x,y) = (0.5, 0.5)$ resulta em $u(0.5, 0.5) = 6.25$ como o apresentado na simulação.

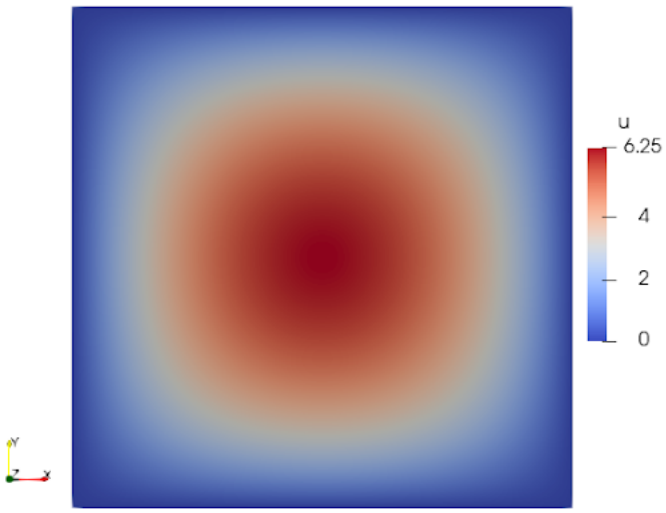
Além disso, o problema foi executado com diferentes tamanhos de malha a fim de verificar a taxa de convergência com o erro da norma L_2 e erro da norma H_1, com ambas as normas sendo definidas na Equação (4.5).

$$\begin{aligned} ||u - \overline{u}||_{L_2} &= \left(\int_\Omega |u - \overline{u}|^2 d\Omega \right)^{(1/2)} = O(h^{k+1}) \\ ||u - \overline{u}||_{H_1} &= \left(\int_\Omega |\nabla u - \nabla \overline{u}|^2 d\Omega \right)^{1/2} = O(h^k) \end{aligned} \tag{4.5}$$

onde $\overline{u}$ é definida como a solução exata do problema e k é referente ao grau do polinômio interpolador do elemento.

Para o cálculo dos erros de ambas as normas foi feito o somatório do erro de cada elemento da malha. Pelo fato do MeshTools implementar somente elementos lineares,

Figura 30 – Resultado para a equação de Poisson bidimensional.



é esperada que a norma L_2 seja de ordem 2 e norma H_1 seja de ordem 1. O problema foi resolvido no MeshTools, como mencionado anteriormente, em malhas com elementos triangulares (TRI3) e quadrangulares (QUAD4) ambos de ordem linear e os resultados são apresentados na Tabela 3. Os erros em escala logarítmica são apresentados na Figura 31 onde pode-se observar taxa de convergência 2 e 1 para as normas L2 e H1 respectivamente.

Tabela 3 – Problema de Poisson - Norma dos erros para malhas bidimensionais com elementos TRI3 e QUAD4 para diferentes discretizações espaciais.

	TRI3		QUAD4	
Grid	Norma L_2	Norma H_1	Norma L_2	Norma H_1
8×8	1.82×10^{-1}	1.92101	5.12×10^{-2}	1.80329
16×16	4.64×10^{-2}	0.953854	1.28×10^{-2}	0.899354
32×32	1.17×10^{-2}	0.475954	3.19×10^{-3}	0.44935
64×64	2.91×10^{-3}	0.237853	7.97×10^{-4}	0.224631
128×128	7.28×10^{-4}	0.118911	1.99×10^{-4}	0.11231
256×256	1.82×10^{-4}	0.059454	5.00×10^{-5}	0.056154

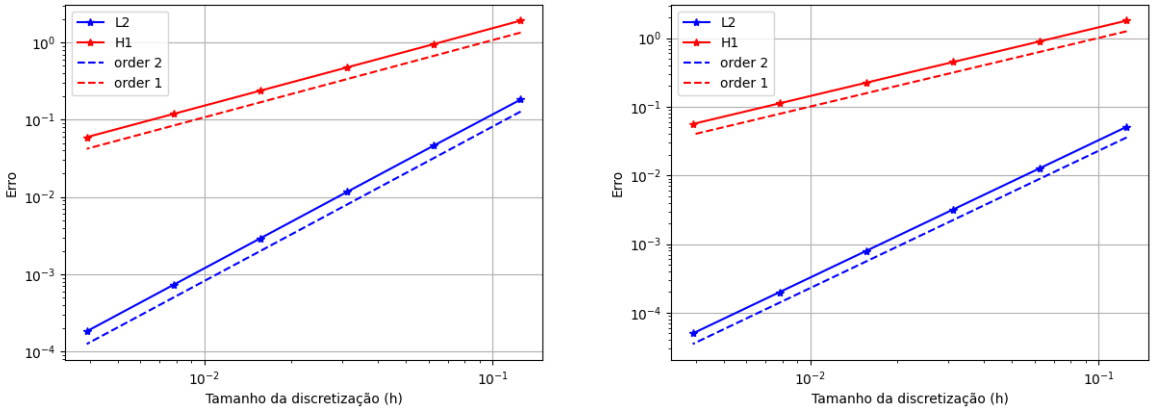
Para os testes em uma malha tridimensional com elementos tetraédricos lineares (TET4) foi utilizada a formulação manufaturada apresentada na Equação (4.3) expandida para a terceira dimensão, resultando em

