Lucas Clemente Vella

Contribuição para a Solução Numérica Monolítica do Modelo Matemático Diferencial para a Fluidodinâmica Incompressível Newtoniana

Universidade Federal de Uberlândia

Faculdade de Engenharia Mecânica

2018

Lucas Clemente Vella

Contribuição para a Solução Numérica Monolítica do Modelo Matemático Diferencial para a Fluidodinâmica Incompressível Newtoniana

Tese apresentada ao Programa de Pós-graduação em Engenharia Mecânica da Universidade Federal de Uberlândia, como parte dos requisitos para a obtenção do título de **Doutor em Engenharia Mecânica**.

Área de Concentração: Transferência de Calor e Mecânica dos Fluidos.

Orientador: Prof. Dr. Aristeu da Silveira Neto

Uberlândia – MG

2018

Dados Internacionais de Catalogação na Publicação (CIP)
Sistema de Bibliotecas da UFU, MG, Brasil.

V43c Vella, Lucas Clemente, 1987-

Contribuição para a solução numérica monolítica do modelo matemático diferencial para a fluidodinâmica incompressível newtoniana [recurso eletrônico] / Lucas Clemente Vella. -

Orientador: Aristeu da Silveira Neto.

Tese (Doutorado) - Universidade Federal de Uberlândia, Programa de Pós-Graduação em Engenharia Mecânica.

Modo de acesso: Internet.

Disponível em: http://dx.doi.org/10.14393/ufu.te.2018.790

Inclui bibliografia.

Inclui ilustrações.

1. Engenharia mecânica. 2. Fluidodinâmica computacional. 3. Dinâmica dos fluidos. 4. Fluidos newtonianos. I. Silveira Neto, Aristeu da, 1955- (Orient.) II. Universidade Federal de Uberlândia. Programa de Pós-Graduação em Engenharia Mecânica. III. Título.

CDU: 621

Maria Salete de Freitas Pinheiro - CRB6/1262

SERVIÇO PÚBLICO FEDERAL
MINISTÉRIO DA EDUCAÇÃO
UNIVERSIDADE FEDERAL DE UBERLÂNDIA
FACULDADE DE ENGENHARIA MECÂNICA
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA MECÂNICA

ATA DE DEFESA DE TESE
NÚMERO DE ORDEM: 246
DATA: 29/06/2018

Às oito horas e trinta minutos do dia vinte e nove de junho de dois mil e dezoito, no Auditório do Laboratório de Mecânica dos Fluidos - Bloco 5P, Campus Santa Mônica, reuniu-se a Banca Examinadora composta pelos Professores Dr. Aristeu da Silveira Neto (orientador), Dr. Aldemir Aparecido Cavalini Júnior, Dr. Francisco José de Souza e Dra. Millena Martins Villar, da Universidade Federal de Uberlândia, Dr. Adenilso da Silva Simão, da Universidade de São Paulo/São Carlos, e Dr. Rudnei Dias da Cunha, da Universidade Federal do Rio Grande do Sul, para, sob a presidência do primeiro, desenvolver o processo de avaliação da tese intitulada ***"Contribuição para a Solução Monolítica do Modelo Matemático Diferencial para a Fluidodinâmica Incompressível Newtoniana"***, apresentada pelo aluno **LUCAS CLEMENTE VELLA**, matrícula número **11413EMC009**, em complementação aos requisitos determinados pelo Regimento do Programa de Pós-Graduação em Engenharia Mecânica para obtenção do título de Doutor. Após discorrer sobre seu trabalho, o candidato foi arguido pelos membros da Banca, diante das comunidades universitária e externa. Em seguida, a tese foi avaliada em seção privada pelos membros da Banca que, ao encerrar o processo, consideraram-na:

( ) Aprovada

(X) Aprovada com modificações a serem submetidas para a aprovação do orientador

( ) Aprovada com modificações a serem submetidas para a aprovação da banca

( ) Reprovada

conferindo ao aluno, em caso de aprovação, o título de Doutor em Engenharia Mecânica, Área de Concentração: **Transferência de Calor e Mecânica dos Fluidos**; Linha de Pesquisa: **Dinâmica dos Fluidos e Transferência de Calor**. As demandas complementares observadas pelos examinadores deverão ser satisfeitas no prazo máximo de 30 dias, para dar validade a esta aprovação. Para constar, lavrou-se a presente ata, que vai assinada pelo presidente e demais membros da Banca.

Assinaturas:

Prof. Dr. Aristeu da Silveira Neto (orientador)	UFU	
Prof. Dr. Aldemir Ap. Cavalini Júnior	UFU	
Prof. Dr. Francisco José de Souza	UFU	
Dra. Millena Martins Villar	UFU	
Prof. Dr. Adenilso da Silva Simão	USP/São Carlos	
Prof. Dr. Rudnei Dias da Cunha	UFRGS	

Uberlândia, 29 de junho de 2018

Agradecimentos

Agradeço ao Prof. Aristeu, pela orientação e paciência.

Agradeço aos meus colegas de laboratório, com os quais sempre pude contar, e que me ajudaram imensamente durante o desenvolvimento deste trabalho.

Agradeço também à FAPEMIG e à Petrobras pelo financiamento.

Resumo

O desenvolvimento de um *software* paralelo para simulação de escoamentos de fluidos incompressíveis é apresentado. São abordados vários aspectos do desenvolvimento da ferramenta, como a solução de sistemas lineares, a escrita de dados, o particionamento do domínio e a distribuição da carga entre vários processos MPI. O problema é descrito em termos das variáveis primárias por meio das equações de Navier-Stokes, que são discretizadas pelo método de diferenças finitas. É desenvolvida uma metodologia de solução monolítica das equações discretizadas do problema, aplicável a malhas deslocadas estruturadas. São avaliados vários esquemas advectivos e várias técnicas de tratamento das não linearidades, incluindo o método de Newton-Raphson completo, sendo a matriz jacobiana explicitamente construída. A turbulência é tratada através de simulação das grandes escalas, onde a viscosidade turbulenta calculada pelo modelo de Smagorinsky, atenuada com a função de van Driest. Os sistemas lineares são resolvidos iterativamente através do complemento de Schur, que possibilita o tratamento das matrizes que surgem naturalmente do problema, e que por conterem zeros na diagonal principal, não podem ser resolvidas por métodos iterativos comuns.

Palavras-chave: escoamento incompressível, solução monolítica, método de Newton, particionamento de domínio.

Abstract

The development of a incompressible fluid flow parallel simulation software is presented. Many aspects of the tool's development are discussed, including the solving of linear systems, data output, domain partitioning and load balancing among multiple MPI processes. The problem is formulated in terms of primary variables through the Navier-Stokes equations, which are discretized using the finite differences method. A framework for solving the discrete equations for the problem in a fully-coupled way applicable to structured staggered grids is developed. Multiple advection schemes and multiple non-linearity treatment techniques are evaluated, including the full Newton-Raphson method, where the Jacobian is explicitly built. The turbulence is treated through large eddy simulation, where the eddy viscosity is calculated via the Smagorinsky model, damped with the van Driest function. Linear systems are solved iteratively through the Schur complement, enabling the handling of the kind of matrices arising naturally from the problem, which contains zeros on the main diagonal, and can not be solved through ordinary iterative methods.

Keywords: incompressible flow, fully-coupled solution, Newton method, domain partitioning

Lista de figuras

Lista de tabelas

Lista de abreviaturas e siglas

2UP	*upwind* de segunda ordem
AMG	*Multigrid* algébrico
API	interface de programação de aplicação
ASM	*Additive Schwarz Method*
BiCGSTAB	gradiente biconjugado estabilizado
CDS	esquema de diferenças centradas
CFD	fluidodinâmica computacional
CFL	Courant-Friedrichs-Lewy
CLAM	*Curved-Line Avection Method*
CUBISTA	*Convergent and Universally Bounded Interpolation Scheme for the Treatment of Advection*
DNS	simulação numérica direta
EDP	equação diferencial parcial
GMRES	método generalizado de minimização de resíduo
GUI	interface gráfica de usuário
HDF5	*Hierarchical Data Format v. 5*
HSFC	*Hilbert Space-Filling Curve*
LES	simulação de grandes escalas
LSC	*least squares commutator*
MAFISC	*Multidimensional Adaptive Filtering Improved Scientific data Compression*
MFLab	Laboratório de Mecânica dos Fluidos
MPI	*Message-Passing Interface*
PHG-G	*Parallel Hypergraph Partitioning* em Grafo
PHG-H	*Parallel Hypergraph Partitioning* em Hipergrafo
RCB	*Recursive Coordinate Bisection*
RCM	Cuthill-McKee reverso
RIB	*Recursive Inertial Bisection*
SCGS	*symmetrical coupled Gauss-Seidel*
SFC	*Space-Filling Curve*
SIMPLE	*Semi-Implicit Method for Pressure Linked Equations*
SIVA	*Simultaneous Variable Adjustments*
SOR	sobrerrelaxação sucessiva
VTK	*The Visualization Toolkit*
XDMF	*eXtensible Data Model and Format*
XML	*Extensible Markup Language*

Lista de símbolos

Ac	Admensional característico para controle do passo de tempo
At	Fator de atenuação de van Driest
C_S	Constante de Smagorinsky
Δ	Comprimento do filtro LES
L	Dimensão característica do escoamento
N	Número total de incógnitas no domínio $n + m$
P	Número de processos MPI na execução
Re	Número de Reynolds
S_{ij}	Elemento do tensor taxa de deformação
S	Tensor taxa de deformação
U	Velocidade característica do escoamento
D	Número de dimensões espaciais
$\mathbb{D}_i^C(\zeta)$	Operador diferença finita espacial de segunda ordem
δ_{ij}	Delta de Kronecker
λ	Segundo coeficiente da viscosidade
μ_t	Viscosidade dinâmica turbulenta
μ	Viscosidade dinâmica
m	Número total de facetas do domínio
ν_t	Viscosidade cinemática turbulenta
ν	Viscosidade cinemática
n	Número total de células domínio
φ_i	i-ésima componente da força de corpo
p	Pressão fluidostática
$\mathbb{R}$	Conjunto dos números reais
ρ	Massa específica do fluido
σ_{ij}	Elemento do tensor de tensões de Cauchy
$\boldsymbol{s}$	Vetor de tamanho N com todas as incógnitas do domínio
t_P	Tempo de processamento gasto na execução do cálculo de um passo de tempo
τ_{ij}	Elemento do tensor deviatórico
t	Tempo
u_i	i-ésima componente da vetor velocidade
$\boldsymbol{u}$	Vetor velocidade
x_i	i-ésima componente do vetor posição

Sumário

1 Introdução

A disponibilidade de computadores cada vez melhores e mais baratos tornou, nas últimas décadas, a fluidodinâmica computacional (CFD) uma ferramenta viável para a indústria. Em cada vez mais situações, é possível utilizar CFD para se obter informações mais precisas do que seria possível com correlações empíricas, e de forma mais barata do que realizar ensaios materiais.

O desenvolvimento de um *software* de CFD é um esforço inerentemente multidisciplinar, que envolve desafios na área de computação no mínimo tão complexos quanto aos aspectos numéricos e físicos associados ao problema em análise. As aplicações em CFD figuram dentre os problemas computacionalmente mais exigentes que existem, e não é coincidência que meteorologia seja um dos principais usos para os maiores supercomputadores do mundo (FELDMAN, 2017).

No presente trabalho apresento minhas contribuições à HigTree, uma biblioteca de *software* para solução numérica de equações diferenciais parciais (EDPs), fruto de um projeto de colaboração entre a UFU, USP e Petrobras.

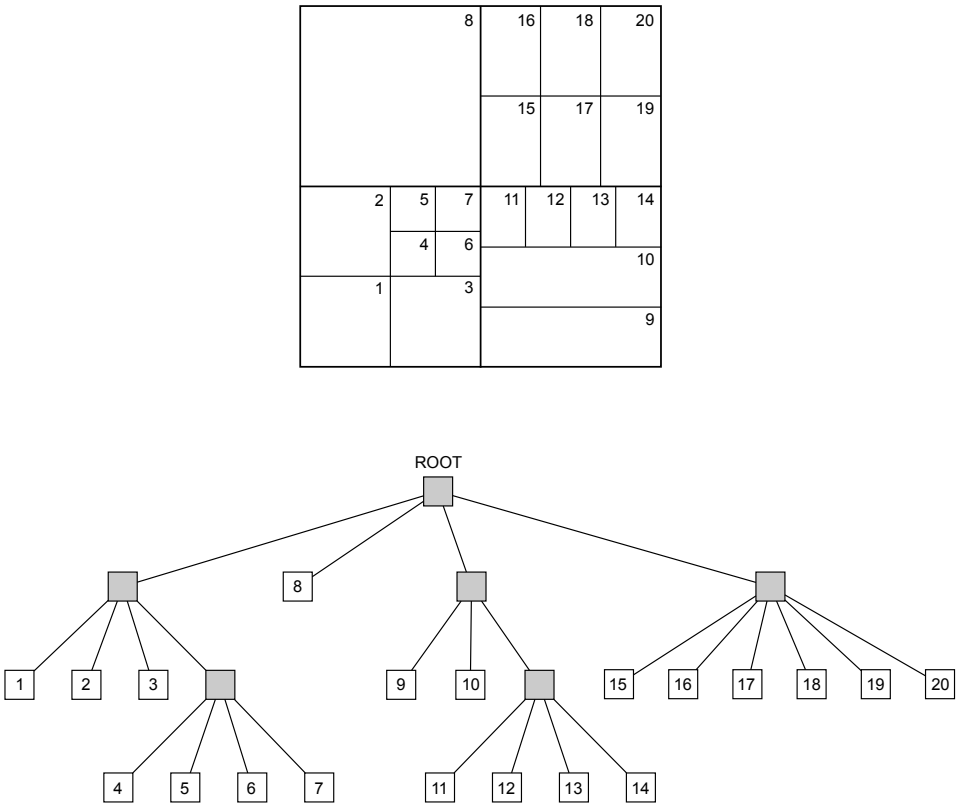
Utilizando as funcionalidades da HigTree, desenvolvi uma ferramenta de CFD para solução numérica da formulação diferencial para escoamentos incompressíveis newtonianos, com acoplamento entre pressão-velocidade monolítico e várias alternativas de tratamento das não linearidades.

1.1 HigTree

A HigTree é uma biblioteca para solução numérica de EDPs com domínio no tempo (em $\mathbb{R}$) e no espaço (em $\mathbb{R}^D$, tal que D é um número arbitrário de dimensões espaciais), desenvolvida com o objetivo de se resolver escoamentos de fluidos, porém com o cuidado para ser genérica o suficiente para outras aplicações e EDPs. Foi inicialmente desenvolvida pelos professores Adenilso da Silva Simão, Antonio Castelo Filho, Fabrício Simeoni de Sousa e Leandro Franco de Souza, pesquisadores da USP – ICMC em São Carlos – SP, até o meu ingresso no projeto, quando passei a colaborar com o desenvolvimento pelo Laboratório de Mecânica dos Fluidos (MFLab) na UFU.

Ela oferece suporte à discretização do domínio em malha estruturada hierárquica, utilizando para o domínio uma estrutura de dados em árvore (o que define o nome HigTree: do inglês, *Hierarchical Grid Tree*), como ilustrado na Figura 1. O domínio da EDP discretizada é formado por várias árvores conectadas, e as folhas da árvore são os elementos

Figura 1 – Exemplo de uma árvore da malha hierárquica da HigTree.



Fonte – Simão et al. (2014)

do domínio: hiper-retângulos[1] de dimensão D. As variáveis podem ser representadas ou nas facetas dos elementos retangulares, ou nos centros dos elementos.

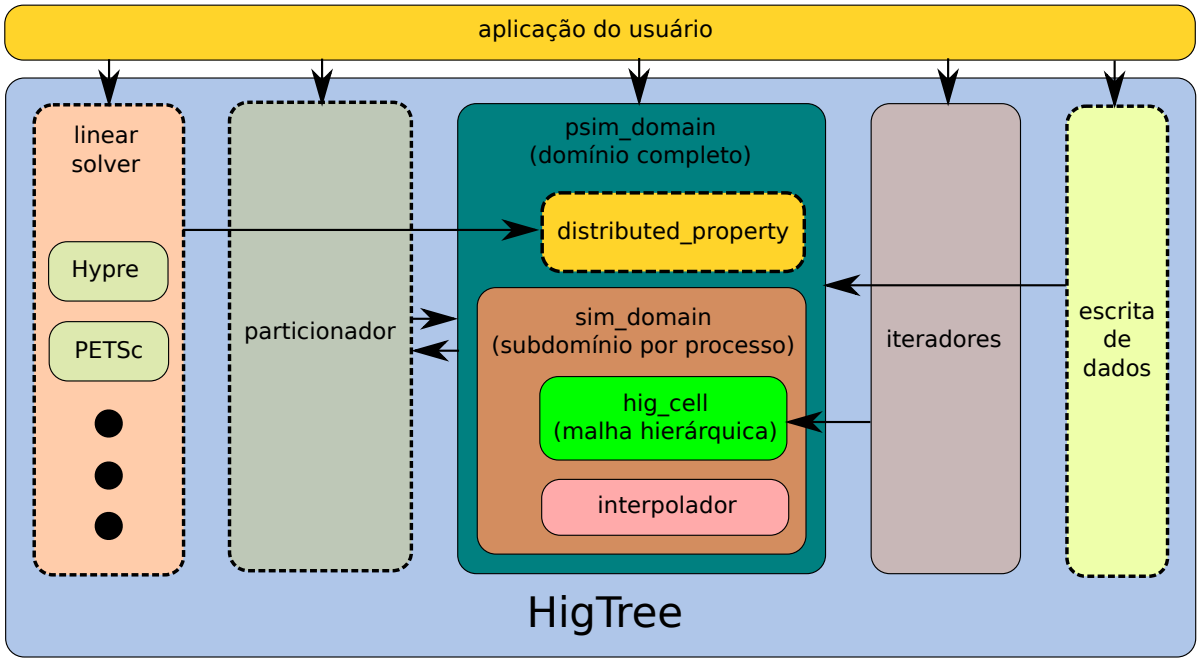
Em si, a HigTree é uma biblioteca de *software*, e não uma aplicação. Apesar de conter várias ferramentas necessárias e úteis à simulação numérica de escoamentos, nela não tem implementado nenhum modelo computacional relativo a problemas físicos. Em vez disso, acompanha-a em sua distribuição um conjunto de exemplos, que são aplicações independentes entre si, cada um demonstando uma funcionalidade ou aplicação específica da HigTree. Dentre os exemplos, estão uma aplicação para resolver a equação de Poisson, e outra para resolver um escoamento em um domínio complexo, onde foi utilizado o método da projeção para o acoplamento pressão velocidade.

A HigTree foi escrita originalmente na linguagem de programação C, utilizando uma arquitetura fortemente orientada a objetos [2]. Eventualmente, desenvolvi alguns módulos na linguagem C++, porém mantendo as interfaces de programação na linguagem original, de modo a não sacrificar a boa interoperabilidade que C possui com outras linguagens de programação. Possui também ferramentas auxiliares em Octave e Python.

[1] Hiper-retângulo é a generalização do retângulo para D dimensões: uma forma geométrica cujos ângulos são todos retos.

[2] Mesmo que C não seja uma linguagem orientada a objetos, as técnicas e princípios do paradigma podem ser utilizados.

Figura 2 – Principais componentes da HigTree e suas interações.



Construída desde a concepção para executar em computadores e *clusters* multi-processados, a HigTree utiliza o modelo de paralelismo e mecanismo de comunicação do padrão *Message-Passing Interface* (MPI) (MPI FORUM, 2015), onde múltiplos processos trabalham cooperativamente para realizar uma tarefa. Assim, todos os componentes da biblioteca estão preparados para funcionar em paralelo.

Os principais componentes da versão atual da HigTree são apresentados na Figura 2. As setas representam a dependência de um componente ao outro. Os componentes circulados por linhas tracejadas são aqueles que eu desenvolvi ou modifiquei significativamente no contexto do presente trabalho. Os componentes são descritos a seguir.

1.1.1 Domínio completo

O domínio completo da simulação é modelado pelas classes `psim_domain` e `psim_facet_domain` (esta última omitida na Figura 2 por ser muito similar à primeira), dando aos processos individuais a visão de onde seu subdomínio se insere no todo. Esta é a funcionalidade mais fundamental da HigTree: abstrair as complexidades inerentes ao paralelismo, cuidando dos detalhes de sincronismo entre processos e divisão de tarefas.