$$u(x, y, z) = 100xyz(1 - x)(1 - y)(1 - z), \tag{4.6}$$

com s, o laplaciano de u definido por

$$s(x, y, z) = 200y(y - 1) + 200x(x - 1) + 200z(z - 1). \tag{4.7}$$

Figura 31 – Taxa de convergência para o problema de Poisson em malhas com elementos (a) TRI3 e (b) QUAD4.



(a) (b)

Foram executadas simulações com diferentes tamanhos de malhas a fim de obter a taxa de convergência para o elemento tridimensional. Estes resultados de convergência são apresentados na Tabela 4 e na Figura 32.

Tabela 4 – Problema de Poisson - Norma dos erros para malhas tridimensionais com elementos TET4 para diferentes discretizações espaciais.

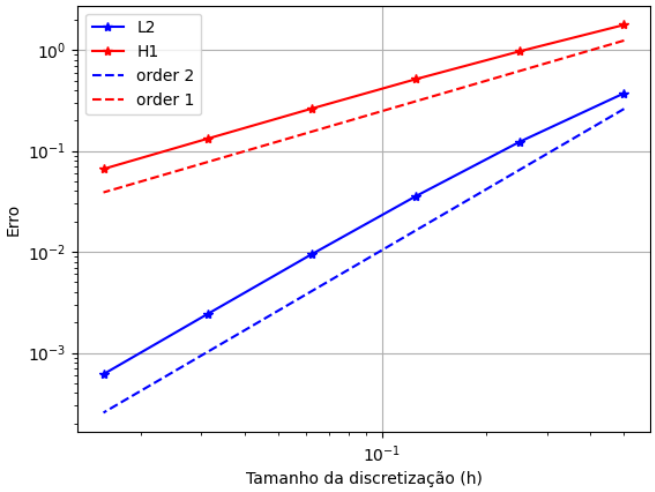
Grid	Norma L2	Norma H1
$2 \times 2 \times 2$	3.73×10^{-1}	1.77611
$4 \times 4 \times 4$	1.24×10^{-1}	0.973478
$8 \times 8 \times 8$	3.56×10^{-2}	0.514756
$16 \times 16 \times 16$	9.48×10^{-3}	0.262878
$32 \times 32 \times 32$	2.42×10^{-3}	0.132355
$64 \times 64 \times 64$	6.14×10^{-4}	0.0663151

Os resultados obtidos através testes bidimensionais e tridimensionais da equação de Poisson, tanto pela solução da equação quanto pelas taxas de convergência, demonstram resolução correta do problema através do Método dos Elementos Finitos.

4.2.2 Rotação do Pulso Gaussiano

Para o primeiro teste transiente, a rotação de um pulso gaussiano, tem-se um pulso em formato gaussiano transladado, com difusão próxima de zero, em formato circular ao redor do centro de um domínio quadrado. A equação é definida em um domínio espacial $[0, 10] \times [0, 10]$ e domínio temporal $]0, 2\pi]$. A condição inicial do problema é dada pelo pulso posicionado em $(x, y) = (5.0, 7.5)$, ou seja, a função que representa a condição inicial do problema é dada por $u_0(x, y) = e^{-0.5r}$ tal que $r = (x - 5)^2 + (y - 7.5)^2$ e, para condição de contorno é aplicada condição de Dirichlet nula. A velocidade de advecção do pulso é dada por