Como outros componentes da HigTree, estas classes são divididas entre discretização de variáveis no centro (`psim_domain`) e na facetas (`psim_facet_domain`) dos elementos retangulares. Elas cuidam da iteração entre porção do subdomínimo local ao processo — descrito na subseção 1.1.2 — com os subdomínios topologicamente conectados contidos em outros processos, doravante denominados processos vizinhos. Para isso, elas conhecem a vizinhança local e a correspondência de cada variável discreta local com sua identificação

global.

Sua principal função é criar as propriedades distribuídas — detalhadas na subseção 1.1.3 — e sincronizar as variáveis ali guardadas com os processos topologicamente conectados (doravante chamados processos vizinhos) nos momentos necessários à aplicação.

1.1.2 Subdomínio local

Também separado por duas situações de variáveis, as localizadas nos centros dos elementos (`sim_domain`) e nas facetas (`sim_facet_domain`), o subdomínio local está diretamente associado ao domínio completo. Ele dá a posição geométrica de cada variável discreta armazenada localmente, e sua conectividade na malha hierárquica. Esse subdomínio é composto por um conjunto de árvores, que podem ser separadas em três tipos: domínio interno, *fringe* e condição de contorno.

O domínio interno consiste nos elementos efetivamente controlados por aquele processo. No decorrer de uma execução, ele é o responsável por calcular e escrever o valor das variáveis do domínio interno. Para isso, ele pode usar os valores conhecidos das outras variáveis do domínio interno, do *fringe* e das condições de contorno.

O *fringe* é a parte do subdomínio local que se sobrepõe ao domínio interno de um processo vizinho, ou até mesmo do mesmo processo, em casos que o domínio é periódico, dando acesso às variáveis geometricamente próximas ao limite entre processos ou dos limites da periodicidade. O processo local utiliza as variáveis do *fringe* somente para leitura, conectando sua solução com a do resto do domínio. O *fringe* é automaticamente atualizado durante a sincronização de uma propriedade distribuída. As variáveis calculadas localmente são copiadas para o *fringe* correspondente do processo vizinho e vice-versa.

Já as árvores da condição de contorno podem determinar uma condição de contorno de Dirichlet, onde o valor da propriedade representada é imposta, ou uma condição de Neumann, onde o valor da primeira derivada da propriedade em relação à direção normal da superfície é imposta. Em ambos os casos, os valores são sempre impostos no centro do elemento das árvores de condição de contorno (mesmo na classe `sim_facet_domain`, onde a variável é normalmente imposta na faceta). Estas árvores são degeneradas em uma dimensão (possuem espessura zero), pois delimitam o domínio.

O subdomínio local oferece a funcionalidade do usuário percorrer todos os seus elementos do domínio interno através de iteradores — descritos na subseção 1.1.6, o que é uma operação fundamental dentro da aplicação, pois é nessa interação que o modelo numérico é aplicado a cada variável discreta calculada.

Também utilizado como uma operação fundamental, a partir de um objeto de subdomínio local é possível interpolar um valor da variável representada em qualquer posição geométrica contida ou nas proximidades do domínio interno. Utilizando o interpo-

lador, qualquer informação advinda do *fringe* ou da condição de contorno é computada transparentemente para a aplicação, e portanto ele é o mecanismo padrão de se obter o valor de uma variável na HigTree.

1.1.3 Propriedade distribuída

A propriedade distribuída, implementada na classe `distributed_property`, é um vetor compartilhado entre processos. É o mecanismo oferecido pela HigTree para o armazenamento dos valores discretos das variáveis da EDP, de modo que eles se mantenham sincronizados entre os vários processos da execução. Em termos matemáticos, uma `distributed_property` é um campo escalar discretizado por todo o domínio da EDP. Campos vetoriais são representados por múltiplas instâncias da classe, uma para cada dimensão.

Cada valor da `distributed_property` é associada a um elemento da malha (centro ou faceta) pelo seu respectivo `sim_domain`, e associada aos processos vizinhos, no caso de fazerem parte do *fringe* de um outro processo, pelo `psim_domain` correspondente. Ao contrário das classes onde a geometria é importante, não existe uma variante para os centros e outra para as facetas, a mesma classe é usada nas duas ocasiões.

O acesso às variáveis pode ser feito diretamente pela aplicação, pela correspondência com os elementos da malha através do `sim_domain` ou transparentemente através do interpolador. Quando o campo escalar é modificado, a aplicação deve solicitar explicitamente a sincronização com os outros processos.

Essa é uma das classes à qual contribuí significativamente durante a execução do presente trabalho, ao implementar a distribuição das propriedades entre os processos MPI. Originalmente, todos os processos compartilhavam uma cópia global do campo escalar, consumindo mais memória que o necessário, e mais banda de rede durante a etapa de sincronização. Este desenvolvimento é descrito em detalhes no capítulo 7.

1.1.4 Interpolador

O interpolador da HigTree é separado em duas partes:

- o procedimento de busca de pontos;
- o cálculo dos pesos de cada ponto na interpolação.

A busca de pontos está intimamente ligada ao `sim_domain`. Quando iniciada através de um subdomínio local, todos os pontos que possivelmente contribuem para o valor da variável dentro do domínio são selecionados em uma busca por proximidade, envolvendo

todas as árvores que compõem aquele subdomínio (sejam elas parte do domínio interno, do *fringe* ou da condição de contorno).

A posição dos pontos é passada para o procedimento de cálculo de pesos, que determina a influência dos valores conhecidos no valor interpolado. Este procedimento em si é independente do `sim_domain`, e pode ser utilizado diretamente pela aplicação para seus propósitos.

O procedimento de cálculo de pesos implementado na HigTree é o método dos mínimos quadrados móveis, que supõe que a variável interpolada obedece a um polinômio de grau requisitado. O polinômio é ajustado de modo a minimizar a diferença entre os valores dos pontos conhecidos e o valor dado pelo polinômio, dando mais importância ao erro nas proximidades do ponto requisitado.

Este procedimento requer um certo número mínimo de pontos para funcionar, que é o número de termos do polinômio de grau e dimensão requisitados (caso em que o ajuste se reduz a uma interpolação exata), mas não tem um número máximo: todos os pontos dados têm influência sobre o valor interpolado.

Tendo os pesos dos pontos encontrados, o `sim_domain` computa a influência das condições de contorno, e retorna para a aplicação o estêncil e os pesos de cada ponto que influencia no valor final do ponto requisitado. Com essa informação o valor pode ser calculado, ou o estêncil pode ser utilizado diretamente na montagem de um sistema de equações.

A HigTree possui um sistema de cache que permite que o trabalho de busca de pontos e cálculo de pesos seja feito somente uma vez por ponto. Se um ponto conhecido for requisitado novamente, o resultado do cálculo já armazenado em uma tabela *hash* é simplesmente retornado.

1.1.5 Estrutura da árvore

A classe `hig_cell` é possivelmente a estrutura mais fundamental da HigTree, que é refletido no fato de ela ser a estrutura de dados que deu o nome à biblioteca. Como é comum nas estruturas de dados de árvore, a classe representa tanto um nó da árvore, que pode ser um nó interno ou uma folha, quanto uma subárvore completa, originada a partir de uma `hig_cell` raiz.

Ela possui uma dimensão geométrica, determinada pelo seus pontos mínimo e máximos no espaço $\mathbb{R}^D$, que define os limites do hiper-retângulo. Também guarda uma referência para um possível nó pai, caso não seja a raiz da árvore, e finalmente, guarda referências para seus nós filhos, cujos hipervolumes compõem completamente o hipervolume do nó pai, sem interseções ou vacâncias.

Os nós filhos de uma `hig_cell` são dispostos em uma grade ortogonal de lados $[q_1, \ldots, q_D]$, tal que o número total de filhos de um nó é dado por $\prod_{i=1}^{D} q_i$. Os filhos são ordenados nessa grade por suas posições geométricas, de modo que, dado um ponto em $\mathbb{R}^D$ contido no nó pai, é possível localizar com D buscas binárias qual nó filho contém aquele ponto.

Embora seja possível formar árvores com um número flexível de filhos na HigTree, seu uso recomendado — e como foi usado no presente trabalho — é que somente o nó raiz tenha um número arbitrário de filhos, formando o nível mais grosseiro do domínio em uma malha estruturada. Dessa malha base, cada nó é então refinado sempre em uma proporção de 2^D para 1 (de modo que $q_i = 2$, para todo i). No caso 3D, a partir da malha base, a malha é então caracterizada como uma *octree*, pois cada nó interno é subdivido em 8.

Essa configuração nos dá a garantia do custo computacional de uma busca pelo nó folha contendo um dado ponto. No pior caso, são necessárias $D \log_2 n$ operações dentro do nó raiz para se determinar a qual elemento da malha base o ponto pertence (o custo de D buscas binárias). Sabendo o elemento da malha base, no máximo são necessárias D operações para determinar a qual filho o ponto pertence (uma comparação por dimensão). Este custo se repete por no máximo $\log_{2^D} n$ vezes, que é a profundidade máxima da árvore. Considerando que D é constante para uma dada execução, a complexidade final da busca, em tempo, é no máximo $O(D \log_2 n + D \log_{2^D} n) = O(\log n)$.

Cada nó possui um identificador opaco[3] para o seu centro, e um para cada uma de suas faces. Esses identificadores são utilizados pelo `sim_domain` para associar as variáveis do domínio discretizado à malha representada pelas `hig_cell`.

1.1.6 Iteradores

Iterador é um conceito muito utilizado em linguagens orientadas a objetos. Basicamente, um iterador é um objeto que oferece pelo menos duas operações:

- retornar o elemento atual;
- avançar.

Iteradores são frequentemente utilizados para percorrer uma sequência de elementos em uma estrutura de dados complexa, mas podem ser tão simples quanto um ponteiro para um *array*, já que este pode ser incrementado, e pode ser usado para se obter o elemento apontado. De fato, em C++ um ponteiro simples satizfaz as condições de um iterador, e portanto pode ser usado em qualquer função ou método da biblioteca padrão onde um iterador é esperado.

[3] Um identificador opaco é um valor que identifica um objeto de forma única, porém não contém nenhuma informação sobre o objeto identificado, e só é relevante naquele contexto específico.

A HigTree implementa dois tipos de iteradores: um que retorna uma sequência de células e outro que retorna uma sequência de facetas. Vários critérios são suportados para a seleção de quais elementos serão retornados, dentre eles:

- todas as folhas de uma árvore;
- todos os elementos de uma árvore (incluindo os nós internos);
- os elementos de uma árvore contidos em um hiper-retângulo;
- a concatenação de múltiplos outros iteradores;
- todos os elementos do domínio interno de um `sim_domain`.

Na HigTree, os iteradores são o principal mecanismo utilizado para percorrer os elementos de uma árvore. A aplicação obtém um iterador ou do domínio local, ou com uma chamada explícita de alguma das funções de seleção de elementos. É então usado dentro de um laço (`for` ou `while`), onde a cada iteração o programador realiza três operações sobre o iterador: obtém o elemento atual, avança o iterador, e verifica se a iteração acabou (informação esta também disponível pelo iterador da HigTree). Ao final do laço, o programador é responsável por destruir explicitamente o iterador, que desaloca a memória utilizada pelo objeto.

1.1.7 Particionador

O particionador é a minha maior contribuição à biblioteca da HigTree, sendo responsável por pegar a descrição das várias árvores que compõem o domínio da simulação e particioná-lo entre os vários processos MPI, de modo que o custo computacional seja distribuído de maneira equilibrada entre os processos. Os subdomínios locais e a conectividade entre os processos é então estabelecida de acordo com este particionamento. Seu desenvolvimento é descrito em detalhes no capítulo 6.

Desenvolvi este novo particionamento em substituição ao mecanismo anterior disponível na HigTree, que suportava particionar somente domínios descritos em uma única árvore (e portanto, retangulares). Domínios compostos por múltiplas árvores, em formatos arbitrários, não podiam ser executados em paralelo, sendo a solução anterior, portanto, provisória para os propósitos da HigTree.

1.1.8 Escrita de dados

Originalmente, a HigTree oferecia suporte parcial à escrita de dados no formato de visualização do *The Visualization Toolkit* (VTK) (SCHROEDER; MARTIN; LORENSEN, 2006), contendo funções para a escrita da malha hierárquica em VTK. Porém, a escrita

dos dados em si não faz parte da biblioteca, podendo ser encontrada somente nos arquivos de exemplo que acompanham a distribuição da HigTree. O suporte à escrita de dados também era limitado pelo fato de somente a variante mais ineficiente do VTK ter sido implementada, a saber: em texto, sem compressão, com arquivos independentes entre si.

No contexto do presente trabalho, implementei dentro da HigTree um formato de saída mais eficiente chamado *eXtensible Data Model and Format* (XDMF) (CLARKE; MARK, 2007), baseado em outros dois formatos de arquivo mais básicos, *Extensible Markup Language* (XML) (BRAY et al., 2008) e *Hierarchical Data Format v. 5* (HDF5) (THE HDF GROUP, 1997-2018), que é mais eficiente por utilizar codificação binária na maior parte dos dados, utilizar compressão e evitar repetições ao fazer referência a arquivos escritos anteriormente. Este desenvolvimento é detalhado no capítulo 4.

Esta continua sendo a única parte da HigTree que não suporta um número arbitrário D de dimensões espaciais. Tanto na saída em VTK quanto em XDMF, D é restrito a 2 ou 3, que são os valores que podem ser tratados pelas ferramentas de visualização.

1.1.9 *Solver* linear

Outro componente da HigTree onde minha contribuição foi significativa é a interface de *solver* de sistemas lineares. Frequentemente, o método numérico para solução de escoamentos envolve a solução de um ou mais sistemas de equações lineares. Aplicações construídas sobre a HigTree podem contar com esta componente para resolver os sistemas lineares que surgem do problema resolvido.

Originalmente, utilizar a biblioteca de *software* PETSc (BALAY et al., 2015) era a única opção de *solver* oferecida pela HigTree. A contribuição consiste na criação de um mecanismo de escolha entre diferentes *solvers*, sem a necessidade de alterar ou recompilar o código-fonte. O sistema linear é descrito como uma multiplicação matriz-vetor através de uma interface de programação de aplicação (API) comum, que encapsula a escolha e execução do *solver*. Este trabalho é detalhado no capítulo 3.

1.2 Cyberex

Para resolver escoamentos incompressíveis, utilizei a HigTree para construir uma aplicação de CFD, batizada de Cyberex. Desenvolvida na linguagem de programação C++, ela foi construída com o objetivo de ser fácil de usar, tanto diretamente, por usuários finais, quanto por outras aplicações, como uma interface gráfica de usuário (GUI) que o utilize como *backend*.

Nesta filosofia, nenhum caso de escoamento configurado no Cyberex deve necessitar que este seja recompilado. Cada caso é configurado como um arquivo na linguagem

JSON (BRAY, 2014) — que é uma linguagem de fácil leitura e interpretação, tanto por pessoas quanto por programas de computador. O caminho deste arquivo é o único parâmetro de execução passado ao Cyberex. Nele estão definidas todas as configurações necessárias à execução do caso, exceto:

- geometria da malha;
- valores definidos por funções do espaço.

A geometria da malha e valores definidos em função da posição — que às vezes são necessários para definir condições de contorno e/ou força de corpo — são dados em arquivos separados. São referenciados dentro do arquivo principal de configuração, pelo caminho relativo ao diretório onde este está localizado.

Cada árvore da malha é definida por um aquivo tipo AMR, que é um formato de arquivo em texto usado inicialmente por outra ferramenta de CFD desenvolvida pelo nosso grupo de pesquisa, o MFSim, e posteriormente herdado pela HigTree. Com este formato, é possível definir malhas bloco estruturada complexas em poucos *bytes*, descrevendo somente os limites de cada bloco retangular de refinamento. Embora não seja ideal para representar malhas hierárquicas arbitrárias, ele tem se mostrado adequado no estágio de desenvolvimento atual da HigTree e do Cyberex.

Funções podem ser definidas de 2 modos:

- por uma aplicação externa, que lê da entrada padrão um ponto em $\mathbb{R}^D$ e escreve na saída padrão o valor da função no ponto;
- por uma função na linguagem de programação Python, que é executada pelo interpretador embutido no Cyberex, passando como parâmetro o ponto no espaço.

Além de portar o método da projeção, já implementado nos arquivos de exemplo da HigTree, para dentro do Cyberex, também implementei o método monolítico para solução de Navier-Stokes incompressível, onde as variáveis da pressão e dá velocidade são resolvidas simultaneamente em um único sistema de equações não lineares.

Implementei múltiplas técnicas para o tratamentos das não linearidades e múltiplos esquemas advectivos, que podem ser selecionados através do arquivo de configuração. Os detalhes da formulação física e dos desenvolvimentos numéricos são dado na parte II do presente trabalho.

1.3 Objetivos

Os métodos segregados para a solução de escoamentos são formulados a partir das variáveis primárias — pressão e velocidade — e podem apresentar problemas de estabilidade em algumas situações, como por exemplo no caso de escoamentos multifásicos com altas razões de densidade entre as fases. Uma solução proposta pela literatura para este problema é a utilização de métodos monolíticos (WACHEM; GOPALA, 2006). No presente trabalho, o objetivo primário é desenvolver uma abordagem para a solução monolítica paralela de escoamentos incompressíveis em malhas cartesianas deslocadas com refinamento estruturado.

Este objetivo é motivado pela aplicabilidade imediata de tal abordagem em ferramentas de CFD preexistentes que utilizam discretização em malha deslocada, mas que sofrem com os problemas de instabilidade do método segregado — como é o caso do próprio MFSim.

A HigTree é utilizada na implementação da abordagem monolítica elaborada, e temos como objetivos secundários o desenvolvimento da própria HigTree como ferramenta de engenharia. As contribuições neste sentido são:

- Flexibilização do sistema de solução de sistemas lineares já existente, permitindo a fácil adição de novas alternativas de *solvers* e sua escolha em tempo de execução.
- Melhorar a *performance* da escrita de dados para visualização e a eficiência em espaço dos dados escritos.
- Permitir o particionamento e execução paralela de problemas com domínios formados por geometrias complexas, já que originalmente, somente problemas com domínios retangulares poderiam ser executados em paralelo na HigTree.
- Distribuir os campos de propriedades físicas entre os processos MPI, de modo que cada processo armazenasse somente os valores pertinentes ao seu subdomínio e vizinhança local.

1.4 Organização do texto

O presente texto é organizado da seguinte forma: o tema é contextualizado pelos dois capítulos iniciais. O capítulo 1 é a introdução, que dá a motivação, os objetivos e descreve as ferramentas desenvolvidas e utilizadas na realização deste trabalho. O capítulo 2 trás uma revisão bibliográfica sobre os principais tópicos de interesse da tese.

Em seguida, o desenvolvimento do texto é divido em duas partes: a primeira, considerando a multidisciplinaridade do problema, trata dos aspectos computacionais

relevantes ao desenvolvimento do trabalho. A segunda parte trata das modelagens física, matemática e numérica da metodologia de CFD desenvolvida.

Na primeira parte, o capítulo 3 descreve o desenvolvimento que realizei sobre a HigTree para dar suporte à escolha — em tempo de execução — do *solver* linear utilizado pela aplicação. No capítulo 4 descrevo a implementação de um formato mais eficiente de escrita de dados para a HigTree. O capítulo 5 apresenta um estudo sobre várias técnicas e bibliotecas de partição de domínios e seu impacto na solução de um sistema de equações lineares. O capítulo 6 descreve a implementação na HigTree de uma das técnicas de partição de domínio estudadas no capítulo anterior. Finalmente, no capítulo 7 descrevo o mecanismo implementado na HigTree para distribuição dos campos de variáveis discretas entre os vários processos da execução.

Na segunda parte, o capítulo 8 descreve as modelagens física e diferencial utilizas no desenvolvimento do Cyberex. O capítulo 9 dá o modelo numérico utilizado, discretização do modelo diferencial descrito no capítulo anterior. O capítulo 10 descreve o mecanismo de solução do sistemas de equações não lineares decorrentes do modelo numérico discreto. Finalmente, o capítulo 11 dá os casos executados e resultados obtidos com a implementação destes modelos e técnicas.

O texto é encerrado com as conclusões e perspectivas, dadas no capítulo 12, que é seguido pelas referências bibliográficas.