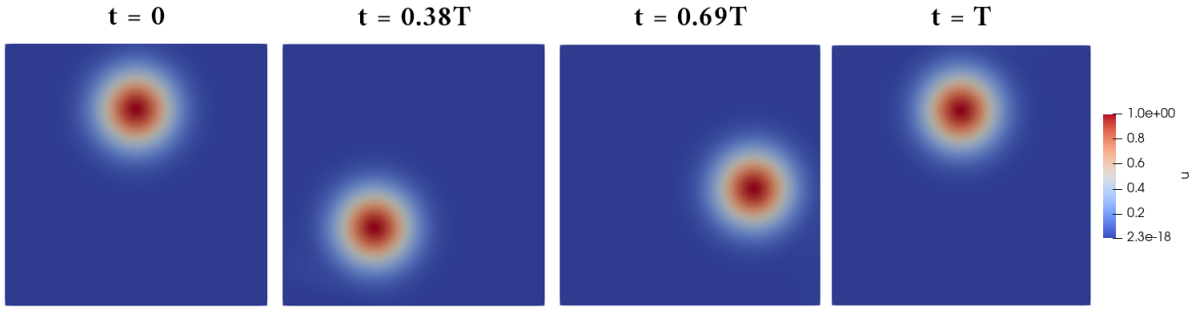
Figura 32 – Taxa de convergência para o problema de Poisson em malhas com elementos TET4.



$$\begin{aligned} v_x &= -y + 5.0 \\ v_y &= x - 5.0 \end{aligned} \tag{4.8}$$

e taxa de difusão $\kappa = 10^{-8}$. No tempo final da simulação, em $T = 2\pi$, o pulso retorna à posição inicial. O problema foi resolvido em uma malha com cerca de 76.000 nós e 150.000 elementos triangulares de ordem linear com discretização temporal $\Delta t = 0.0025$. Na Figura 33 é apresentado o resultado obtido através do MeshTools.

Figura 33 – Solução do teste padrão da rotação de um cone gaussiano.



Além disso, foi feita uma análise de escalabilidade paralela do MeshTools para este problema. Os resultados são apresentados na Tabela 5.

O resultado encontrado no MeshTools está segundo o encontrado no artigo base (49) demonstrando a corretude da solução. Além disso, a escalabilidade paralela do problema é satisfatória tendo em vista que o Speedup obtido foi superlinear. Isto indica que o tempo de execução se reduziu em uma proporção maior que o número de processos que executaram o problema em paralelo. O Speedup superlinear pode ser explicado pelo tamanho da memória *cache* do *Workstation* em comparação ao tamanho da malha solucionada no

Tabela 5 – Teste Rotação do Pulso Gaussiano - Análise de desempenho para malha composta por elementos TRI3 com diferentes números de processos MPI paralelos

# Processos MPI	Tempo total (s)	Speedup	Eficiência
1	2828.0		
2	937.0	3.018	1.509
4	530.0	5.329	1.332
8	306.0	9.233	1.154

problema. Devido ao número reduzido de nós da malha, o número de *cache hit* é maior que o número de *cache miss*, que por consequência resulta um tempo de execução abaixo do esperado com a divisão de trabalho entre os processos paralelos.

4.2.3 Alongamento do Disco

O segundo teste transiente, o teste do alongamento de um disco, foi inicialmente apresentado por (10). Neste problema uma concentração em formato de disco com raio 0.15 posicionado inicialmente em $(x, y) = (0.5, 0.75)$ é alongado através da velocidade de advecção da concentração pelo domínio em formato espiral e retornada à sua posição inicial no final do tempo estipulado. O campo de velocidades de advecção da concentração de interesse é dado por

$$v_x = \sin(2\pi y)\sin^2(\pi x) \tag{4.9}$$

$$v_y = -\sin(2\pi x)\sin^2(\pi y) \tag{4.10}$$

O campo de velocidades causa um alongamento no disco em formato de espiral. Para analisar o erro, (34) recomenda multiplicar o campo de velocidades pela função

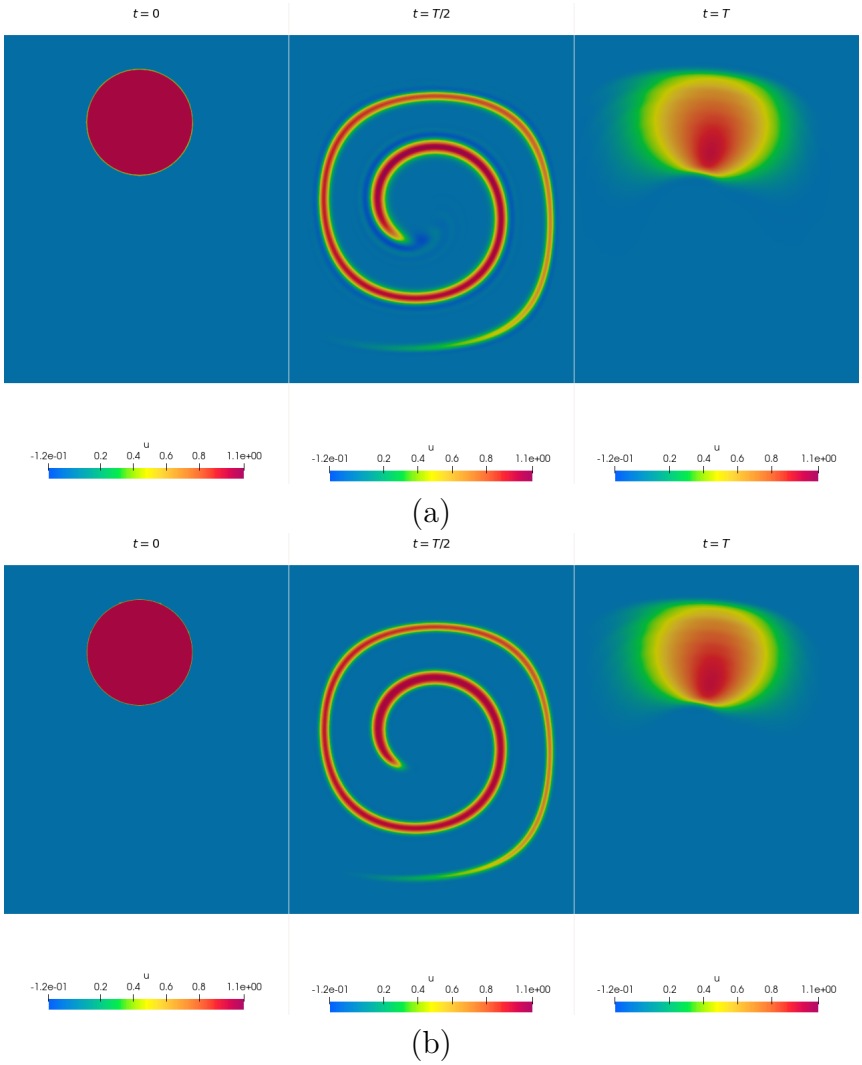
$$g(x) = \cos(\pi t/T) \tag{4.11}$$

a qual permite o fluido retornar à sua posição inicial ao final do tempo T. Neste trabalho foi utilizado $T = 8$ e aplicado discretização temporal $\Delta t = 0.0025$. Além disso, neste problema foi utilizada a formulação estabilizada com SUPG (12) e CAU (5) em um fator de 10% do calculado. Este fator aplicado no termo CAU foi estipulado a fim de reduzir a não-linearidade do sistema a ser resolvido.

As Figuras 34(a) e 34(b) apresentam a solução para malhas com elementos TRI3 e QUAD4 respectivamente nos tempos $t = 0$, $t = T/2$ e $t = T$. A malha de elementos TRI3 tem cerca de 33.000 nós e 66.000 elementos e já a malha de elementos QUAD4 tem cerca de 56.000 nós e 57.000 elementos. É observado que a solução através da malha com elementos QUAD4 tem menos oscilações que a malha com elementos TRI3. Além disso, a

escalabilidade paralela do MeshTools foi analisada para este problema e os resultados são apresentados na Tabela 6.

Figura 34 – Problema de alongamento de um disco:(a) solução TRI3 e (b) solução QUAD4



(a)

(b)

Tabela 6 – Teste Alongamento do Disco - Análise de desempenho para malha composta por elementos TRI3 com diferentes números de processos MPI paralelos

# Processos MPI	Tempo total (s)	Speedup	Eficiência
1	979.3		
2	464.3	2.109	1.055
4	182.7	5.360	1.340
8	104.8	9.344	1.168

Os resultados encontrados para o problema de alongamento do disco estão equivalentes ao apresentado na literatura base (14) demonstrando que a solução realizada através do MeshTools está correta. Também, a escalabilidade paralela na solução deste problema é satisfatória para os três casos paralelos testados, com 2, 4 e 8 processos MPI. Neste caso, os resultados referentes à eficiência paralela apresentaram a mesma superlinearidade no Speedup encontrados na Seção **4.2.2**.

Para este teste, além da verificação da corretude e da escalabilidade paralela, também foi calculado o tempo de execução das etapas de leitura da malha, integração da equação diferencial e escrita dos resultados tanto em arquivo VTK quanto em imagens através do Catalyst. Esta análise é apresentada na Tabela 7 e foi realizada a fim de verificar a influência da utilização do Catalyst no tempo total do programa. Neste exemplo foi estabelecido um intervalo de geração de imagem através do co-processamento do Catalyst a cada 10 passos no tempo e escrita em formato VTK dos dados brutos a cada 20 passos no tempo.

Tabela 7 – Teste Alongamento do Disco - Tempo de execução de cada etapa da solução da equação.