2 Revisão bibliográfica

A simulação numérico-computacional de escoamentos incompressíveis é um problema difícil, cujas soluções vêm sendo elaboradas ao longo de décadas. Uma multitude de abordagens já foram propostas para cada um dos diferentes aspectos do problema, o que sugere a possibilidade de, utilizando somente o que já foi proposto, se formular dezenas de combinações possíveis de soluções, nos mais variados graus de viabilidade prática ou teórica. Nesse cenário, nenhuma abordagem se sobressai definitivamente sobre todas as outras.

No presente trabalho, temos o interesse de estudar a classe de soluções onde as variáveis primárias — pressão e velocidade — são resolvidas diretamente a partir da formulação diferencial do problema. Mais especificamente, o nosso interesse é a classe de soluções monolíticas, que prometem maior estabilidade (TUREK, 1996; CHEN; PRZEKWAS, 2010; WACHEM; GOPALA, 2006; XIAO; DENNER; WACHEM, 2017) e escalabilidade (DENG et al., 1994; PERNICE; TOCCI, 2001; DARWISH; MOUKALLED, 2014) se comparados com os métodos segregados, que são atualmente utilizados pelo nosso grupo de pesquisa no contexto dos projetos em desenvolvimento no MFLab (LIMA, 2012; BARBI, 2016; MELO, 2017; DAMASCENO, 2018). Seguindo esta linha, na seção 2.1 é dada uma visão sobre os desenvolvimentos e trabalhos que resolvem monoliticamente as variáveis da pressão e velocidade.

Uma das dificuldades do problema está diretamente relacionada à grande quantidade de informações contidas na solução de um escoamento, que requer tanto uma quantidade de pontos discretos proporcionalmente grande, quanto uma resolução temporal suficientemente grande para capturar corretamente a influência caótica das não linearidades. Frequentemente, é inviável simular problemas corriqueiros de engenharia em um único computador. Assim que a computação tornou viável a solução de um problema utilizando vários computadores ligados em rede, a solução em paralelo de escoamentos passou a ser investigada. Na seção 2.2 é apresentada uma revisão bibliográfica de trabalhos neste sentido.

2.1 Solução monolítica

O algoritmo *Simultaneous Variable Adjustments* (SIVA), de Caretto, Curr e Spalding (1972) foi a primeira tentativa de solução monolítica das equações do balanço da quantidade de movimento e do balanço de massa (DENG et al., 1994). Entretanto, o método segregado *Semi-Implicit Method for Pressure Linked Equations* (SIMPLE) (PATANKAR; SPALDING, 1972) — publicado logo em seguida pelo mesmo grupo de pesquisa em CFD do *Imperial*

College — rapidamente suplantou o interesse pelo SIVA, graças à sua simplicidade e baixo custo em memória (DARWISH; MOUKALLED, 2014).

Em Vanka (1986) um escoamento em uma malha deslocada 2D uniforme é resolvido com acoplamento pressão-velocidade monolítico. Para solução do sistema não linear é utilizado um *solver multigrid* com um relaxador especializado: uma generalização de Gauss-Seidel chamada *symmetrical coupled Gauss-Seidel* (SCGS).

Em vez de operar atualizando o resíduo uma linha por vez, como o método de Gauss-Seidel original, o SCGS opera em blocos de cinco em cinco linhas da matriz do sistema de equações linearizado. Os resíduos das cinco variáveis correspondentes a cada elemento de malha (quatro velocidades nas faces mais a pressão no centro) são atualizados simultaneamente. A operação de divisão pelo elemento da diagonal principal, presente no Gauss-Seidel, é substituída por uma inversão de matriz 5×5, feita analiticamente. A submatriz invertida é formada pelas colunas correspondentes às variáveis sendo atualizadas e pelas linhas correspondentes às quatro equações da quantidade de movimento das faces do elemento, mais a sua equação da continuidade correspondente. A correção do resíduo é subrelaxada por uma constante antes de ser somada à solução.

Embora não diretamente referenciado, o SCGS é muito similar ao SIVA, pois ambos resolvem simultaneamente todas as 5 variáveis presentes em cada célula da malha deslocada 2D uniforme. A principal contribuição de Vanka (1986) sobre o SIVA de Caretto, Curr e Spalding (1972) é a utilização do *multigrid* para acelerar a convergência da solução do sistema de equações.

Em vez de utilizar uma malha deslocada, o trabalho de Deng et al. (1994) utiliza uma malha colocalizada para discretizar e resolver monoliticamente a pressão e a velocidade de um escoamento incompressível tridimensional. Para isto, utiliza uma suavização da pressão baseada no trabalho de Schneider e Raw (1987), que propõe um método alternativo — porém similar — à interpolação de Rhie-Chow (RHIE; CHOW, 1983) para estabilizar a solução da pressão em malhas colocalizadas.

Além de permitir a utilização de malha colocalizada, cuja criação e tratamento é mais simples do que de uma malha deslocada, utilizar um mecanismo de suavização de pressão oferece ainda uma outra vantagem para a solução monolítica: ela provê um termo dependente da pressão na equação da continuidade. Isso torna o sistema linear mais fácil de ser resolvido por métodos iterativos, pois elimina os zeros da diagonal principal da matriz do sistema linear monolítico — que ocorreriam na equação da continuidade no caso de uma malha deslocada. Mais do que eliminar os zeros, o método proposto por Deng et al. (1994), assim como o método de Rhie-Chow, garante dominância diagonal na matriz, tornando possível, por exemplo, utilizar o método de Gauss-Seidel como relaxador, ao invés do mais complexo SCGS.

Turek (1996) compara várias possibilidades de métodos de solução de Navier-Stokes transiente incompressível em elementos finitos. O autor varia as possibilidades para o esquema advectivo, o tipo do acoplamento entre pressão-velocidade (segregado ou monolítico) e o impacto da linearização do termo advectivo. O sistema de equações linearizadas foi descrito como uma composição de submatrizes, de tal modo que, para resolvê-la de maneira monolítica, o complemento de Schur da submatriz correspondente aos termos difusivos e advectivos das equações da quantidade de movimento é aproximado por um precondicionador especializado.

O passo de tempo para cada configuração testada foi ajustado de modo que o resultado fosse o mesmo em todos os casos, tornando os experimentos comparáveis. Assim foi possível determinar qual configuração possui o melhor custo-benefício em termos de tempo de processamento para execução em um computador de mesa. Nesta situação, o autor recomenda sempre utilizar um método de integração temporal de segunda ordem, pois o ganho é significativo e o custo é desprezível, se comparado com Euler implícito de primeira ordem. Similarmente, o autor desencoraja explicitar os termos não lineares, pois a solução implícita destes termos permite passos de tempo 10 vezes maiores se comparado com soluções linearizadas. Finalmente, o autor recomenda utilizar o acoplamento monolítico somente no caso do número de Reynolds ser muito baixo, pois para um número de Reynolds alto, a vantagem do monolítico só seria visível no caso de malhas muito finas, o que exclui os casos que podiam, na época, ser executados em computadores de mesa.

Em Pernice e Tocci (2001), um procedimento *multigrid* geométrico é utilizado para resolver monoliticamente um escoamento incompressível 2D em uma malha deslocada. O sistema não linear é resolvido implicitamente através de um método de Newton inexato. A matriz jacobiana não é montada explicitamente. Ao invés disso, a solução aproximada para o resíduo do método de Newton é encontrada através de um método de subespaço de Krylov precondicionado com *multigrid* geométrico. A relaxação do *multigrid* é feita pelo próprio método SIMPLER (PERNICE; TOCCI, 2001 apud PATANKAR, 1981), que ao contrário do SCGS, foi especificamente projetado para levar em consideração os aspectos físicos do problema sendo resolvido.

Elman et al. (2008) descrevem um *framework* que generaliza e descreve algebricamente a utilização feita do método SIMPLER como relaxador do *multigrid* feita em Pernice e Tocci (2001). A matriz do sistema de equações é descrita por uma decomposição LDU, onde sua solução depende da inversão eficiente do complemento de Schur da decomposição. Neste contexto, o método SIMPLE pode ser visto como um precondicionador que aproxima o inverso do complemento de Schur. Outros precondicionadores para este objetivo foram estudados, incluindo o *least squares commutator* (LSC) (ELMAN et al., 2008 apud ELMAN et al., 2005), implementado na biblioteca de *software* PETSc e utilizado no presente trabalho.

Seguindo uma linha de desenvolvimento similar a Deng et al. (1994), Wachem e Gopala (2006) também utilizam uma abordagem em malha colocalizada e estabilizada pela interpolação de Rhie-Chow para resolver monoliticamente o acoplamento entre pressão velocidade em um escoamento incompressível. Neste trabalho, essa abordagem é utilizada para simular escoamentos bifásicos por questões de estabilidade, pois os métodos segregados se mostram instáveis para tratar o gradiente de massa específica que surge na fronteira entre os fluidos, bem como os termos fontes com valores numéricos grandes.

Também com uma formulação colocalizada em malha não estruturada, Darwish, Sraj e Moukalled (2009) desenvolveram uma abordagem para a solução monolítica de escoamentos incompressíveis. Mas diferentemente Wachem e Gopala (2006), a formulação não contempla o termo temporal, portanto objetiva somente escoamentos em estado permanente. O método é baseado em uma variação do algoritmo SIMPLE para malhas colocalizadas, que utiliza a interpolação de Rhie-Chow para estabilizar a solução, e deriva uma equação para a pressão em cada volume da malha. O termo advectivo é linearizado na discretização por volumes finitos, de modo que o problema seja representado por um sistema linear. Por utilizar as equações da pressão do SIMPLE ao invés das equações da continuidade, o sistema gerado é bem condicionado e com dominância diagonal, e pôde ser resolvido por *solver* linear iterativo. O número de iterações da solução monolítica se mostrou quase independente do tamanho da malha, ao contrário da solução segregada com o SIMPLE, o que gera uma economia de tempo de processamento de várias ordens de magnitude do método monolítico sobre o segregado, especialmente nas malhas mais refinadas.

Chen e Przekwas (2010) estenderam o trabalho de Darwish, Sraj e Moukalled (2009) para escoamentos 3D, em malha estruturada ou não estruturada, assim como estenderam a aplicabilidade do método para além do regime incompressível, incluindo o regime compressível subsônico. Os autores incluem as variáveis de temperatura no sistema linear monolítico resolvido, porém somente para efeito de conveniência na solução do sistema linear, pois a solução da temperatura não está acoplada diretamente pelo equacionamento com a solução da velocidade e pressão. Em vez disso, a convergência da temperatura com a pressão e velocidade é feita iterativamente, utilizando uma equação de estado para traduzir o campo de temperatura calculado em cada iteração para o campo de massa específica que é usado nas equações do movimento para escoamentos compressíveis.

Em Darwish e Moukalled (2014), a abordagem apresentada em Darwish, Sraj e Moukalled (2009) também é estendida para mais regimes de escoamento além do incompressível, mas ao contrário de Chen e Przekwas (2010), as equações da energia térmica também são diretamente acopladas às equações do movimento no sistema linear através da equação de estado do gás ideal — que por ser linear, mantém a linearidade do sistema de equações. Por utilizar um acoplamento forte entre a temperatura e as outras

variáveis, o método consegue representar além de escoamentos compressíveis subsônicos — como em Chen e Przekwas (2010) — também escoamentos transônicos e supersônicos, sendo portanto adequado a todos os regimes de velocidade.

Xiao, Denner e Wachem (2017) também resolveram de maneira monolítica escoamentos em todos os regimes de velocidade, em malhas não estruturadas. Entretanto, o método por eles apresentado é adequado tanto para soluções transientes quanto para escoamentos compressíveis permanentes, ao contrário de Darwish e Moukalled (2014), cujo resultado de escoamentos a alto número de Mach só possuem relevância física em regime permanente. A equação da quantidade de movimento é linearizada, e a pressão e velocidade são resolvidos de maneira monolítica em um único sistema linear. A discretização da continuidade é feita através de Rhie-Chow, que é suficiente para tornar o sistema linear tratável por um *solver* iterativo, pois um termo dependente da pressão dá dominância diagonal para a matriz. A equação da entalpia total é resolvida separadamente, e o acoplamento pressão e massa específica é feito iterativamente.

Podemos distinguir na presente revisão da literatura três linhas de desenvolvimento para solução monolítica das variáveis primárias do problema. São elas:

- Solução em malha deslocada, com a solução do sistema global de equações decomposta na solução em blocos de variáveis primárias, sendo cada bloco referente a um elemento da malha (CARETTO; CURR; SPALDING, 1972; VANKA, 1986);

- Solução em malha colocalizada, com algum mecanismo de suavização da pressão — como a interpolação de Rhie-Chow — aplicada às equações da continuidade, de modo a conferir dominância diagonal à matriz do sistema linear, para que este possa ser resolvido com *solvers* iterativos de propósito geral (DENG et al., 1994; WACHEM; GOPALA, 2006; DARWISH; SRAJ; MOUKALLED, 2009; CHEN; PRZEKWAS, 2010; DARWISH; MOUKALLED, 2014; XIAO; DENNER; WACHEM, 2017);

- Decomposição algébrica das matrizes do sistema de equações, aproximação do inverso do complemento de Schur através de um precondicionador especializado, e solução por um procedimento iterativo até a convergência, possivelmente acelerado por um método de subespaço de Krylov (TUREK, 1996; PERNICE; TOCCI, 2001; ELMAN et al., 2008).

O presente trabalho se encaixa na terceira categoria, onde os sistemas lineares do problema são resolvidos através de um precondicionador para aproximação do inverso do complemento de Schur. Essa abordagem é descrita em detalhes no capítulo 10.

2.2 Paralelismo

O fator limitante para a solução de problemas de CFD em um único computador foi alterado ao longo dos anos. Graças a evolução da tecnologia de construção de computadores, que tem proporcionado um crescimento exponencial do número de transistores por *chip*, a limitação deixou de ser memória — quando a resolução espacial necessária à solução de um problema não cabia fisicamente na memória de um único computador — e passou a ser o tempo de processamento, de modo que um problema suficientemente grande necessita de um número proporcionalmente grande de unidades de processamento para ser resolvido em um tempo viável.

Essa mudança se deu no momento em que, por limitações físicas, o aumento da frequência de *clock* dos processadores não conseguiu mais acompanhar a miniaturização do processo de fabricação de *chips*. Desde então, os fabricantes de processadores tentam compensar essa limitação com o aumento da complexidade do *hardware* (OLUKOTUN; HAMMOND, 2005). Já que não é possível aumentar o número de operações lógicas que cada circuito realizava por segundo, muitas das estratégias adotadas tentam maximizar o número de circuitos que trabalham simultaneamente, tirando proveito de todos os níveis de paralelismo possíveis.

Rapidamente, o custo da melhoria da *performance* das aplicações foi repassado aos programadores, na forma de processadores com múltiplas unidades de processamento. Assim, mesmo que o *software* seja executado em um computador de mesa, ele deve agora ser desenvolvido para executar em paralelo se quiser aproveitar toda a capacidade do processador (SUTTER; LARUS, 2005). Essa tendência de certa forma serviu para unificar as duas correntes de desenvolvimento existentes na comunidade de CFD: de *software* paralelo para execução em *clusters* de computação, e a de *software* serial, com o objetivo de ser executado em uma estação de trabalho.

De repente, para ser obter a *performance* ótima de um *software* executando em um computador *workstation*, passou a ser necessário utilizar o mesmo tipo de técnica e ferramentas utilizadas para nos *cluster* de computação: a programação paralela. Enquanto que o desenvolvimento de propósito geral seguiu a tendência de se utilizar *threads* para novos desenvolvimentos e paralelização de códigos antigos, a comunidade de CFD seguiu, em geral, pelo caminho já conhecido e utilizado em *clusters* de computação multiprocessada, realizando a paralelização através do MPI.

Embora originalmente concebido como protocolo de passagem de mensagens entre computadores, o MPI se generaliza transparentemente para funcionar entre processos de um mesmo computador — que são executados paralelamente nos múltiplos núcleos de um mesmo processador — e até em ambientes híbridos, como um *cluster* formado por computadores multiprocessados, onde alguns processos da execução estão na mesma

máquina e outros são remotos.

A vantagem de se utilizar MPI é, além do aproveitamento de códigos antigos, a unificação do desenvolvimento tanto para *workstations* e *clusters*. O mesmo código-fonte serve para os dois, e a diferenciação do que pode ser tratado em um ou em outro se dá somente pelo tamanho do problema. A desvantagem é que utilizar *threads* permite alguns tipos de otimizações que não são possíveis através de MPI, pois, por padrão, toda a memória do programa é compartilhada entre todas as *threads*, o que teoricamente requer menos cópias de dados, ao preço de maior complexidade de programação e risco de *bugs* na sincronização entre as *threads*.

Alguns trabalhos seguiram pela linha de hibridizar *software* de computação científica originalmente desenvolvido no modelo MPI para funcionar simultaneamente com *threads* (GUO et al., 2012; WEILAND et al., 2012; GAHVARI et al., 2012). Os ganhos foram limitados e em situações específicas, havendo até perda de *performance* em alguns casos, se comparado com as versões originais, em MPI somente. Por ser difícil mostrar alguma vantagem teórica do MPI sobre *threads*, estes resultados desanimadores sugerem a dificuldade prática de se desenvolver e manter um modelo híbrido ótimo. Por esse motivo, os desenvolvedores do PETSc, objeto de desenvolvimento de Weiland et al. (2012), desencorajam o uso de *threads* com MPI em favor de MPI puro. Citando sua documentação *online*[1]:

> *The core PETSc team has come to the consensus that pure MPI using neighborhood collectives and the judicious using of MPI shared memory (for data structures that you may not wish to have duplicated on each MPI process due to memory constraints) will provide the best performance for HPC simulation needs on current generation systems, next generation systems and exascale systems. It is also a much simpler programming model then MPI + threads (leading to simpler code).*
>
> *Note that the PETSc team has no problems with proposals to replace the pure MPI programming model with a different programming model but only with an alternative that is demonstrably* better*, not with something more complicated that has not been demonstrated to be better nor that has any technical reason to be believed to be any better. Ever since the IBM SP2 in 1996 we've been told that "pure MPI won't scale to the next generation of machine", this has yet to be true and there is no reason to believe that it will be true.*

Uma situação onde o modelo híbrido é uma necessidade é em supercomputadores massivamente paralelos, como centenas de milhares de unidades de processamento, onde a memória por núcleo continua sendo o fator limitante — como no caso de Baker et al. (2012), com 4 GB de memória para cada 4 núcleos de processamento. Nesse caso, o uso de *threads* ajuda a economizar memória, pois a memória de um processo é compartilhada entre todas as suas *threads*, reduzindo a necessidade de duplicar dados. Mesmo assim, a

[1] Disponível em <https://www.mcs.anl.gov/petsc/miscellaneous/petscthreads.html>, acessado em 2018-05-28.

razão de *threads* por processos é um parâmetro sensível, que deve ser finamente ajustado para cada combinação de *hardware* e sistema operacional.

Se por um lado as ferramentas paralelas antigas conseguiram mostrar que paralelismo por MPI continua sendo uma solução viável nas arquiteturas paralelas modernas de computador, por outro, lado ferramentas originalmente concebidas para serem seriais foram adaptadas para poder aproveitar a capacidade total dos processadores atuais. Se a paralelização for feita utilizando MPI, a ferramenta ganha também a capacidade de ser executada em *clusters*, o que o torna uma escolha padrão na comunidade de CFD e de computação científica em geral. Este foi o caminho adotado dentro do nosso grupo de pesquisa com relação à ferramente MFSim[2], como descreve Lima (2012).

A paralelização da solução numérica de uma EDP é no máximo tão boa quanto o particionamento do domínio discreto. Shang (2014) faz uma comparação da qualidade do particionamento e o impacto de vários particionamentos na *performance* da simulação de um escoamento 3D em malha não estruturada com o Code_Saturne (ARCHAMBEAU; MÉCHITOUA; SAKIZ, 2004), uma ferramenta de CFD *open source*. O autor comparou várias bibliotecas de *software* e algoritmos para particionamento publicamente disponíveis. Os algoritmos de particionamento baseados no grafo de conectividade dos elementos da malha se mostraram os mais adequados nesta situação, conseguindo um ganho de *performance* de até 30 vezes o particionamento geométrico — baseado somente na posição geométricas dos elementos — originalmente disponível no Code_Saturne. No capítulo 5 é apresentado um estudo semelhante a este, porém aplicado à situação da HigTree, quantificando o impacto do particionamento na solução de um sistema linear. No capítulo 6 é detalhada a implementação, dentro da HigTree, da técnica de particionamento escolhida.