Etapa	Tempo (s)	Fator (%)
Leitura	12.07	4.081
Catalyst	0.11	0.038
Escrita VTK	0.24	0.080
Integração	283.49	95.802
Total	295.91	100.000

Através da Tabela 7 pode-se visualizar que o tempo de geração das imagens totalizou 0.11 segundos, o tempo para escrita dos dados brutos totalizou 0.24 segundos. Além de gerar os resultados filtrados em menos tempo, o requisito de disco também é reduzido pelo fato da imagem no caso desse teste ocupar cerca de 30 Kilobytes, o dado bruto em VTK ocupa 3.4 Megabytes, ou seja, cerca de duas ordens de grandeza maior.

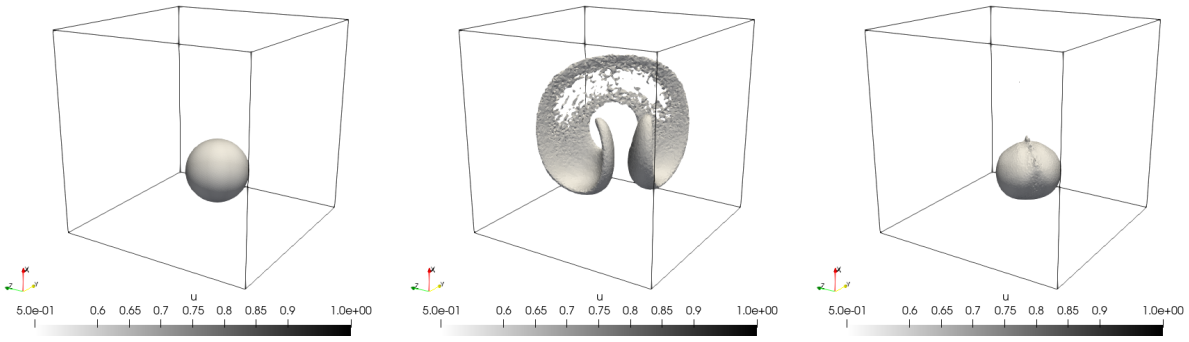
4.2.4 Alongamento da Esfera

A versão tridimensional do problema de alongamento de um disco também foi resolvida pelo MeshTools, aplicando uma malha composta por 744.846 nós e 4.456.944 elementos tetraédricos (TET4). Para este problema é dado o nome de alongamento de uma esfera, apresentado em (13). Os resultados para os tempos $t = 0$, $t = T/2$ e $t = T$ são apresentados na Figura 35.

No momento de maior alongamento da esfera, ou seja, no instante $t = T/2$ apresentando na Figura 35b, é perceptível a ausência de concentração em uma região da malha. Isto ocorre pelo fato do problema ter sido resolvido em uma malha grosseira, de forma que, nesta região da malha com maior alongamento da concentração, a camada do fluido se torna tão fina que os nós dos elementos referentes não são capazes de calcular a concentração devida. A fim de visualizar claramente o movimento da concentração no interior do domínio do problema, foi aplicado um filtro para ficar aparente somente a região de interesse cujo valor da variável é conhecido.

Além da visualização do movimento do fluido, também foi calculado a perda de massa entre o primeiro e último passos no tempo. Para isto foi aplicado o filtro de

Figura 35 – Solução do problema de alongamento de uma esfera em (a) $t = 0$, (b) $t = T/2$ e (c) $t = T$.



(a) (b) (c)

integração do ParaView no volume visualizado e assim, calculado uma perda de 33% da massa. Seria possível reduzir essa perda de massa da concentração de interesse com o aumento do nível de refinamento da malha.

Os resultados deste problema também estão conforme o encontrado na literatura de referência, (13) demonstrando corretude na solução. Para este teste também foi analisado o tempo de execução das principais etapas de solução da equação, a qual é apresentada na Tabela 8. Pode-se observar que, neste caso, a geração das visualizações in situ, através do Catalyst, levou aproximadamente 209 segundos, enquanto que a escrita dos dados brutos em formato VTK levou 1.84 segundos. Esse aumento no tempo de execução do Catalyst na Tabela 8 se da por conta da utilização de filtros mais complexos para extrair corretamente o movimento da concentração.

Tabela 8 – Teste Alongamento da Esfera - Tempo de execução de cada etapa da solução da equação.

Etapa	Tempo (s)	Fator (%)
Leitura	0.14	0.0006
Catalyst	209.01	0.9168
Escrita VTK	1.84	0.0081
Integração	22,586.00	99.0745
Total	22,796.99	100.0000

Para este problema foi estipulado um intervalo de escrita dos dados brutos em arquivo VTK a cada 50 passos no tempo e a geração de imagens através do Catalyst a cada 25 passos no tempo. Com estes intervalos foram gerados no total cerca de 1.2 Gigabytes de dados escritos em arquivo VTK e 3.3 Megabytes de dados escritos em imagens através do Catalyst. Tendo em vista que a proporção de arquivos escritos em VTK é o dobro das imagens, devido aos intervalos estipulados, caso o co-processamento do Catalyst não fosse aplicado e houvesse a necessidade manter a escrita dos dados a cada 25 passos no tempo, o dobro de espaço em disco seria necessário para escrita de todos os arquivos VTK. Em

problemas de maior escala com aplicação de malhas mais refinadas, essa possibilidade de escrita dos resultados pós-processados se torna interessante.

5 CONCLUSÃO

Neste trabalho foi apresentado o MeshTools, um arcabouço computacional para manipulação de malhas de elementos finitos e solução destas em paralelo. Este arcabouço é armazenado no seguinte repositório de código: `gitlab.com/camata/meshtools/`. MeshTools pode ser utilizado em sistemas computacionais massivamente paralelos como *clusters*, ou em um laptop pessoal utilizando ao máximo do paralelismo em CPU. Para a solução do sistema linear advindo de um problema de elementos finitos, foi acoplado ao MeshTools a biblioteca PETSc a qual disponibiliza diversos métodos para solução de sistemas lineares e pré-condicionadores. Através do MeshTools é possível aplicar técnicas de computação de alto desempenho na malha de entrada, possibilitando sua resolução em um sistema compartilhado e/ou distribuído através do particionamento de domínio e coloração elementar. Também, é possível otimizar a localidade dos dados da malha na memória incrementando o uso da memória *cache* através da reordenação nodal. Pelo MeshTools é possível aplicar estas técnicas de otimização na malha de interesse e escrevê-la em um arquivo formato VTK a fim de utilizá-la em outro *software* para solução da mesma. Ao solucionar um problema de Elementos Finitos através do MeshTools é possível obter os resultados da simulação em imagens, tabelas ou outros formatos além dos dados brutos através do co-processamento, ou visualização *in situ*, via interface de programação ParaView Catalyst.

Para demonstrar a corretude e eficiência do MeshTools foram executados testes padrão bem estabelecidos na literatura. Através destes resultados é possível concluir que a solução dos problemas de elementos finitos, tanto bidimensionais quanto tridimensionais, são realizados corretamente e com eficiência paralela adequada. Ainda é interessante solucionar um problema de elementos finitos através do MeshTools em um sistema computacional de larga-escala para verificar sua eficiência para um número maior de processos MPI. Além disso, é de interesse verificar a eficiência da solução do MeshTools em um sistema híbrido fazendo uso de paralelismo em memória distribuída e compartilhada. Através da PETSc é possível paralelizar a solução do sistema linear em *threads*, dessa forma uma malha particionada no MeshTools e distribuída em processos MPI pode ter a eficiência paralela da sua solução incrementada fazendo uso também do paralelismo em memória compartilhada.

Ainda, é desejável implementar no MeshTools uma ferramenta de refinamento paralelo de malha diminuindo os requisitos de memória do sistema onde é executado. Com esta ferramenta seria possível utilizar uma malha grosseira como entrada do problema e distribuí-la entre os processos. Em seguida, cada processo faria o refinamento da sua malha local com o auxílio de um arquivo que informasse as curvas da geometria desejada. Também é desejado implementar a solução de equações de Navier-Stokes (47) aumentando a gama de problemas solucionáveis através do MeshTools. E por fim é de interesse implementar,

através do ParaView Catalyst, um sistema de orientação (*user steering*) como apresentado em (51). Fazendo uso deste sistema, o usuário é capaz de orientar a simulação ajustando os parâmetros de entrada enquanto a simulação está em andamento de modo, por exemplo, a contornar possíveis divergências numéricas ao longo da simulação.

REFERÊNCIAS

1 Mesh blueprint - conduit documentation. `https://llnl-conduit.readthedocs.io/en/latest/blueprint_mesh.html`, Acessado: 25/11/2023.

2 Ahrens, J., Geveci, B., Law, C., Hansen, C., and Johnson, C. 36-paraview: An end-user tool for large-data visualization. *The visualization handbook 717* (2005), 50038–1.

3 Aiken, H. *Proposed automatic calculating machine*. Springer Berlin Heidelberg, Berlin, Heidelberg, 1982, pp. 195–201.

4 Alnæs, M., Blechta, J., Hake, J., Johansson, A., Kehlet, B., Logg, A., Richardson, C., Ring, J., Rognes, M. E., and Wells, G. N. The fenics project version 1.5. *Archive of numerical software 3*, 100 (2015).

5 Alvarez Henao, C. *Um Estudo sobre Operadores de Captura de Descontinuidades para Problemas de Transporte Advectivos*. PhD thesis, 04 2004.