[2] Na época, a ferramenta era denominada AMR3D.

Parte I

Desenvolvimento e *Performance*

3 Escolha dinâmica de *solver* de sistemas lineares

A biblioteca de *software* HigTree possui uma interface matricial para resolver sistemas de equações lineares. Originalmente, esta interface se baseava internamente no pacote de código aberto PETSc para resolver iterativamente o sistema linear.

Para permitir que o *solver* de sistemas lineares usado por uma aplicação HigTree fosse alterado facilmente, eu separei esta interface da implementação interna, que agora pode ser provida por diferentes pacotes e algoritmos, sendo o PETSc somente um deles. Esta mudança é desejável por pelo menos duas razões:

- Durante a fase de desenvolvimento, é desejável que seja possível integrar facilmente ao HigTree um novo algoritmo de *solver* e/ou um pacote de *solver* de terceiros, possibilitando testar sua aplicabilidade e *performance* nos tipos de matrizes de sistemas lineares que surgem dos casos tratados por ele.
- Durante o uso da HigTree em produção, o usuário teria a possibilidade de escolher entre os vários *solvers* disponíveis aquele que melhor se encaixa ao seu caso de simulação e à sua infraestrutura de *hardware*.

A interface de *solver* do HigTree recebe o sistema linear na forma de uma matriz esparsa distribuída e um vetor com o lado direito das equações, particionados entre os processos MPI. Cada processo MPI mantem uma parte do domínio do problema, e cada um é responsável por prover, independentemente, os coeficientes e os elementos do lado direito para o seu subdomínio.

Originalmente, a funções da interface da HigTree simplesmente encapsulavam as funções do PETSc, o que significa que cada função de *solver* na HigTree simplesmente chamavam a função correspondente do PETSc. Os dados passados de forma distribuída ao PETSc eram organizados nas suas estruturas de dados internas, que também são distribuídas.

Para desacoplar a interface do PETSc, as assinaturas das funções originais foram mantidas — tanto quanto possível — iguais, de modo a impactar o mínimo possível com o código existente. As diferenças foram na função de criar um *solver*, que agora recebe um parâmetro extra que diz qual biblioteca de *solver* usar, e na função para ler a solução do sistema linear, que retornava uma estrutura de dado interna ao PETSc. Ela foi alterada para, ao invés de retornar um objeto definido pelo PETSc, ela carrega a solução diretamente na estrutura de dados `distributed_property` da HigTree, para ser usado na

simulação. Esta mudança foi necessária para garantir que a nova interface do *solver* fosse independente de implementação, de modo que o *solver* pudesse ser trocado sem precisar mudar como o resto do código que interage com ele.

Ao invés de chamar o PETSc diretamente, as funções da interface agora passam por mais um nível de indireção, chamando a função de implementação real para a biblioteca de solver sendo utilizada. As funções de implementação do PETSc é que são agora responsáveis por interagir com ele, e cada alternativa de *solver* possui seu próprio conjunto de funções de implementação. Todas as funções de implementação com um mesmo papel estão restritas a uma mesma assinatura e definição. Por exemplo, todas as implementações da função para carregar o lado direito do sistema linear (existe uma para cada uma das diferentes implementações de *solvers*) devem receber parâmetros do mesmo tipo e na mesma ordem.

Quando uma aplicação chama uma função de *solver* da HigTree, como por exemplo `void slv_set_bi(solver *s, int i, real v);`, ela, por sua vez, chama o ponteiro de função para a implementação real — armazenado no objeto `solver` da HigTree — passando para ele os mesmos parâmetros recebidos: `s`, `i` e `v`. A função de implementação então converte o parâmetro `s` em um ponteiro para sua estrutura de dados interna, particular àquele tipo de *solver*, contendo as variáveis e estruturas de dado específicas àquela implementação. A função então usa estes dados para realizar sua tarefa.

A conversão entre o tipo `solver *` e um tipo interno, específico por implementação, funciona porque este tipo interno tem um atributo `solver s;` como o primeiro elemento de sua `struct`. A linguagem C garante que o ponteiro para uma estrutura de dados tem o mesmo endereço que o ponteiro para o seu primeiro elemento, e deste modo, usando esta técnica, herança de tipos pode ser implementada em C.

Além de prover uma estrutura de dados principal herdando do tipo básico `solver`, cada *solver* implementado deve prover:

- Uma função de implementação que se encaixe na assinatura de cada função de indireção na interface de *solver* geral;
- Uma instância global e constante da estrutura `struct_solver_vtable`[1] contendo um ponteiro para cada função de implementação dada por aquela implementação;
- Uma função de criação globalmente visível, que recebe quaisquer parâmetros necessários à instanciação daquele tipo de *solver* e preenche sua estrutura de dados privada. Esta função retorna o ponteiro para o primeiro elemento `solver s;` nela contido, que será posteriormente passada para todas as funções da API do *solver*.

[1] O nome `vtable` vem de *virtual table*, a estrutura de dados usada por linguagens orientadas a objeto para associar dinamicamente chamadas de funções com suas implementações em um tipo de dado polimórfico.

Durante o desenvolvimento, a primeira opção de *solver* implementada foi o próprio PETSc, já que era simplesmente uma questão de reorganizar as funções existentes para o novo formato de interface/implementação.

Outra opção de *solver* implementada foi o Hypre (FALGOUT; YANG, 2002). Hypre é uma biblioteca de *software* livre usada internamente pelo PETSc como um provedor de vários precondicionadores distribuídos, utilizados pelos métodos de subespaço de Krylov do PETSc. Dentre os precondicionadores providos pelo Hypre estão a fatoração LU incompleta e o *Multigrid* algébrico (AMG).

O Hypre também pode ser usado por conta própria, como um resolvedor completo de sistemas lineares, pois ele também provê implementações distribuídas para os aceleradores de Krylov, como gradiente biconjugado estabilizado (BiCGSTAB) e o método generalizado de minimização de resíduo (GMRES).

Tanto PETSc quanto Hypre têm uma implementação simples por trás da interface da HigTree, pois ambos são bibliotecas distribuídas via MPI, logo é uma questão de simplesmente traduzir as chamadas da HigTree para suas funções equivalentes.

Os outros dois *solvers* implementados eram centralizados, e necessitavam que a matriz esparsa e o vetor do lado direito estivessem inteiramente contidos em um único processo. Uma vez que eles possuíam esta característica em comum, outra interface para *solvers* foi adicionada: a interface para *solvers* centralizados. Este *pseudo-solver* implementa todas as funções de indireção da interface de *solvers* distribuídos da HigTree, mas ao invés de resolver o sistema, ele somente organiza todas as entradas recebidas em um único processo MPI, e então chama o *solver* real a partir dele. Este *pseudo-solver* construído sob a interface de *solvers* distribuídos da HigTree simula um solver distribuído a partir de um solver centralizado. Ele permite que *solvers* monoprocessados sejam facilmente integrados dentro da HigTree como os *solvers* distribuídos via MPI.

O primeiro *solver* centralizado implementado foi o ViennaCL, que é uma biblioteca contendo múltiplos algoritmos implementados em OpenCL e em CUDA, que rodam na GPU, sendo potencialmente mais rápidos que algoritmos feitos para CPU.

O outro *solver* centralizado é uma implementação *multithread* do método de sobrer-relaxação sucessiva (SOR). O código havia sido previamente desenvolvido internamente no grupo, e foi incluído na HigTree com propósitos de comparação.

A função para criação de um *solver*, `solver* slv_create(SolverEngine engine, int first_gid, size_t size)`, passou a ter o parâmetro extra `SolverEngine engine`, que recebe uma das seguintes opções:

- `SOLVER_ANY`,
- `SOLVER_HYPRE`,

- `SOLVER_PETSC`,
- `SOLVER_SOR`,
- `SOLVER_VIENNACL`.

Este novo parâmetro indica qual implementação de *solver* deverá ser criada — exceto o `SOLVER_ANY`, que tem um significado especial. Quando `SOLVER_ANY` é especificado, o programa procura pela variável de ambiente `HIGTREE_SOLVER_ENGINE` e escolhe qual *solver* utilizar dependendo se o valor da variável for um dos seguintes valores: `hypre`, `petsc`, `sor` ou `viennacl`. Se o *solver* também não puder ser determinado pela variável de ambiente, um solver padrão razoável para propósito geral é escolhido. Atualmente o padrão é o PETSc.

4 Escrita de dados em XDMF

Originalmente, a HigTree provia suporte subótimo à escrita de arquivos para visualização em arquivos VTK. A biblioteca em si continha a funcionalidade para representar a malha do domínio em VTK, e as aplicações eram responsáveis pela escrita dos valores das propriedades. Essa abordagem possuía as seguintes desvantagens:

- Repetição de código, pois como parte das funcionalidades não fazia parte da biblioteca, ela tinha que ser reimplementada ou copiada em cada nova aplicação que utilizasse a HigTree. É o caso dos vários exemplos distribuídos junto com seu código-fonte, todos reimplementando a escrita das variáveis da simulação no formato VTK;
- Codificação ineficiente em texto decimal. Uma variável do tipo ponto flutuante de precisão dupla pode ser representada em exatamente 8 *bytes*, porém quando representada com base decimal em texto, o mesmo tipo de variável necessita de no mínimo 21 *bytes* para ser representado exatamente. Portanto, uma representação em texto decimal de resultados numéricos é necessariamente ou maior ou menos precisa que uma representação binária (frequentemente ambos). No caso particular cada valor numérico era representado com 9 *bytes*, portanto, pouco maior que a versão binária, porém muito menos preciso.
- Sem compressão de dados. Essa é uma questão de economia de espaço de armazenamento, pois resultados de simulações facilmente chegam a ocupar múltiplos *gigabytes*, quanto de *performance*, pois a escrita física dos dados no dispositivo de armazenamento geralmente é muito mais lenta que seu tratamento no processador, e ao se diminuir a quantidade de dados escrita utilizando um algoritmo de compressão, a velocidade do procedimento como um todo fica maior.
- Repetição de informação armazenada. Cada passo de tempo era armazenado em um arquivo independente, de modo que toda a malha e topologia dos elementos era reescrita em cada arquivo, mesmo no caso de a malha ser idêntica. Além disso, na descrição dos elementos retangulares da malha, os vértices comuns entre vários elementos não eram identificados e compartilhados. Em vez disso, um mesmo vértice era descrito múltiplas vezes, uma para cada elemento de qual fazia parte.

Para resolver estas limitações, seria necessário um formato de arquivo que obedecesse a vários requisitos. Ele precisaria:

- trabalhar com domínios não uniformes, suportando refinamento hierárquico;

- ser codificado em binário;
- ter suporte a referências para outros arquivos, para evitar ter de escrever a malha do domínio a todo passo de tempo;
- suportar algoritmos de compressão;
- suportar a escrita em paralelo por diferentes processos;
- ser suportado pelas ferramentas de visualização VisIt (CHILDS et al., 2012) e Paraview (AYACHIT, 2015), usadas pelo nosso grupo de pesquisa.

O formato escolhido foi o XDMF com HDF5, que além que cumprir com todos os requisitos, já é utilizado pelo grupo, onde a prévia experiência com essas tecnologias aceleraria o trabalho de desenvolvimento. Assim como o formato VTK, esse novo formato tipo de escrita de dados também é limitado a $D \in \{2, 3\}$, ao contrário do resto da HigTree que pode trabalhar com dimensões arbitrárias.

Um passo de tempo é representado por um único arquivo XML, que contém a estrutura XDMF, e um arquivo HDF5 para cara processo envolvido na execução. Os valores das variáveis, e quando necessário, a geometria da malha, são armazenados no arquivo HDF5, compactados com o algoritmo de compressão especializado *Multidimensional Adaptive Filtering Improved Scientific data Compression* (MAFISC) (HÜBBE; KUNKEL, 2013), quando disponível. Alternativamente, o algoritmo de compressão de propósito geral Gzip (DEUTSCH, 1996) é utilizado.

O problema de representar o domínio hierarquicamente refinado foi resolvido com a utilização do tipo de topologia "`Hexahedron`" do XDMF (ou, no caso 2D, "`Quadrilateral`"), que permite a representação de malhas arbitrárias compostas por essa primitiva geométrica. A posição dos vértices e sua conectividade para formar os hexaedros são armazenados em *datasets* independentes dentro do arquivo HDF5, onde o *datasets* é o termo utilizado no manual do HDF5 para descrever um vetor multidimensional.

As posições são armazenadas como uma sequência de vetores 3D (ou 2D) de pontos flutuantes, a conectividade como uma sequência de inteiros, agrupados de 8 em 8 (ou 4 em 4, no caso 2D), indexando os vetores do outro *dataset* que são os vértices que compõem cada primitiva.

A escrita da malha é feita elemento por elemento, onde primeiro os vértices daquele elemento são potencialmente escritos em um *dataset*, e então, em outro *dataset* é escrita a lista dos índices correspondentes aos vértices que compõe o hiper-retângulo daquele elemento.

Antes de um vértice ser escrito, ele é buscado em uma tabela *hash* contendo os vértices anteriormente escritos no *dataset*. Se este vértice não for encontrado, significa que

é a primeira vez que ele é visto. Então ele é escrito no *dataset* e inserido na tabela *hash*, tendo como valor sua posição de inserção no *dataset*. Se, por outro lado, aquele vértice em particular for encontrado na tabela, significa que já está escrito no *dataset*, então o valor já armazenado na tabela é usado para referenciar ao vértice na escrita da conectividade do elemento, no outro *dataset*. Isso garante que nenhum ponto repetido será escrito no *dataset* das posições dos vértices, economizando, assim, espaço.

Cada propriedade da simulação é também escrita no seu próprio *dataset* dentro do arquivo HDF5, com o seu nome dado pelo usuário da biblioteca do HigTree. Isso permite que um número arbitrário de diferentes propriedades sejam gravados para visualização. As propriedades também podem ser agrupadas em tipos fisicamente significantes: do mesmo modo que faz sentido escrever a pressão como um campo escalar, pode ser razoável agrupar as D componentes da velocidade como um único campo vetorial, e ser assim entendido pelo programa de visualização.

Uma vez que a maior parte dos dados estejam armazenados em um formato binário compactado dentro dos arquivos HDF5, o arquivo XML faz referência a estes arquivos e aos seus *datasets* individuais. A lógica e a interpretação dos dados armazenados em HDF5 é dada pelo arquivo XML, que obedece às regras sintáticas e semânticas definidas pelo padrão XDMF.

Para evitar ter de escrever a malha do domínio a todo passo de tempo — nas situações em que ela não é alterada — somente os valores das propriedades são escritos nos arquivos HDF5 do passo de tempo. Para descrever a malha, o arquivo XML simplesmente referencia os arquivos HDF5 escritos anteriormente. Nesta situação, um mesmo arquivo XML referencia dois conjuntos de arquivos HDF5: o do tempo presente, contendo os valores das variáveis, e o de um tempo anterior, contendo a malha.

A escrita em paralelo é alcançada com cada processo escrevendo independentemente seu arquivo HDF5, e então o processo MPI de *rank* 0 escreve o arquivo XML, referenciando a todos os arquivos HDF5. O desequilíbrio de tarefas de um único processo escrever todo o arquivo XML tem custo insignificante, porque o custo da escrita do XML é muito menor que o custo da escrita dos arquivos HDF5, que contém o grosso dos dados. O tamanho de cada arquivo XDMF cresce proporcionalmente ao número de processos MPI participando da execução, enquanto o tamanho do HDF5 cresce proporcionalmente ao número de elementos no subdomínio daquele processo.

Enquanto que a implementação da funcionalidade para simulações 3D ocorrera sem problemas, a mesma funcionalidade em 2D necessitou de um tratamento especial. Por alguma razão, o Paraview espera vetores 3D mesmo quando a simulação tem um domínio bidimensional, e o XDMF não provê nenhum meio padrão de especificar a dimensionalidade do campo vetorial. Então, para fazer uma saída vetorial 2D funcionar corretamente no Paraview, foi dado o valor 0 à terceira componente de todos os vetores.

Métrica	VTK	XDMF	Razão
Tamanho	132,5 MB	13,7 MB	9,67
Tempo de escrita	6,70 s	3,80 s	1,76

Tabela 1 – Ganhos do novo formato de saída de dados.

Para prover este valor no caso 2D, foi criado dentro do arquivo HDF5 que contém a malha um *dataset* especial. Esse *dataset* é esparso: possui tamanho definido, porém nenhum dado gravado. Pelo padrão do HDF5, tal *dataset*, quando lido, retorna o valor 0 para todos os elementos não definidos (que, no caso, é a sua totalidade). Este *dataset*, por não conter dados, ocupa um tamanho fixo de poucos *bytes* no arquivo HDF5, e portanto não impacta no tamanho final da saída de dados. O arquivo XML define as duas primeiras componentes do campo vetorial a partir do *dataset* real, contendo os valores das variáveis, e a terceira componente é tirada do *dataset* esparso.

Para comparar as vantagens do novo sistema sobre o antigo, foram escritos 40 passos de tempo de uma execução com 13952 elementos 2D, em um domínio complexo formado por múltiplas árvores, o mesmo mostrado na Figura 11. Os dados mostrados na Tabela 1 mostram ganhos significativos tanto em *performance* quanto em espaço em disco. É importante salientar que o tempo aqui dado é o somente o tempo de escrita em disco, que pode ser desprezível frente ao tempo de computação de uma simulação completa.

5 Estudo dos métodos de partição de domínio

Numa solução numérica paralela de EDPs, o ganho em relação à execução serial é no máximo tão bom quanto o particionamento do domínio entre as várias unidades computacionais. Um bom particionamento possui duas características: a primeira é que a distribuição dos elementos do domínio é bem balanceada entre os subdomínios. Ou seja, nenhum subdomínio tem muito mais ou muito menos elementos que os outros. Idealmente, todos devem ter o mesmo número de elementos.

A segunda caraterística do bom balanceamento é que o número de elementos vizinhos que são separados pela partição deve ser baixo. Podemos definir esse número como a conectividade do particionamento: quanto mais células conectadas entre diferentes subdomínios, maior a conectividade do particionamento. Quanto maior a conectividade de um particionamento, mais dados serão trocados entre os subdomínios, refletindo diretamente no custo de comunicação. Idealmente, o número de vizinhos separados deve ser o mínimo possível. A Figura 3 mostra três possíveis particionamentos para um domínio, sendo que somente um deles é considerado bom.

Frequentemente, estes dois objetivos são contraditórios, portanto um bom algoritmo de particionamento deve conseguir obter um bom compromisso entre balanceamento e minimização de comunicação.

Como adiantado na seção 1.1.7, o algoritmo de partição de domínio implementado originalmente na HigTree era provisório, pois não tinha suporte a domínios compostos por múltiplas árvores. Antes de implementar seu novo mecanismo de particionamento — descrito no capítulo 6 — conduzi um estudo sobre as várias soluções já existentes para o problema do particionamento com os seguintes objetivos:

- determinar impacto do particionamento no custo computacional da solução de um sistema linear criado a partir de uma EDP discretizada;
- determinar a aplicabilidade do algoritmo à HigTree e estimar seu o custo de implementação.

O custo computacional na solução de um sistema linear é uma métrica importante, pois é a etapa mais cara, em termos de comunicação entre processos, dos vários métodos disponíveis para a solução numérica das equações de Navier-Stokes.