6 Anderson, E., Bai, Z., Bischof, C., Blackford, S., Demmel, J., Dongarra, J., Du Croz, J., Greenbaum, A., Hammarling, S., McKenney, A., and Sorensen, D. *LAPACK Users' Guide*, third ed. Society for Industrial and Applied Mathematics, Philadelphia, PA, 1999.

7 Arndt, D., Bangerth, W., Davydov, D., Heister, T., Heltai, L., Kronbichler, M., Maier, M., Pelteret, J.-P., Turcksin, B., and Wells, D. The deal. ii finite element library: Design, features, and insights. *Computers & Mathematics with Applications 81* (2021), 407–422.

8 Ayachit, U., Bauer, A., Geveci, B., O'Leary, P., Moreland, K., Fabian, N., and Mauldin, J. Paraview catalyst: Enabling in situ data analysis and visualization. In *Proceedings of the First Workshop on In Situ Infrastructures for Enabling Extreme-Scale Analysis and Visualization* (New York, NY, USA, 2015), ISAV2015, Association for Computing Machinery, p. 25–29.

9 Balay, S., Abhyankar, S., Adams, M. F., Benson, S., Brown, J., Brune, P., Buschelman, K., Constantinescu, E. M., Dalcin, L., Dener, A., Eijkhout, V., Faibussowitsch, J., Gropp, W. D., Hapla, V., Isaac, T., Jolivet, P., Karpeev, D., Kaushik, D., Knepley, M. G., Kong, F., Kruger, S., May, D. A., McInnes, L. C., Mills, R. T., Mitchell, L., Munson, T., Roman, J. E., Rupp, K., Sanan, P., Sarich, J., Smith, B. F., Zampini, S., Zhang, H., Zhang, H., and Zhang, J. PETSc Web page. `https://petsc.org/`, 2023.

10 Bell, J. B., Colella, P., and Glaz, H. M. A second-order projection method for the incompressible navier-stokes equations. *Journal of computational physics 85*, 2 (1989), 257–283.

11 Bird, K., and Sherwin, M. J. *American Prometheus: the triumph and tragedy of J. Robert Oppenheimer*. Atlantic Books, 2021.

12 Brooks, A. N., and Hughes, T. J. Streamline upwind/petrov-galerkin formulations for convection dominated flows with particular emphasis on the incompressible

navier-stokes equations. *Computer methods in applied mechanics and engineering 32*, 1-3 (1982), 199–259.

13 Camata, J., Rossa, A., Valli, A., Catabriga, L., Carey, G., and Coutinho, A. Reordering and incomplete preconditioning in serial and parallel adaptive mesh refinement and coarsening flow solutions. *International Journal for Numerical Methods in Fluids 69*, 4 (2012), 802–823.

14 Camata, J. J., HENAO, C., and COUTINHO, A. Reduced integration with hourglass stailization for bi-linear quadrlilateral elements in libmesh. In *V National Congress in Mechanical Engineering* (2008).

15 Chan, W.-M., and George, A. A linear time implementation of the reverse cuthill-mckee algorithm. *BIT Numerical Mathematics 20*, 1 (1980), 8–14.

16 Chorin, A. J. Numerical solution of the navier-stokes equations. *Mathematics of computation 22*, 104 (1968), 745–762.

17 Cuthill, E., and McKee, J. Reducing the bandwidth of sparse symmetric matrices. In *Proceedings of the 1969 24th National Conference* (New York, NY, USA, 1969), ACM '69, Association for Computing Machinery, p. 157–172.

18 Dagum, L., and Menon, R. Openmp: an industry standard api for shared-memory programming. *Computational Science & Engineering, IEEE 5*, 1 (1998), 46–55.

19 Dupont, T., Kendall, R. P., and Rachford, Jr, H. An approximate factorization procedure for solving self-adjoint elliptic difference equations. *SIAM Journal on Numerical Analysis 5*, 3 (1968), 559–573.

20 Garey, M. R., and Johnson, D. S. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W. H. Freeman Co., USA, 1990.

21 Geuzaine, C., and Remacle, J.-F. Gmsh: A 3-d finite element mesh generator with built-in pre-and post-processing facilities. *International journal for numerical methods in engineering 79*, 11 (2009), 1309–1331.

22 Guo, X., Lange, M., Gorman, G., Mitchell, L., and Weiland, M. Developing a scalable hybrid mpi/openmp unstructured finite element model. *Computers Fluids 110* (2015), 227–234. ParCFD 2013.

23 Gutknecht, M. H., and Röllin, S. The chebyshev iteration revisited. *Parallel Computing 28*, 2 (2002), 263–283.

24 Harrington, P., Yoo, W., Sim, A., and Wu, K. Diagnosing parallel i/o bottlenecks in hpc applications. In *International Conference for High Performance Computing Networking Storage and Analysis (SCI7) ACM Student Research Competition (SRC)* (2017), p. 4.