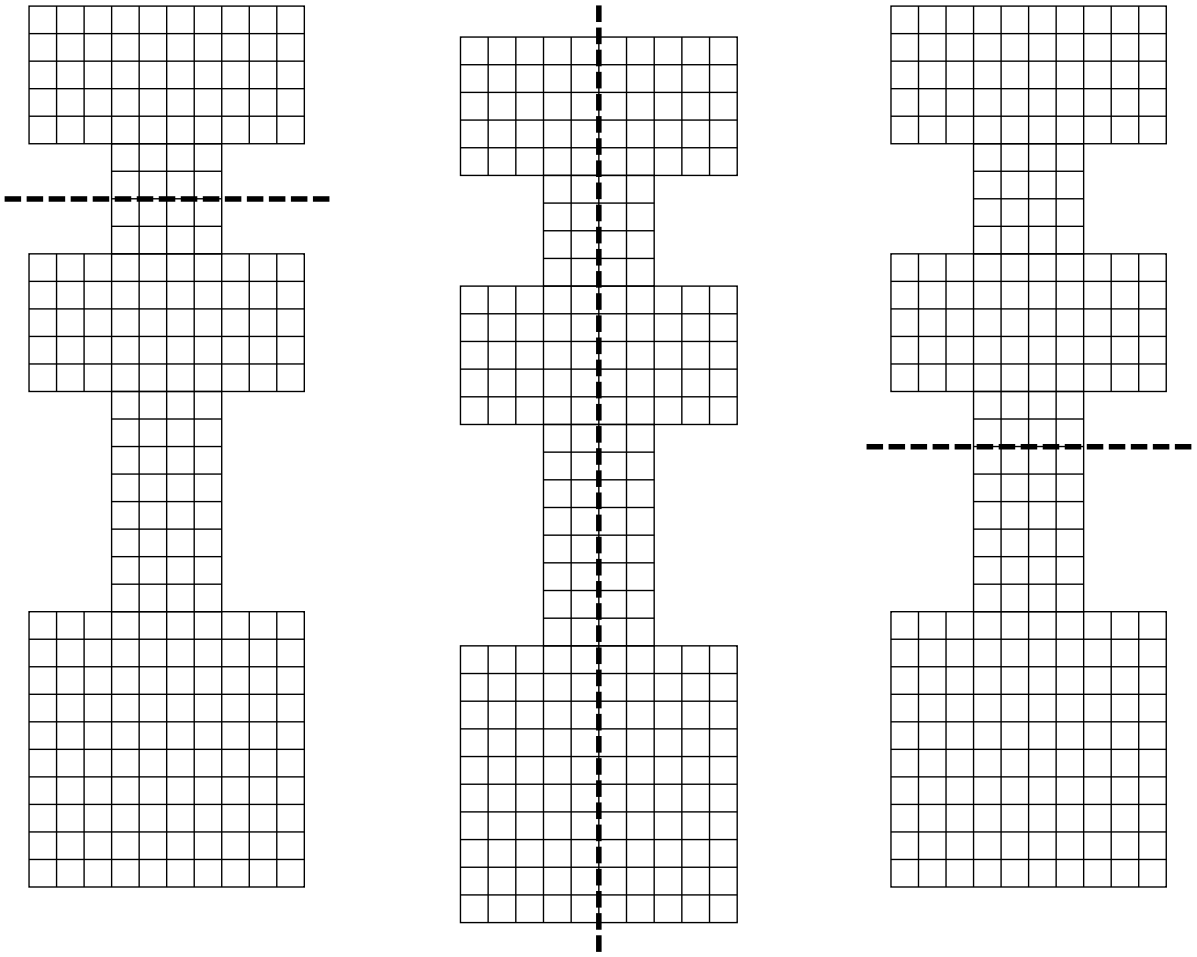
Com base nestes resultados, poderíamos determinar qual dos algoritmos seria implementado, considerando os objetivos e recursos disponíveis no escopo do projeto de

Figura 3 – Várias possibilidades de particionamento de um domínio discreto em 2 subdomínios.

(a) Particionamento ruim por ser desbalanceado. A conectividade é baixa, porém a partição inferior possui muito mais elementos que a partição superior.

(b) Particionamento ruim por ter conectividade muito alta, apesar do número de elementos ser perfeitamente balanceado entre os 2 subdomínios.

(c) Particionamento bom. Número de elementos perfeitamente balanceado entre as duas partes, e a conectividade é a menor possível.



desenvolvimento da HigTree.

Este estudo também era de interesse para o desenvolvimento do MFSim — antigo AMR3D — outra ferramenta de CFD desenvolvida pelo nosso grupo de pesquisa no escopo do mesmo projeto de cooperação. Ele está em um estágio de desenvolvimento mais maduro, mas utiliza uma abordagem de malha adaptativa bloco estruturada (LIMA, 2012), que é diferente da malha hierárquica da HigTree.

Foram estudadas três ferramentas disponíveis publicamente para realizar o particionamento dinâmico — e em paralelo — do domínio da simulação. Foram elas:

ParMETIS: (KARYPIS; SCHLOEGEL, 2011) Um pacote de *software* para particiona-

mento de grafos e malhas, reescrito em paralelo a partir do pacote METIS, feito pela *University of Minnesota Twin Cities*;

PT-Scotch: (CHEVALIER; PELLEGRINI, 2008) Um pacote de *software* para particionamento de grafos feito pelo *Laboratoire Bordelais de Recherche en Informatique*, extensão em paralelo do pacote Scotch;

Zoltan: (DEVINE et al., 2002) Um pacote de *software* para particionamento paralelo, balanceamento de carga e serviços de gerenciamento de dados felo pelo *Sandia National Laboratories*. Além de algoritmos próprios, ele também inclui uma interface comum para o ParMETIS e o PT-Scotch.

Para realizar este estudo, foi escrito um programa que lê, de um conjunto de arquivos, um sistema linear discretizado construído a partir de uma EDP, a posição de cada nó discreto (correspondente à cada linha da matriz). O programa então reparticiona o sistema linear com o algoritmo de particionamento sendo testado, para então resolvê-lo.

O sistema linear usado no estudo foi tirado do primeiro passo de tempo do programa `read-amr-solve-cavity-ns-example-3d` do pacote da HigTree. O programa foi adaptado para, além de escrever o sistema linear do acoplamento pressão-velocidade, também escrever em arquivo a posição dos elementos da malha, e sua partição original usada dentro do programa. A malha utilizada foi um cubo com 3 níveis de refinamento e mais de 3 milhões de elementos. Essa malha foi originada do refinamento adaptativo em uma simulação de camada de mistura realizada no MFSim.

5.1 Algoritmos de Particionamento

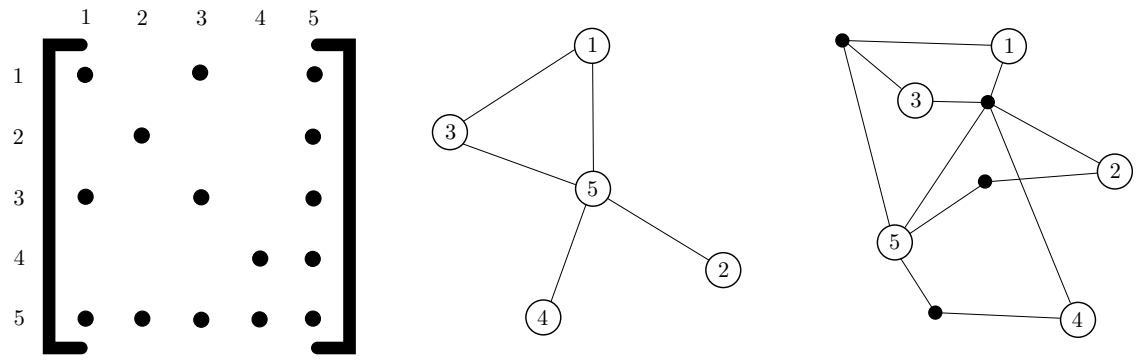
Os particionamentos testados podem ser organizados em 4 categorias:

Particionamento geométrico é aquele que é feito usando como base somente o peso e a posição geométrica dos elementos. Eles são mais rápidos de serem computados, mas produzem resultados teoricamente piores, pois não levam em conta os custos de comunicação entre os elementos. Como entrada, somente a posição e o peso de cada elemento são passados.

Particionamento em grafo é aquele que leva em conta o peso e a conectividade entre pares de elementos, sendo os custos de comunicação mapeados através desta conectividade. São teoricamente melhores, mas mais lentos de serem computados, gastam mais memória e são mais complexos de se implementar e usar, quando comparado com os geométricos. Como entrada, cada elemento é dado como um vértice do grafo, cada um contendo um peso. Para cada linha da matriz do sistema linear, uma aresta

Figura 4 – Matriz esparsa e seus correspondentes grafo e hipergrafo de conectividade.

(a) Elementos não nulos. (b) O grafo. (c) O hipergrafo.



é criada entre a diagonal principal e cada elemento não nulo da diagonal principal, indicando uma dependência de dados, como na Figura 4.

Particionamento em hipergrafo é aquele que generaliza o conceito matemático de grafo para levar em conta a conectividade simultânea entre grupos de elementos. Hipergrafos são capazes de descrever acuradamente os custos de comunicação de uma computação paralela (CATALYUREK; AYKANAT, 1999), e portanto seriam os que, teoricamente, produzem os melhores resultados. Comparado com os algoritmos em grafo, ao invés de múltiplas arestas por linha, cada linha da matriz do sistema linear é descrita como uma única hiperaresta, contendo todos os vértices correspondentes às colunas não nulas, como mostrado na Figura 4c.

Particionamento imediato foi a categoria atribuída ao algoritmo que não utiliza nenhuma informação em particular do problema além do peso de cada elemento. Possui implementação muito simples e foi utilizado somente para referência.

Os algoritmos utilizados nos testes foram:

- Algoritmo de particionamento imediato:

  Particionamento uniforme foi o algoritmo mais simples utilizado. O sistema linear é dividido entre os processos na sequência em que ele é carregado dos arquivos: as primeiras linhas são dadas ao primeiro processo, as linhas subsequentes são dadas ao segundo processo, e assim sucessivamente, até que, ao final, a partição está perfeitamente balanceada, mas sem levar em consideração nenhuma característica inerente ao problema. Foi implementado somente como referência.

- Algoritmos de particionamento geométricos:

***Recursive Coordinate Bisection* (RCB)** do Zoltan é um algoritmo que, a cada iteração, reparte o domínio em dois, separando por um plano alinhado aos eixos coordenados (por isso de "*coordinate*" no nome) de modo que o peso dos elementos em cada lado seja o mais equilibrado possível. Essa iteração é aplicada recursivamente a cada uma das metades, até se obter o número desejado de subdomínios (BERGER; BOKHARI, 1987).

***Recursive Inertial Bisection* (RIB)** do Zoltan é como o RCB, com a diferença que o plano de corte não é necessariamente ortogonal. Em vez disso, o eixo com maior momento de inércia é calculado, e o plano de corte é perpendicular a ele (WILLIAMS, 1991), permitindo que a divisão de peso entre os dois subdomínios seja melhor equilibrada que no RCB.

***Hilbert Space-Filling Curve* (HSFC)** do Zoltan utiliza uma extensão em $\mathbb{R}^3$ da curva de Hilbert para mapear a posição de todos os elementos no espaço em uma sequência bem definida (PILKINGTON; BADEN, 1994). A curva de Hilbert é uma curva fractal contínua que possui duas propriedades interessantes:

1. ela passa por todos os pontos de um volume. Isso significa que qualquer ponto dentro daquele volume está em alguma posição determinada da curva (ou, em outras palavras, tem uma coordenada definida no espaço da curva);
2. pontos do volume próximos entre si tendem a ficar próximos em coordenadas da curva.

O algoritmo se resume a determinar em qual posição relativa à curva está o centro de cada elemento, e então é feito um particionamento uniforme, agrupando os elementos por intervalos dentro da curva, como na Figura 5. Na figura, a malha é dividida em 4 partes perfeitamente balanceadas, cada uma contendo 25 elementos. A curva tracejada é uma representação grosseira da curva de Hilbert — suficiente para demonstrar sua aplicação no algoritmo — porém a curva em si não pode ser completamente representada graficamente, pois ela passa por todos os pontos da superfície. A representação gráfica pode ser refinada quantas vezes forem necessárias: ela tende à curva real à medida que o número de refinamentos tende ao infinito.

***Space-Filling Curve* (SFC)** do ParMETIS é uma implementação diferente do HSFC do Zoltan. A curva utilizada não é necessariamente a mesma, pois a extensão da curva de Hilbert para $\mathbb{R}^3$ não é única (HAVERKORT, 2011).

Original da HigTree é o particionamento que foi gravado em arquivo juntamente com o sistema linear. É considerado um particionamento geométrico porque, antes do sistema linear ser gravado, o particionamento foi gerado dentro da HigTree por um algoritmo de bisseção recursivo (como o RCB). Logo, este

particionamento não é feito durante a execução do teste: a divisão dos elementos entre os processos já está calculada, e é lida de um arquivo.

- Algoritmo de particionamento em hipergrafo:

  ***Parallel Hypergraph Partitioning* em Hipergrafo (PHG-H)** do Zoltan foi o único particionador de hipergrafo utilizado. O algoritmo funciona em múltiplos níveis, aglutinando vértices conectados para formar vértices de um hipergrafo mais grosseiro. O particionamento é feito do grafo mais grosseiro para o mais detalhado (DEVINE et al., 2006).

- Algoritmos de particionamento em grafo:

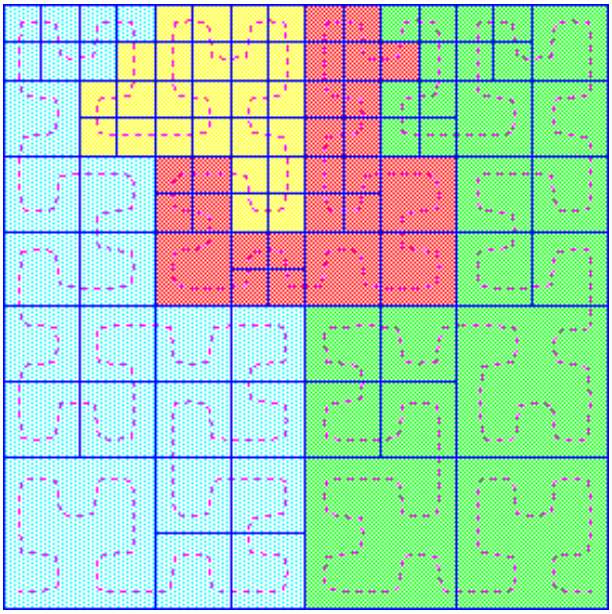
  PT-Scotch é um pacote *software* que provê um particionador em grafo paralelo. Seu funcionamento é parecido com o RCB no sentido em que a malha será recursivamente dividida em duas, entretanto, o corte que separa os subdomínios não é geométrico. Em vez disso, os vértices do grafo são distribuídos entre os dois subdomínios de modo a minimizar o número de arestas atravessando de um lado a outro, e portanto, minimizando a quantidade de comunicação que acontece entre fronteiras (CHEVALIER; PELLEGRINI, 2008).

  Multilevel K-Way Partitioning do ParMETIS é um algoritmo de particionamento de grafos que funciona em múltiplos níveis (KARYPIS; KUMAR, 1999), de modo similar ao PHG. Entretanto, ao contrário do PHG, não funciona sobre hipergrafos.

  ***Parallel Hypergraph Partitioning* em Grafo (PHG-G)** do Zoltan é o mesmo algoritmo base usado no particionamento de hipergrafos, mas executado sobre um grafo simples. Como um hipergrafo é uma generalização do conceito de grafo, todo grafo também é um hipergrafo. Deste modo, o mesmo grafo simples que é passado aos outros algoritmos também pode ser usado no PHG, e funcionará normalmente. Entretanto, não terá qualquer vantagem que a descrição mais detalhada do hipergrafo possa oferecer.

  Cuthill-McKee reverso (RCM) é um algoritmo de ordenação de matriz para minimização de banda. A ordenação dada para as linhas da matriz pelo algoritmo foi usada para ordenar os elementos da malha, que então foram divididos em intervalos balanceados entre os processos, como é feito no HSFC e no particionamento uniforme. A ideia ao propor esse uso para o RCM foi de que a minimização da banda da matriz global agruparia os elementos com grande conectividade entre si, e portanto a comunicação seria reduzida, quando uma matriz assim ordenada é dividia.

Figura 5 – Particionamento de uma malha refinada 2D feito por uma curva de Hilbert.



Fonte – Aftosmis (2004)

5.2 Procedimento dos Testes

Durante a execução de um teste, cada processo MPI envolvido mapeia na memória o sistema linear completo a partir dos arquivos, usando a chamada de sistema `mmap()`. Esta técnica foi utilizada neste programa de teste para facilitar a implementação, porém ela não é viável durante uma simulação real, porque o sistema linear normalmente não é gravado em arquivo, mas sim gerado em partes por cada processo. Isso significa que, numa simulação, o sistema linear nunca fica centralizado em um único lugar de modo que possa ser facilmente acessado por todos os processos.

Tendo acesso ao sistema linear, o programa procede diferentemente para cada tipo de particionamento escolhido. Caso a execução seja para testar o particionamento original, este é simplesmente carregado de um arquivo e usado. Em todos os outros casos é feito um particionamento uniforme, que pode ser de dois modos: se a execução for para testar o próprio particionamento uniforme, é feito um particionamento completo, utilizando as configurações dadas para aquele caso (conforme detalhado na seção 5.3). Se a execução for para testar algum outro algoritmo, é feito um particionamento simples que considera um peso igual para todas as linhas da matriz.

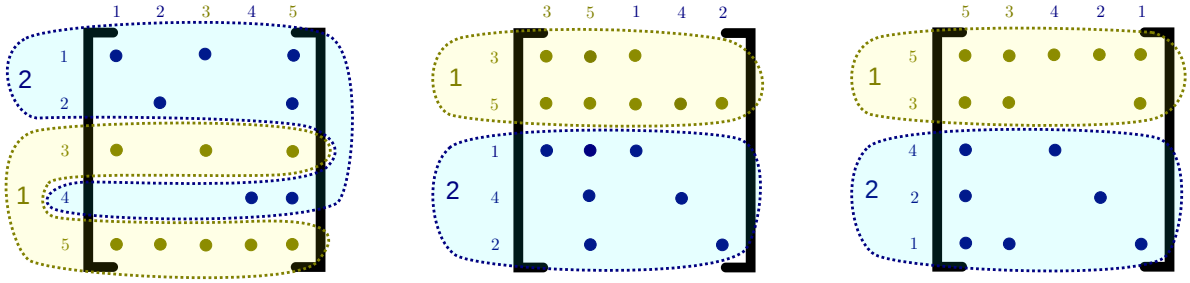
Esse particionamento simples é usado somente para distribuir a carga do sistema linear completo entre os processos envolvidos na execução, para que a partição final seja feita em paralelo. Exceto o algoritmo Cuthill-McKee reverso, que foi implementado dentro do programa, os outros particionamentos são feitos através do Zoltan, que além de seus algoritmos próprios, provê interface para acessar os algoritmos do ParMETIS e do

Figura 6 – Várias ordenações possíveis para uma mesma matriz.

(a) Partição sobre matriz original. As linhas dentro de uma partição não são contíguas.

(b) Reordenação da matriz original que agrupa as linhas por processo.

(c) Outra reordenação possível.



PT-Scotch.

Feita a partição final que será utilizada naquela execução, cada processo saberá quais elementos farão parte de seu subdomínio. Na matriz global do sistema linear, isso se traduz nas linhas da matriz que serão parte daquele processo. Note que, neste ponto, as linhas pertencentes a um processo não são necessariamente contíguas: nenhuma suposição é feita sobre a distribuição dos elementos escolhida pelo particionador. É então necessário reordenar a matriz para que as linhas pertencentes a cada processo fiquem contíguas. Como mostra a Figura 6, existam várias ordenações válidas.

Teoricamente, serviria qualquer ordenação trivial que mantivesse as linhas contíguas por processo. Entretanto, para a matriz final resolvida não ter qualquer influência do particionamento original feito pela HigTree, garantindo uma comparação justa somente pelo mérito de cada algoritmo testado, cada submatriz local do processo é ordenada com o RCM. Na visão final da matriz global, as linhas são dispostas na ordem dos identificadores de seus processos (primeiro as linhas do processo 0, depois as linhas do processo 1, e a assim por diante), sendo a ordem das linhas dentro de cada processo dada pela ordenação local. A Figura 7 mostra a forma de uma matriz gerada pela HigTree, usada como entrada do programa, e a Figura 8 mostra a nova forma global dessa matriz depois de reparticionada e reordenada.

Tendo a nova sequência das linhas, a matriz final é montada na memória. Para se mapear as posições originais das colunas da matriz para a nova ordenação, foi criada uma estrutura de dados de dicionário distribuído, capaz de encontrar rapidamente a qual nova coluna cada elemento não nulo deve pertencer, dado o índice da coluna antiga. Uma vez disposto na nova sequência, o sistema linear é resolvido através da interface de *solvers* da HigTree — descrita no capítulo 3 — até a tolerância do resíduo.

A solução dada pelo *solver* é então reorganizada para a ordenação original da matriz, de antes do particionamento, e norma do resíduo para esta solução é recalculada utilizando-se a matriz original. A diferença entre este resíduo e o resíduo dado pelo *solver*

Figura 7 – Distribuição dos elementos não nulos de uma matriz gerada pela HigTree no acoplamento pressão-velocidade do método da projeção no problema da cavidade, com uma malha de 1875 elementos e 2 níveis de refinamento.

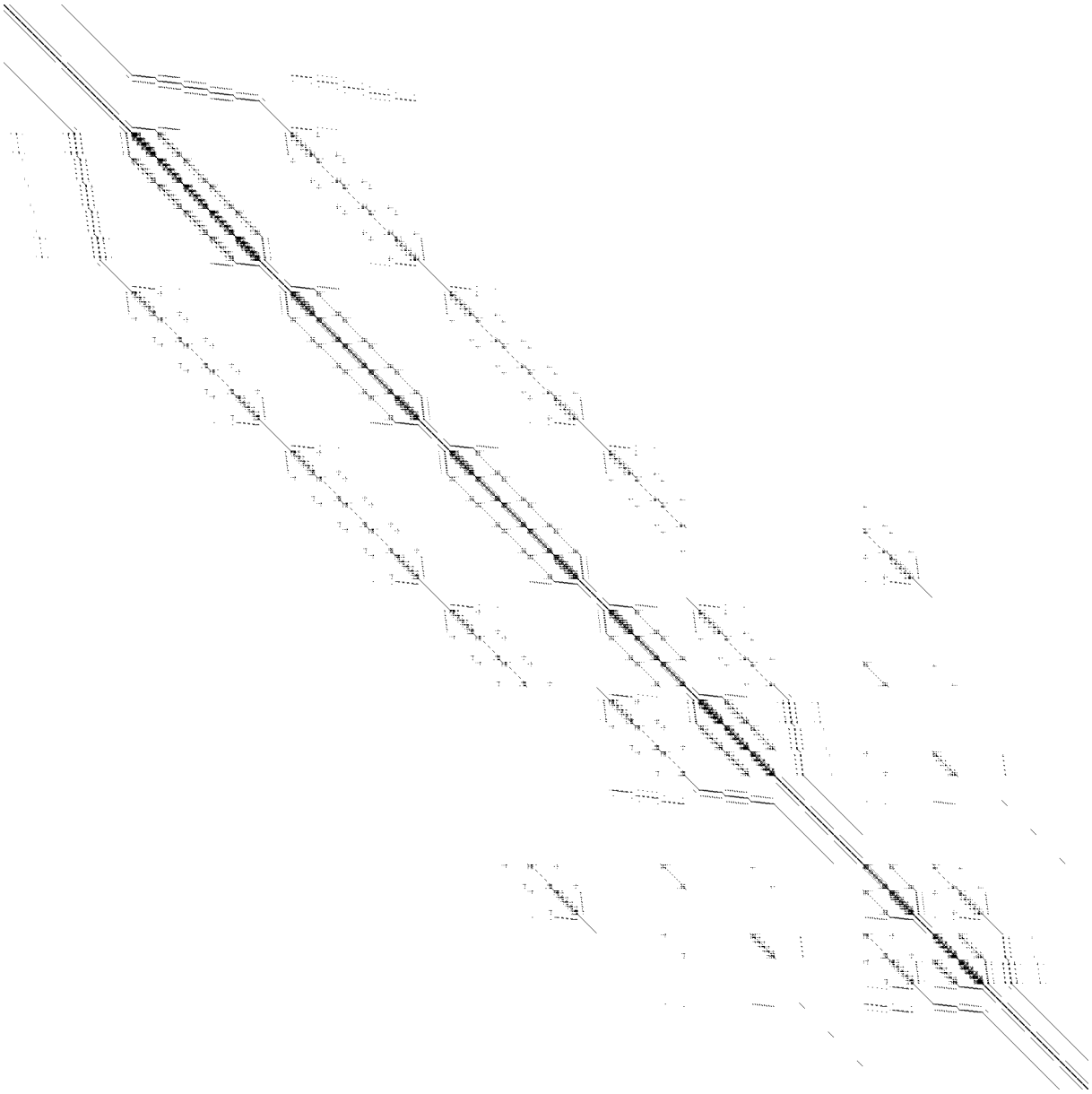
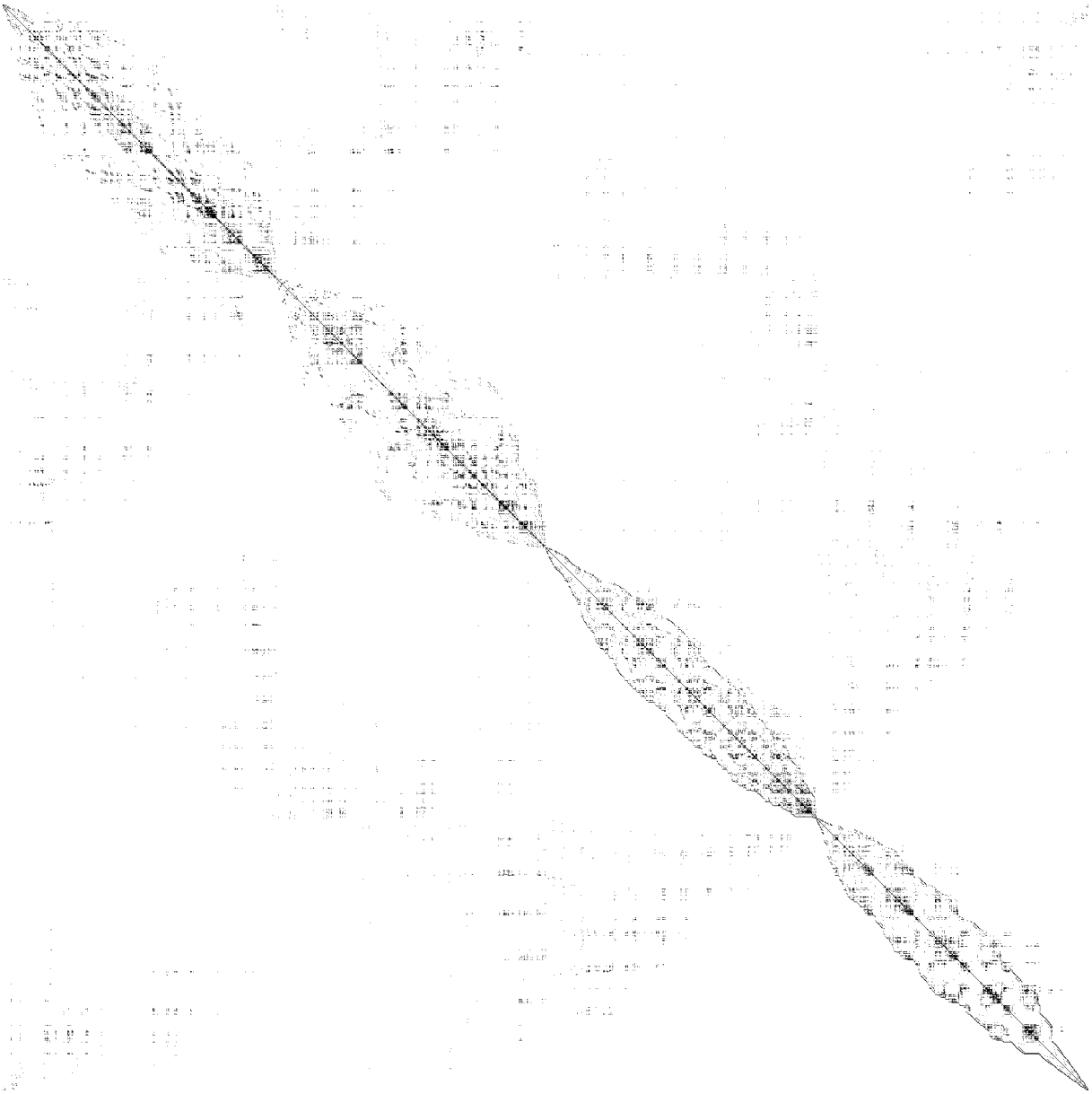


Figura 8 – Distribuição dos elementos não nulos da matriz da Figura 7, depois de reparticionada em 4 subdomínios pelo algoritmo de hipergrafo, e reordenada localmente em cada um dos 4 processos com o algoritmo de redução de banda RCM.



é verificada. Em todos os casos, a diferença verificada foi da ordem de zero de máquina, o que garante que não houve nenhum erro no processo de particionamento e reordenação da matriz, e que, portanto, os sistemas lineares original e particionado são equivalentes.

O processo de solução do sistema e verificação é repetido 20 vezes para cada caso. Isso é feito somente para amortizar variações aleatórias no tempo de execução, mas não representa múltiplos passos de tempo do problema de acoplamento pressão-velocidade original, pois o sistema linear utilizado é sempre o mesmo.

5.3 Configuração dos Casos

Para cada um dos algoritmos já descritos, com exceção do particionamento original, todas as combinações das seguintes variações de configuração foram executadas:

- Número de nós exclusivos do *cluster* utilizados (cada um com 20 núcleos de processador):
  - 3 nós;
  - 4 nós.
- Peso dos elementos não nulos fora da diagonal principal:
  - Zero;
  - Metade do peso da diagonal principal;
  - Peso da diagonal principal.
- Uma das seguintes configurações de *solver* e carga no nó:
  - Precondicionador *Additive Schwarz Method* (ASM) do PETSc com 20 processos por nó;
  - Precondicionador AMG do Hypre com 20 processos por nó;
  - Precondicionador AMG do Hypre com 4 processos por nó.

Já o particionamento original foi rodado em todas as combinações das seguintes configurações:

- Número de nós exclusivos do *cluster* utilizados (cada um com 20 núcleos de processador):
  - 3 nós;
  - 4 nós.

- Ordenação das submatrizes locais:
  - Ordenação original;
  - Cuthill-McKee reverso.
- Sempre com 20 processos por nó, usou uma das seguintes configurações de *solver*:
  - Precondicionador ASM do PETSC;
  - Precondicionador AMG do Hypre;

O particionamento original da HigTree não foi rodado na configuração de 4 processos por nó porque as partições gravadas em arquivo continham 80 e 60 subdomínios, (cabendo em, respectivamente, 4 e 3 nós do *cluster*, rodando com 20 processos cada), e não em 12 e 16 subdomínios, como seria necessário com 4 processos por nó. Já a configuração sem a ordenação das submatrizes locais foi feita para se determinar o impacto da ordenação no tempo de solução do sistema.

5.4 Resultados

A principal pergunta que se procurou responder neste estudo era: qual o impacto da qualidade do particionamento na solução do sistema linear? Em particular, se um particionamento melhor, com menor conectividade, faria uma diferença substancial no tempo de execução do *solver*. Os resultados mostraram que somente os algoritmos muito ruins conseguiram impactar conclusivamente no tempo de execução do *solver*, sendo o particionador geométrico do ParMETIS e o particionamento feito com o RCM particularmente ruins, como mostra a Figura 9. Note que nos casos rodados com o AMG, em vez do tempo total de solução do sistema linear, foi usado o tempo médio de uma iteração do *solver*. Foi feito assim por dois motivos:

1. O particionamento não pareceu impactar no número de iterações necessárias, ao contrário do que ocorre nos casos rodados com ASM. Comparando a distribuição do número de iterações da Figura 10a com a Figura 10b, nota-se que na primeira cada particionamento requer um número característico de iterações, enquanto que na segunda o número de interações varia em torno da mesma média. Portanto, se o tempo total fosse considerado no AMG, o impacto do particionamento seria contaminado pela variável aparentemente aleatória que foi número de iterações do *solver*;
2. O número médio de iterações foi muito pequeno quando comparado com os casos executados usando ASM (em média 36,5 vezes menor), e portanto o impacto de uma

Tabela 2 – Comparação da média do tempo de execução das várias configurações de peso usadas para os elementos não nulos da matriz.

	Tempo Médio (s)			Ganho sobre Peso 0	
Configuração	Peso 0	Peso $^1/_2$	Peso 1	Peso $^1/_2$	Peso 1
ASM-20	12,46	11,31	11,48	10,2%	8,6%
AMG-20	0,874	0,870	0,869	0,4%	0,6%
AMG-4	1,037	1,045	1,040	-0,8%	-0,3%
Média				3,3%	2,9%

variação aleatória de poucas iterações a mais ou a menos em alguma execução seria muito mais significativo se o número de iterações fosse levado em conta.

Olhando somente para os algoritmos bons na Figura 9, aqueles que compõem as aglomerações no canto inferior esquerdo de cada gráfico, notamos que alguns claramente se destacam em garantir menor conectividade. Em particular o PHG Grafo, que consistentemente gera as malhas menos conexas, seguido de perto do PHG Hipergrafo, que está muito próximo em conectividade, e também o PT-Scotch, que tem uma média de conectividade melhor que a dos outros particionadores bons.

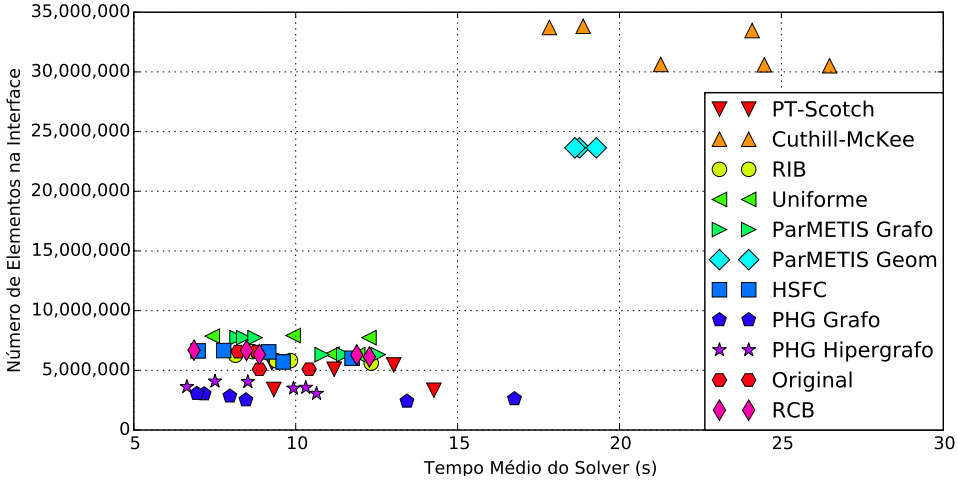
Entretanto, notamos que a consistência em garantir boa conectividade não se traduz, necessariamente, numa melhor performance, como podemos ver, por exemplo, na Figura 9b, onde as execuções do RCB têm conectividade entre 2 e 3 vezes maior que as execuções do PHG-G, porém é o primeiro é tão rápido quanto o segundo, nos melhores casos, e mais rápido, nos piores casos.

Somente no caso da Figura 9b podemos distinguir claramente um algoritmo que se sobressai aos outros, que é o PHG-H. Isso se dá possivelmente porque a variação do grande número de iterações necessárias pelo precondicionador ASM, na Figura 9a se sobressai à influência da custo de comunicação. Já na Figura 9c o pouco número de processos torna o custo da comunicação menos importante.

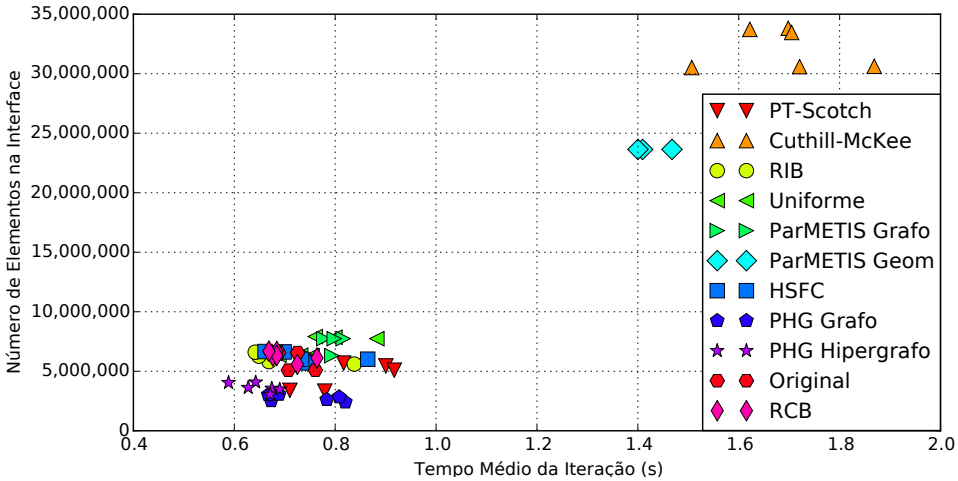
A Tabela 2 mostra que as vezes pode ser vantajoso considerar um peso de particionamento para cada elemento não nulo da matriz — resultando em um peso diferente para cada linha — em vez de simplesmente atribuir o mesmo peso à todas as linhas da matriz. A questão parece estar relacionada ao funcionamento interno de cada *solver*, provavelmente dependendo da razão entre o número de operações matriz-vetor (que são proporcionais ao número de elementos não nulos da matriz) e o número de operações vetor-vetor (que são proporcionais ao número de linhas da matriz, ou equivalentemente, ao número de elementos do vetor solução) presentes no algoritmo. Neste caso, a vantagem de pesar os elementos não nulos só se pronunciou no caso do método ASM, possivelmente pelo maior número de iterações exigidas no *solver*.

Figura 9 – A conectividade do particionamento pelo tempo de execução, para cada um dos precondicionadores testados.

(a) ASM com 20 processos por nó.



(b) AMG com 20 processos por nó.



(c) AMG com 4 processos por nó.

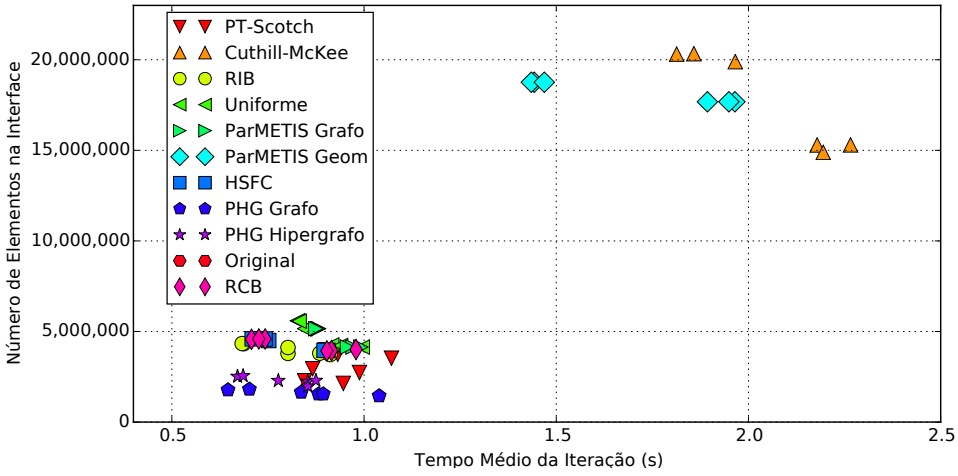
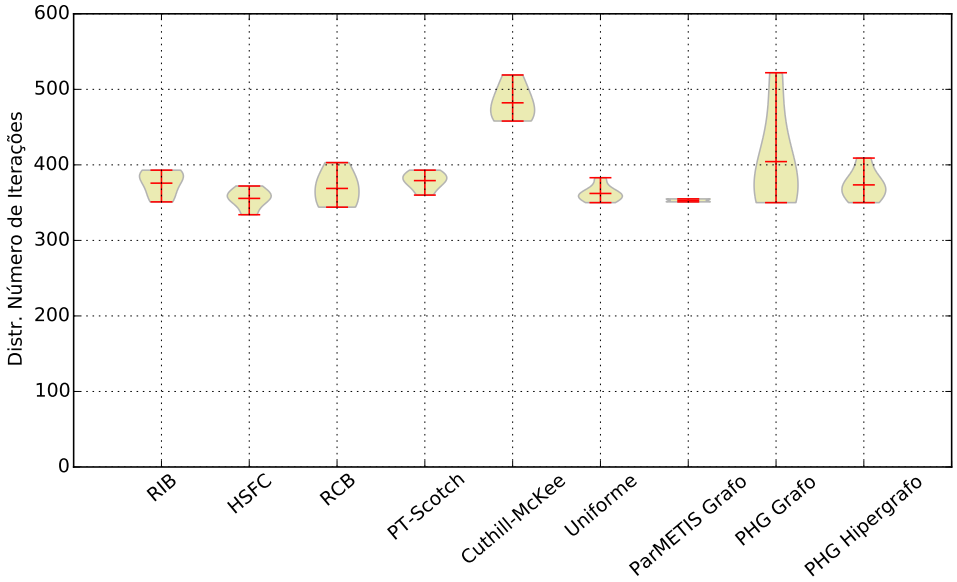
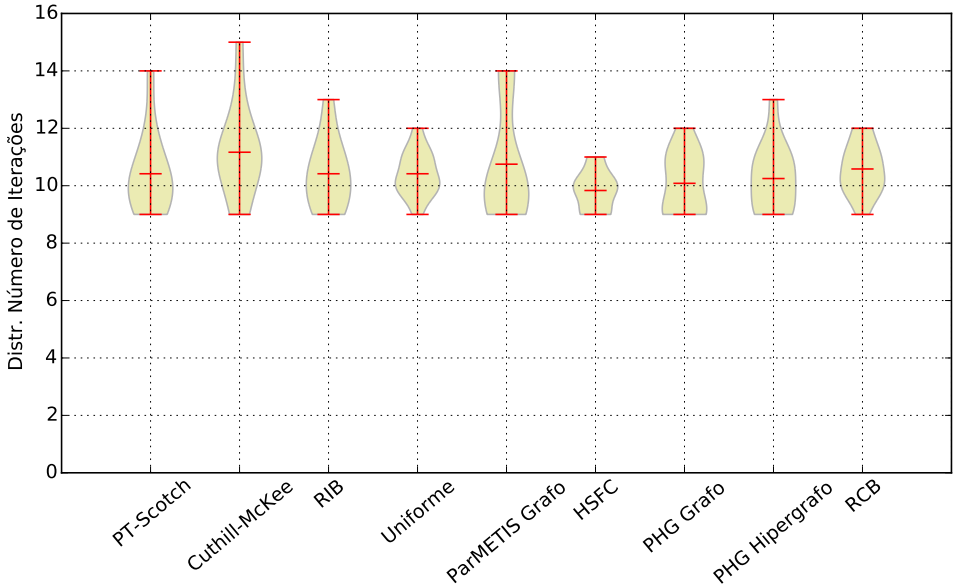


Figura 10 – Número de iterações médio, máximo e mínimo necessário para executar cada tipo de particionamento com cada precondicionador.