25 Hernandez, V., Roman, J. E., and Vidal, V. SLEPc: A scalable and flexible toolkit for the solution of eigenvalue problems. *ACM Trans. Math. Software 31*, 3 (2005), 351–362.

26 Hughes, T. J. *The finite element method: linear static and dynamic finite element analysis*. Courier Corporation, 2012.

27 Hulbert, G. M., and Hughes, T. J. Space-time finite element methods for second-order hyperbolic equations. *Computer methods in applied mechanics and engineering 84*, 3 (1990), 327–348.

28 Karypis, G., and Kumar, V. Multilevel graph partitioning schemes. In *ICPP (3)* (1995), pp. 113–122.

29 Karypis, G., and Kumar, V. Metis: A software package for partitioning unstructured graphs, partitioning meshes, and computing fill-reducing orderings of sparse matrices.

30 Karypis, G., Schloegel, K., and Kumar, V. Parmetis: Parallel graph partitioning and sparse matrix ordering library.

31 Kirk, B. S., Peterson, J. W., Stogner, R. H., and Carey, G. F. libmesh: a c++ library for parallel adaptive mesh refinement/coarsening simulations. *Engineering with Computers 22* (2006), 237–254.

32 Kitware. Vtk file format. `https://examples.vtk.org/site/VTKFileFormats/`. Acessado em 15/10/2023.

33 Laboratory, L. B. N., of Energy. Office of Scientific, U. S. D., and Information, T. *Conduit - Scientific Data Exchange Library for HPC Simulations*. United States. Department of Energy, 2014.

34 Leveque, R. J. High-resolution conservative algorithms for advection in incompressible flow. *SIAM Journal on Numerical Analysis 33*, 2 (1996), 627–665.

35 Lisi, M. Some remarks on the cantor pairing function. *Le Matematiche 62*, 1 (2007), 55–65.

36 Naumov, M., Chien, L., Vandermersch, P., and Kapasi, U. Cusparse library. In *GPU Technology Conference* (2010).

37 Netzer, R. H. B., and Miller, B. P. What are race conditions? some issues and formalizations. *ACM Lett. Program. Lang. Syst. 1*, 1 (Mar. 1992), 74–88.

38 Nieminen, J., and Yliluoma, J. Function parser for c++. `http://warp.povusers.org/FunctionParser/fparser.html`. Acessado em 22/09/2023.

39 NVIDIA, Vingelmann, P., and Fitzek, F. H. Cuda, release: 10.2.89, 2020.

40 Ramos, J. I. Picard's iterative method for nonlinear advection–reaction–diffusion equations. *Applied Mathematics and Computation 215*, 4 (2009), 1526–1536.

41 Rokos, G., Gorman, G., and Kelly, P. H. A fast and scalable graph coloring algorithm for multi-core and many-core architectures. In *European Conference on Parallel Processing* (2015), Springer, pp. 414–425.

42 Saad, Y., and Schultz, M. H. Gmres: A generalized minimal residual algorithm for solving nonsymmetric linear systems. *SIAM Journal on scientific and statistical computing 7*, 3 (1986), 856–869.

43 Sahni, O., Zhou, M., Shephard, M. S., and Jansen, K. E. Scalable implicit finite element solver for massively parallel processing with demonstration to 160k cores. In *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis* (2009), pp. 1–12.

44 Schroeder, W., Martin, K., and Lorensen, B. *The Visualization Toolkit (4th ed.)*. Kitware, 2006.

45 Shewchuk, J. R., et al. An introduction to the conjugate gradient method without the agonizing pain, 1994.

46 Stuart, A. M., and Peplow, A. The dynamics of the theta method. *SIAM journal on scientific and statistical computing 12*, 6 (1991), 1351–1372.

47 Temam, R. *Navier-Stokes equations: theory and numerical analysis*, vol. 343. American Mathematical Soc., 2001.

48 Trilinos Project Team, T. *The Trilinos Project Website*.

49 Valli, A. M., Catabriga, L., Santos, I. P., Coutinho, A. A., and Almeida, R. C. Predictor-multicorrector scheme for the dynamic diffusion method. *Proceeding Series of the Brazilian Society of Computational and Applied Mathematics 3*, 2 (2015).

50 Walter, J., Koch, M., et al. ublas. *Boost C++ software library available from http://www. boost. org/doc/libs* (2006).

51 Yi, H., Rasquin, M., Fang, J., and Bolotnov, I. A. In-situ visualization and computational steering for large-scale simulation of turbulent flows in complex geometries. In *2014 IEEE International Conference on Big Data (Big Data)* (2014), pp. 567–572.