(a) Execuções com ASM.



(b) Execuções com AMG.



Nota – A espessura da área amarela indica a distribuição das amostras ao longo do eixo vertical.

Tabela 3 – Comparação da média do tempo das execuções em números diferentes de nós.

Configuração	3 Nós	4 Nós	Ganho	*Speedup*
ASM-20	13,34	9,66	38,1%	1,036
AMG-20	0,905	0,813	11,3%	0,835
AMG-4	1,157	0,954	21,3%	0,910

A variação no número de nós do *cluster* usados em cada caso teve o impacto esperado no tempo de execução, como mostrado na Tabela 3: o tempo de execução foi menor nas execuções com mais processos, e quanto pior o tempo de execução do *solver*, melhor (mais próximo de 1) foi o *speedup*.

O *speedup* aqui se refere à razão entre o inverso do tempo de execução e o número de nós utilizados, valor este normalizado pela execução com menor número de nós. No caso da Tabela 3, o *speedup* se refere à execução com 4 nós, normalizado pela execução com 3 nós, sendo, portanto, definido como $\frac{3T_3}{4T_4}$, onde T_3 o tempo de execução com 3 nós, e T_4 o tempo de execução com 4 nós.

Finalmente, mesmo que caso por caso as execuções não se sobressaiam consistentemente, podemos ver pela Tabela 4 que, na média, há uma forte tendência de melhor *performance* do algoritmo de hipergrafo — que obteve os melhores tempos. Entretanto, após estudar cada um dos algoritmos, ficou claro que somente um deles poderia ser usado sem alterações profundas na estrutura do código da HigTree: o RCB do Zoltan. Não por coincidência, este foi o mesmo algoritmo utilizado no MFSim (LIMA, 2012), e a HigTree, que embora não utilizasse o Zoltan, originalmente utilizava uma implementação própria do mesmo algoritmo.

A particularidade do algoritmo que torna o RCB usável é a sua garantia de que os subdomínios serão retangulares e alinhados aos eixos coordenados. Isso é necessário no MFSim porque, em sua malha estruturada, as células são organizadas em blocos retangulares uniformes, que devem ser encaixados nos subdomínios particionados.

Já na HigTree, o refinamento não se dá em blocos uniformes, mas sim em uma estrutura de árvore, que permite refinamento em regiões arbitrárias. Entretanto, cada árvore em si é retangular, e como cada subdomínio é formado por uma ou mais árvores, eles estão restritos ao único formato que as árvores pode adotar.

O ganho de 11% do PHG-H sobre o RCB na solução de um sistema linear não se traduz necessariamente em um ganho de 11% na execução em uma simulação de CFD, pois existem outras etapas do método numérico que também são computacionalmente caras — cujo impacto do particionamento não foi avaliado no presente estudo — e que influenciariam no tempo de execução total.

Mesmo que a quantificação do ganho em um cenário real não seja precisa, a

Tabela 4 – Comparação da média do tempo de execução dos particionamentos disponíveis (original e RCB) e o particionador com melhor resultado médio.

	Tempo Médio (s)			Ganho	
Configuração	Orignal	RCB	PHG-H (Melhor)	RCB	PHG-H (Melhor)
ASM-20	9,87	9,67	8,92	2,07%	10,7%
AMG-20	0,720	0,700	0,649	2,86%	11,0%
AMG-4	—	0,829	0,786	—	5,5%

Nota – O tempo é dado por iteração nos casos AMG, e total no caso ASM.

vantagem do particionamento em hipergrafo é distinta. Entretanto, devido ao grande tempo de desenvolvimento necessário para se adaptar a HigTree de modo a suportar o particionamento por PHG-H, consideramos esta tarefa fora do escopo do presente trabalho, e escolhemos utilizar o RCB para desenvolver o novo sistema de particionamento da HigTree. A adaptação da HigTree para suportar particionamentos em formatos arbitrários, o que possibilitaria a utilização do PHG-H, é deixada como perspectiva para trabalhos futuros.

6 Implementação de particionamento com múltiplas árvores

Como explicado na seção 1.1.5, cada árvore da HigTree está limitada a um formato retangular, portanto, para descrever domínios mais complexos, são usadas múltiplas árvores que, conectadas, formam o domínio como um todo. A Figura 11 mostra 10 arvores retangulares formando um domínio complexo. O particionador originalmente implementado na HigTree, só funciona em cima de árvores individuais, e portanto não é adequado para casos como este. Outra limitação do particionador original é que o particionamento é serial, feito em um único processo, e portanto está limitado a um domínio que caiba inteiro na memória de um único computador.

A maneira ideal de se realizar esse particionamento é considerando o domínio como um todo, observando a relação de vizinhança entre todas as células, mesmo entre árvores diferentes. De preferência, o particionamento deveria ser realizado em paralelo: sem necessitar carregar a malha completa em um único nó de processamento.

Neste capitulo descrevo a implementação da solução proposta para se resolver este problema, utilizando o algoritmo RCB do Zoltan. Conforme explicado na seção 5.4, este foi o único dos algoritmos estudados que poderia ser usado na estrutura de árvores atualmente suportada pela HigTree.

6.1 A distribuição da carga

No mecanismo de particionamento implementado, cada uma das árvores que será utilizada na simulação pode ser dada por qualquer um dos processos MPI. Uma estratégia simples que para dividir a tarefa de carregamento e/ou criação das árvores entre os processos é o *round-robin*: as árvores a serem carregadas são distribuídas uma a uma,

Figura 11 – Domínio 2D complexo formado por múltiplas árvores retangulares conectadas.

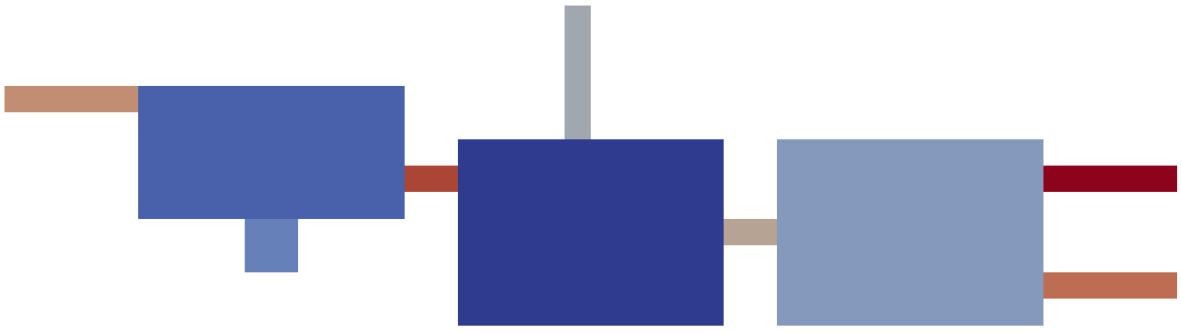
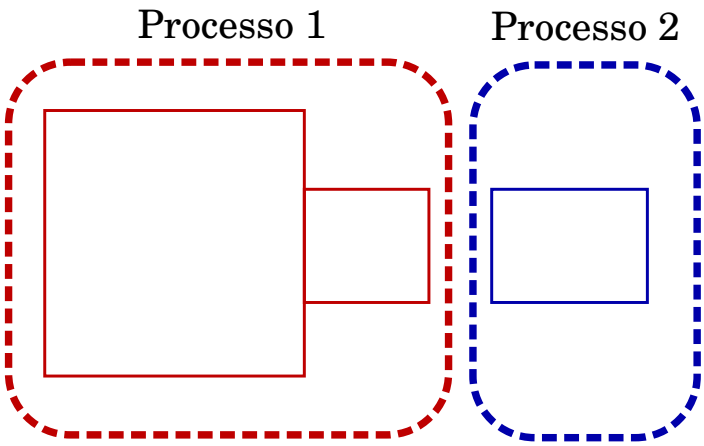


Figura 12 – Para iniciar o particionamento, cada processo contribui com qualquer quantidade de árvores.



para cada processo, em ordem circular, garantindo uma diferença de no máximo uma árvore por processo (essa é a estratégia usada no programa de exemplo criado, descrito na seção 6.4). Entretanto, essa não é uma restrição imposta: as árvores podem ser dadas ao particionamento por qualquer processo, em qualquer quantidade. Na Figura 12, duas árvores são dadas pelo processo 1, e uma árvore pelo 2.

O menor elemento que será dividido entre os processos é um nó do primeiro nível da árvore, logo abaixo da raiz, e portanto o domínio é dividido no nível de malha mais grosso de cada árvore. O peso de cada elemento é calculado como sendo o número de nós folha partindo daquela célula, portanto, se a célula não for refinada, ela própria é a folha, e seu peso é 1. O peso calculado mais a posição do centro de cada elemento são dados ao Zoltan, que realiza a partição dos elementos em utilizando o RCB.

Cada elemento é identificado por dois inteiros: seu número de sequência dentro da árvore, seguido do identificador global único dado à cada árvore no início do particionamento. Em cada processo, o Zoltan retorna uma lista contendo o destino final de cada um dos seus elementos: a qual processo deverá pertencer cada elemento de suas árvores locais. Um exemplo de particionamento dado pelo Zoltan é mostrado na Figure 13, onde uma árvore inteira e fração de outra deverão ser enviadas pelo processo 1 ao processo 2.

Como o algoritmo usado garante que os subdomínios serão retangulares, os elementos locais são agrupados em caixas retangulares — uma por árvore local por processo remoto — e enviados em lotes aos seus processos de destino, como mostrado na Figura 14. Os elementos enviados são removidos das árvores locais, que são então ajustadas para um tamanho reduzido, ou são apagadas, caso não reste nenhum elemento daquela árvore no processo.

Para cada lote retangular de elementos recebidos é criada uma nova árvore no processo remoto. Isso gera um problema em potencial que é a fragmentação do domínio,

Figura 13 – Cada processo recebe do Zoltan quais partes de suas árvores deverão ser enviadas aos processos vizinhos.

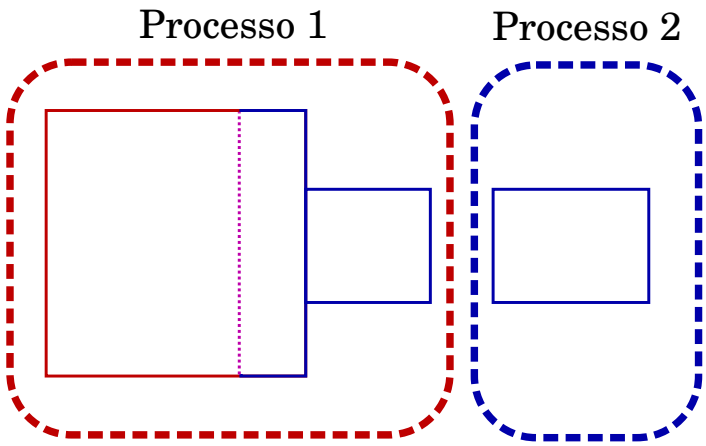
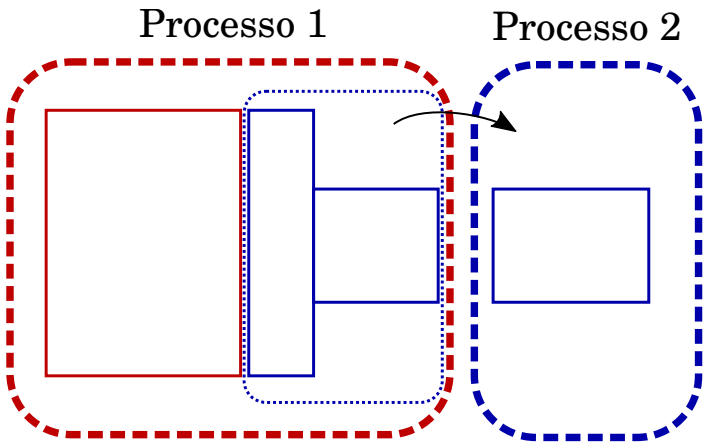


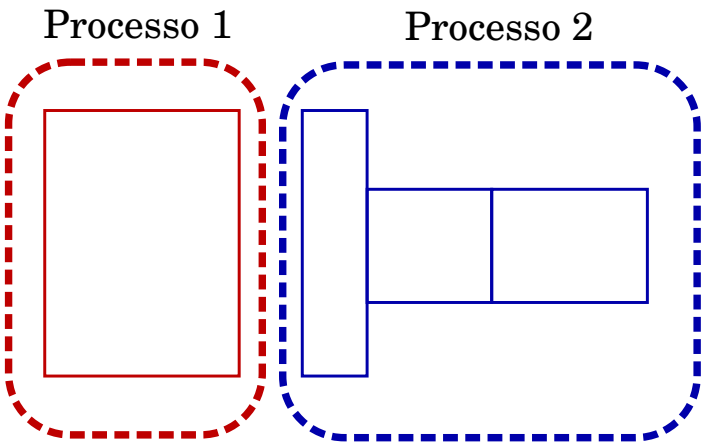
Figura 14 – Os elementos pertencentes a processos remotos são agrupados em áreas retangulares e enviados.



no caso de haver alteração da malha do domínio e reparticionamento, para balanceamento da carga entre processos. No caso da Figura 15, o processo 2 utiliza agora três árvores para representar seu subdomínio, sendo que somente duas bastariam. Isto não é um problema atualmente, pois a HigTree ainda não suporta alteração dinâmica do domínio, então esta fragmentação ocorre no máximo uma vez. Porém, refinamento adaptativo é uma funcionalidade planejada, e portanto o problema é relevante, que poderia causar uma fragmentação cada vez maior do domínio.

Para entendê-lo, considere uma árvore que foi dividida em duas pela interface entre dois processos, isso acontece porque uma árvore sempre está totalmente contida em um processo. Eventualmente, por necessidade da simulação, a malha desse domínio é refinada, e o equilíbrio do número de processos entre os elementos é perturbado. Para restabelecer o equilíbrio, a malha precisa ser reparticionada. No novo particionamento, a interface entre processos é movida, dividindo novamente a árvore entre dois processos.

Figura 15 – Cada região recebida se torna uma nova árvore, o que deixa o subdomínio do processo 2 subdividido em mais partes do que o necessário.



Temos que, o que eram somente duas árvores ao final do primeiro particionamento, tornaram-se agora três. À medida que refinamentos forem sendo feitos e desfeitos — e que o balanceamento for mudando ao longo da simulação — as árvores tornar-se-ão cada vez mais fragmentadas pelas novas interfaces entre processos.

Para resolver esse problema, um algoritmo de aglutinação de árvores foi elaborado. Todas as árvores recebidas e remanescentes no processo após o particionamento passam por uma etapa de unificação. Idealmente, árvores vizinhas deveriam ser unificadas de modo a restar o menor número de árvores possível para formar a geometria daquele subdomínio. Na prática, não parece ser possível criar um algoritmo que ofereça tal garantia e ainda seja rápido o suficiente para ser viável. Uma solução ingênua — e claramente inviável por seu custo superpolinomial — envolveria testar todas a possibilidades de aglutinação para determinar qual geraria o menor número de árvores possível.

Em vez de tentar buscar a solução ideal, o algoritmo implementado tenta simplesmente unificar as árvores duas a duas em ordem arbitrária, até que não seja mais possível unir quaisquer duas árvores do conjunto restante — isto é, tal que o resultado da união não seja retangular. Esse procedimento não garante que ao final ter-se-á o menor número de árvores possível, mas é suficiente para garantir que o número de árvores não aumentará indefinidamente à medida que reparticionamentos forem feitos.

No exemplo das figuras, na Figura 14 a partição fragmentou uma das árvores do processo 1, aumentando o número total de árvores de três para quatro, como mostrado na Figura 15. Porém, uma das árvores transferidas para o processo 2 pôde ser unificada com sua vizinha lá presente, restaurando a quantidade original de três árvores. A Figura 16 mostra o domínio final, após a unificação.

Para que duas árvores possam ser unidas, é necessário que a geometria resultante seja retangular e que suas subdivisões internas sejam compatíveis. Mas isso não é suficiente:

Figura 16 – Duas das árvores do processo 2 foram unificadas, resultando no menor número de árvores possível para este particionamento.

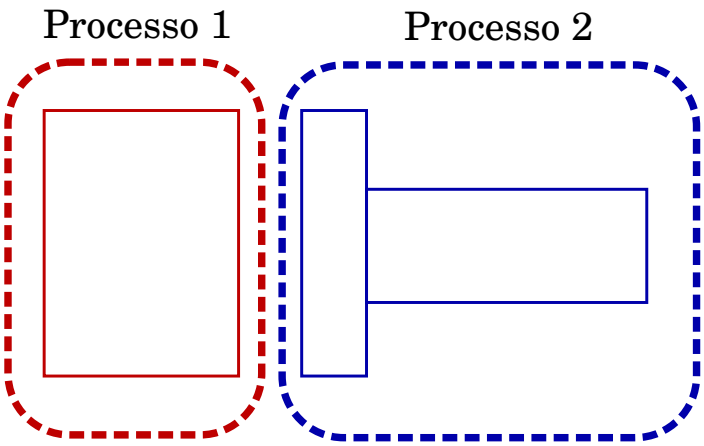
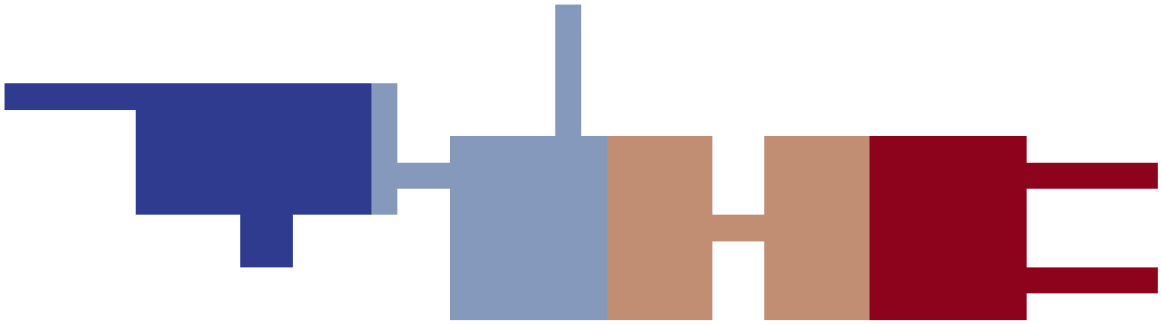


Figura 17 – Domínio 2D complexo particionado em 4 processos.



também é necessário que as árvores cumpram a mesma função física e numérica dentro da simulação. Essa informação é dada pelo usuário do particionamento na forma de grupos: cada árvore pertence a um grupo, e todas as árvores do grupo cumprem a mesma função numérica e possuem as mesmas variáveis no domínio da simulação.

Somente árvores pertencentes ao mesmo grupo podem ser aglutinadas, e portanto, árvores do domínio interno devem estar em grupos diferentes das árvores que determinam condições de contorno. Mesmo entre as árvores da condição de contorno, mais de um grupo pode ser necessário, por exemplo, no case de árvores do contorno com condição de Dirichlet que não devem ser unificadas com árvores com condição de Neumann.

Todo esse desenvolvimento já é suficiente para carregar um domínio construído a partir de múltiplas malhas, e particioná-lo em paralelo de modo que os subdomínios sejam bem equilibrados e conectados. A Figura 17 mostra o resultado do particionamento em 4 processos da malha da Figura 11 feito pela nova implementação.

6.2 A Construção do *Fringe*

Além do particionamento e distribuição de carga em si, para dar suporte a múltiplas árvores em paralelo foi necessário modificar também o tratamento do *fringe*. O *fringe* é uma borda que se estende em um número configurável de células para dentro do domínio controlado pelo processo vizinho, e mantém uma cópia das suas variáveis no processo local. Esse *fringe* é usado em toda parte da simulação que necessita dos valores das variáveis próximos às interfaces entre processos.

A solução original para o *fringe* não previa sua extensão para dentro de árvores originalmente distintas. A sua construção se limitava a, se uma árvore original fosse dividia pelo particionamento, o corte era feito m células antes do limite do subdomínio (onde m é o número de células do *fringe*), e essas células extras eram marcadas como sendo parte do *fringe* numa estrutura de dados chamada de `partition_table`.

Na nova solução, além de ter que considerar o *fringe* para as árvores que foram divididas pelo particionamento, também é necessário considerar o *fringe* entre árvores que já eram originalmente independentes dentro de um mesmo processo, mas foram separadas em processos diferentes. Além disso, existem casos onde o *fringe* atravessa mais de uma árvore até chegar no seu tamanho requisitado. Por isso, a construção do *fringe* foi feita como um procedimento completamente independente do particionamento, que trata indistintamente todos os casos.

O procedimento é feito em duas etapas: a primeira é a identificação dos vizinhos imediatos, onde é determinado as áreas de contato dos subdomínios entre cada dois processos. Se existir alguma área de contato entre os subdomínios de dois processos, eles são considerados vizinhos imediatos. A segunda etapa é uma busca em largura em paralelo partindo das áreas de contato, feita para determinar quais elementos de uma árvore de domínio serão replicados como *fringe* em outro processo. Se for construído um *fringe* ligando um processo ao outro, eles são considerados vizinhos.

Existe a possibilidade de dois processos serem vizinhos, mas não imediatos. Neste caso, eles não compartilham uma fronteira de subdomínios, porém estão a menos de m células base de distância um do outro, de modo que o *fringe* de um faz referência ao outro.

Para identificar os potenciais vizinhos imediatos, é feita uma filtragem inicial que exclui processos que definitivamente não podem ser vizinhos. Essa filtragem é a única etapa da construção do *fringe* onde o custo computacional é função do número global de processos — ao contrário das etapas seguintes onde o custo depende somente da vizinhança do processo. Portanto, essa filtragem inicial é o gargalo de escalabilidade do procedimento, por isso foi mantida o mais simples possível.

É feito um *broadcast* de todos para todos, onde cada processo informando uma *bounding box* com os seus limites. Então cada processo testa os limites de sua própria

bounding box com os limites recebidos, identificando assim quais processos são espacialmente vizinhos, e portanto há a possibilidade de compartilharem uma interface entre domínios.

Cada processo compara os limite de cada uma das suas árvores com os limites da *bounding box* do seu processo, identificando as regiões de interseção. Cada uma dessas interseções é então comparada com os limites dos potenciais vizinhos imediatos, gerando assim, uma lista de interfaces existentes entre árvores locais e cada vizinho imediato. Cada lista gerada é enviado ao processo vizinho correspondente, que ao receber, compara quais interfaces recebidas também intersectam suas interfaces enviadas para o processo de origem. As interseções entre as áreas enviadas e recebidas são as áreas em que uma árvore do processo local tocam uma árvore do processo vizinho.

É responsabilidade de cada processo identificar quais células locais fazem parte do *fringe* de cada um dos seus vizinhos. Para cada processo vizinho, é criada uma estrutura que guarda o estado da busca já realizada, onde são marcadas as células locais pertencem ao *fringe* daquele vizinho. Essa estrutura é contém, para cada árvore do subdomínio local, uma simples *array* multidimensional de inteiros, de dimensão igual ao número de filhos da árvore, onde é armazenado a proximidade de cada célula até o ponto de contato com o vizinho.

As áreas de contato entre subdomínios servem como sementes da busca. Inicialmente são marcadas todas as células da árvore que fazem interseção com a área de contato daquele vizinho, a quem são atribuídas o valor m. O algoritmo de busca em largura parte daí, decrementando o valor em um para cada nova célula adicionada na busca. A busca prossegue até chegar ao valor 1 ou não ser possível marcar mais células. Todas as células com valor maior do que 0 localizadas na fronteira da árvore disparam uma nova busca nas árvores vizinhas que estão em contato (exceto se esta for a origem da busca), estejam elas no mesmo processo ou em outro.

Esse repasse do trabalho de busca para outros processos permite que sejam identificados células do *fringe* em processos que não sejam vizinhos imediatos. Uma vez identificados esses vizinhos indiretos, não é necessário fazer nenhum tratamento especial: contanto que um processo possua *fringe* referente a outro processo, eles são considerados vizinhos, não importando se existe uma interface direta entre os dois.

Como não se sabe de início quantas tarefas de busca em cada árvore deverão ser disparadas através da rede, é relativamente difícil de se determinar se o processamento está concluído ou não; afinal, qualquer processo que está ocioso pode receber uma semente para continuar a busca do *fringe* pertencente a um vizinho indireto. Para resolver esse problema foi implementado um algoritmo distribuído de detecção de terminação chamado *weight throwing* (MATTERN, 1989), que é capaz de identificar quando não há mais nenhuma mensagem pendente no sistema, e todos os processos estão ociosos. Ao final da busca, cada processo saberá quais células locais deverão fazer parte do *fringe* em algum processo

vizinho.

Finalmente, são construídas as árvores do *fringe* que o processo enviará para seus vizinhos. Todas as células marcadas com valor maior que 0 farão parte do *fringe* do vizinho que iniciou a busca. Para cada árvore marcada, para cada vizinho, é feita outra busca em largura, dessa vez mais simples que a primeira busca, pois não se propaga para outras árvores. Essa busca delimita as regiões retangulares, internas à árvore local, que serão as árvores do *fringe* construídas para cada vizinhos.

Cada árvore é construída e enviada ao vizinho correspondente. Nesta etapa, o algoritmo de detecção de terminação é usado novamente, pois o processo destinatário de cada árvore ainda não tem a informação de quem são seus vizinhos indiretos.

Ao contrário da solução de particionamento original, o *fringe* é formado por árvores diferentes das do domínio interno. Cada processo agora considera as árvores que fazem parte do *fringe* em separado das árvores que compõem o domínio interno. Isso é diferente da estrutura adotada anteriormente, onde o *fringe* era uma extensão da árvore do domínio, delimitado por informações contida na estrutura `partition_table`.

6.3 Adaptação do código para usar `partition_graph`

Além do particionamento em si, foi necessário adaptar como as outras partes do código enxergam e interagem com o domínio distribuído. Essa interação era mediada pela estrutura `partition_table`, que continha informações globais de todas as árvores de todos os processos, além de quais partes dessas árvores pertenciam ao *fringe*. Uma cópia idêntica do mesmo `partition_table` tinha que ser replicada em cada um dos processos, o que pode ser considerada uma limitação, já que por questões de escalabilidade, é indesejável que todos os processos tenham uma visão global do domínio. Idealmente, a complexidade e o uso de memória de cada processo deve se limitar à visão local de seu subdomínio e vizinhança, e ser independente do tamanho total da simulação e número de processos.

Outra limitação do `partition_table` é que este não guardava a conectividade entre árvores originalmente distintas. Em vez disso, a conectividade entre subdomínios era baseada na suposição de que o domínio era formado por uma única árvore original, que quando particionada, cada um dos processos cuidaria de uma suas fatias. Essa limitação da estrutura estava atrelada à limitação do particionador original, que particionava cada árvore individualmente. Como a motivação original deste novo particionar era a necessidade de se trabalhar com domínios complexos, compostos globalmente por múltiplas árvores conectadas, foi necessário remover essa limitação imposta pela estrutura antiga.

Para resolver estas limitações, foi criada uma nova estrutura de dados usada para armazenar os detalhes relativos ao particionamento, e suprir a função antes exercida pelo

`partition_table`. A nova estrutura foi batizada de `partition_graph`, e ao contrário do `partition_table`, é uma estrutura local e escalável, pois não contém informações sobre todos os processos. Em vez disso, armazena somente a conectividade do processo local com os seus vizinhos identificados na busca do *fringe*. A principal estrutura dentro da `patition_graph` é uma tabela *hash*, indexada pelo *rank* MPI, que para cada processo vizinho, guarda:

- a lista das árvores do *fringe* local que referenciam células naquele processo vizinho;
- quais células das árvores locais são referenciadas por árvores do *fringe* daquele processo vizinho.

Os processos que não são vizinhos simplesmente não estão representados nessa tabela *hash*, o que torna a estrutura independente do número total de processos, e portanto, escalável. As informações aí contidas são suficientes para que cada processo saiba exatamente o que enviar e receber de cada um dos seus vizinhos, seja para sincronização dos identificadores globais, seja nas sincronização das propriedades físicas.

Na API original da HigTree, o particionamento era uma operação independente da criação do domínio da simulação: o usuário chamava o particionador, e depois, utilizando as árvores resultantes em cada processo, criava as estruturas da simulação local `sim_domain` e global `psim_domain`. Entretanto, embora a relação não fosse explícita na API, internamente o código parecia supor que todas as árvores geradas pelo particionamento — e somente elas — haviam sido adicionada ao `sim_domain`. Com o novo particionamento, procurei manter essa independência entre particionamento e a criação do domínio, mas com o cuidado de não fazer suposições sobre quais árvores particionadas seriam realmente usadas na simulação.

O domínio global da simulação, `psim_domain`, que antes continha a estrutura `partition_table` como propriedade interna, agora tem a estrutura `partition_graph` no lugar, mas em vez de usar toda conectividade entre vizinhos descrita na nova estrutura, o grafo é filtrado para conter somente as árvores também contidas no domínio da simulação local `sim_domain`. Assim, a conectividade entre vizinho só existe nas árvores que tanto foram criadas pelo processo de particionamento quanto foram adicionadas ao domínio local. Certamente o caso de uso mais natural é o domínio ser composto exatamente pelas mesmas árvores do particionamento, mas esse processo de filtragem dissocia as duas etapas tornando o código mais flexível: o usuário pode colocar árvores que não serão usadas no domínio dentro do processo de particionamento, e o usuário pode colocar árvores criadas fora do particionamento junto com as árvores do particionamento no seu domínio de simulação.

Os domínios de simulação globais, tanto o de centro quanto o de face (`psim_domain`

e `psim_facet_domain`, respectivamente) utilizam a informação da conectividade entre os vizinhos para sincronizar os identificadores globais das células e das faces, que devem ser únicos dentro do domínio global. O algoritmo de sincronização original, que baseado no `partition_table` utilizava comunicação síncrona, e era linear no número total de processos. Já o novo algoritmo implementado baseado no `partition_graph` utiliza transmissão assíncrona de dados, e é linear no número de vizinhos para cada processo. Isso não garante que o novo algoritmo será sempre mais rápido que o original, mas torna-o escalável, pois a operação agora independe do número total de processos.

Além das estruturas globais, a própria estrutura local `sim_domain`, que lida somente com o processo local, e é agnóstica à paralelização (paralelização é responsabilidade do `psim_domain`) também foi alterada. Algumas funções da HigTree com acesso ao `sim_domain` necessitam iterar apenas pelas células internas ao subdomínio (excluindo-se o *fringe*). Originalmente o `sim_domain` armazenava os limites geométricos do domínio interno, e utilizava esse limites para filtrar as células do *fringe* da iteração. Para se adequar melhor ao novo paradigma, onde as árvores do *fringe* são independentes das árvores internas, o mecanismo de filtro foi alterado: ao percorrer somente as células internas ao subdomínio, em vez de filtrar pela geometria, as árvores do *fringe* são simplesmente deixadas de fora da iteração.

A diferenciação entre as árvores locais e as árvores do *fringe* é feita pela posição em que elas são armazenadas pelo `sim_domain`, onde as árvores locais são as primeiras da lista interna do `sim_domain`, e a partir de um certo índice conhecido, estão as árvores que compõem o *fringe*.

A última etapa da adaptação do código para o uso do `partition_graph` foi na refatoração de todos os exemplos e testes contidos no pacote da HigTree, adaptando-os para a nova API que fora desenvolvida para o particionamento e criação de domínio. A mudança da antiga API para a nova inclui, dentre outras coisas, a substituição da estrutura `partition_table` para `partition_graph`. Ao final, todos os exemplos compilam com sucesso, e todos os testes incluídos originalmente na HigTree executam corretamente, onde são incluídos vários casos em paralelo, que passaram a utilizar o novo sistema de particionamento.

6.4 Execução de Caso com Domínio Composto

Dos dois problemas elencados inicialmente sobre o particionador original da HigTree — eram eles: a impossibilidade de se utilizar domínios compostos por múltiplas árvores e o procedimento de particionamento em si não ser realizado em paralelo — a pior e mais crítica era a primeira, que limitava quais casos podiam ser executados em paralelo. O principal caso de exemplo de domínios compostos contido no repositório da HigTree era

executado em *serial* por causa desta limitação.

Um caso com este mesmo domínio foi executado tanto em série quanto em paralelo — utilizando o novo particionamento — e o ganho da execução em paralelo foi quantificado. O domínio é o mesmo mostrado nas Figuras 11 e 17, em 2D, com duas saídas de massa com condição de Neumann no lado direito, e duas entradas, à esquerdo e acima.

Usando um programa exemplo da HigTree, o caso foi executado com a configuração original dada pelo *script* `ct-re100-01.sh`. Qualitativamente não havia nenhuma diferença entre a execução *serial* e a paralela. Para se obter um ponto de comparação quantitativo, o caso foi alterado para executar com um número de Reynolds maior, de modo que, sem modelo de turbulência e usando diferenças centradas como esquema advectivo, a solução numérica divergiria naquela malha. O tempo físico da divergência deveria ser idêntico em ambas as execuções para garantir equivalência numérica entre a execução *serial* e paralela. As propriedades adimensionais utilizadas na simulação foram:

Reynolds : $\frac{UL}{\nu} = 1000$

Largura das entradas e saídas : $1L$

Velocidade média nas entradas : $1U$

Tempo físico simulado : aprox. $6\frac{L}{U}$ (até a divergência)

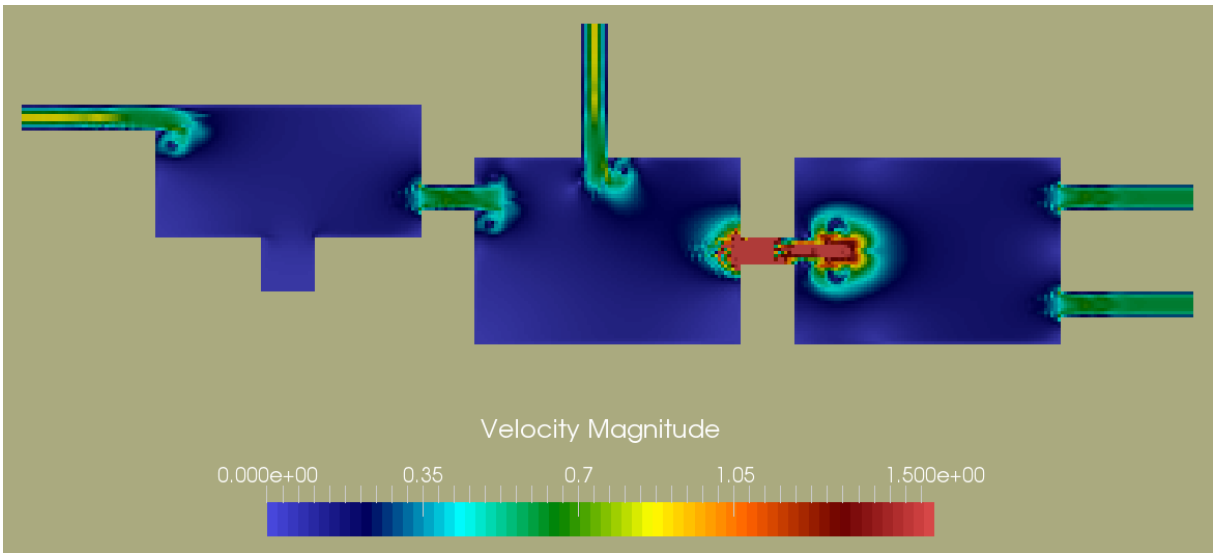
Passo de tempo : $0,01\frac{L}{U}$

onde U é a velocidade característica do escoamento, L é seu comprimento característico e ν é a viscosidade cinemática do fluido.

O caso foi executado com 1 e 4 processos por 690 passos no tempo, que é um pouco além de divergirem completamente com 670 iterações. Por ser um caso configurado para divergir, a representatividade física da solução numérica não é importante, somente o tempo gasto em cada execução e a concordância entre os dois resultados: da execução em paralelo — que utiliza o novo particionado — com o resultado da execução em *serial* — onde não ocorre particionamento.

Quantitativamente, ambos os casos divergiram exatamente no mesmo passo de tempo, o que demonstra a equivalência numérica entre eles. Qualitativamente as visualizações são idênticas, inclusive após o início da divergência, que se desenrola por dezenas de passos de tempo até a divergência completa no tempo $6,7\frac{L}{U}$. A Figura 18 mostra o campo de velocidade no tempo $6,6\frac{L}{U}$, último quadro salvo onde o campo de velocidade ainda tem valores finitos, quando ainda é possível distinguir a influência do escoamento antes do método numérico divergir completamente. É possível notar a expansão da região onde o campo vetorial não faz sentido físico a partir do estrangulamento.

Figura 18 – Divergência no teste comparativo entre execução paralela e *serial* em domínio complexo.



Os tempos de execução em uma estação de trabalho com 4 núcleos de processamento são mostrados na Tabela 5. Note que o *speedup* em paralelo, normalizado pela execução serial, foi maior que o ideal de 1. Isso porque o *solver* linear convergiu com menos iterações na execução em paralelo do que na execução serial. A razão para a melhor convergência é desconhecida, mas provavelmente foi influenciada pela enumeração dos elementos, que é diferente nas duas execuções, já que ela depende do particionamento. A enumeração, por sua vez, define diretamente a ordenação das incógnitas no sistema linear, afetando a sua convergência (nesse caso particular, por sorte, para melhor). Outro fator importante — que por si só não explica o *speedup* acima de 1, mas é muito relevante no tempo de uma execução paralela — é que os custos de comunicação foram desprezíveis nesse caso, já que a comunicação entre os processos foi interna a um único computador.

Tabela 5 – Tempo e *speedup* do caso 2D de domínio complexo.

Processos	Tempo	*Speedup*
1	10 min 33,198 s	1,0
4	2 min 28,165 s	1,1

7 Distribuição dos campos de propriedades

A HigTree oferece suporte à criação de aproximações discretas para campos escalares contínuos. Um campo escalar é representado pela estrutura de dados `distributed_property`. Seu nome faz referência ao fato de que o campo estará automaticamente associado às árvores que compõem o domínio de simulação distribuído. Os valores deste campo de propriedades distribuída podem ser armazenados ou no centro das células ou no centro das faces perpendiculares a um dos eixos de coordenadas.

Especificamente, na solução de Navier-Stokes em malha deslocada 2D, os valores do campo de pressão são localizados no centro das células, os valores da componente X do campo de velocidade são armazenados nas faces perpendiculares ao eixo X e, similarmente, os valores da componente Y da velocidade são armazenadas nas faces perpendiculares ao eixo Y. Cada um desses campos de propriedades físicas, inclusive outros campos auxiliares — como o campo de velocidade estimada, usado no método da projeção — é representado por uma instância da estrutura `distributed_property`.

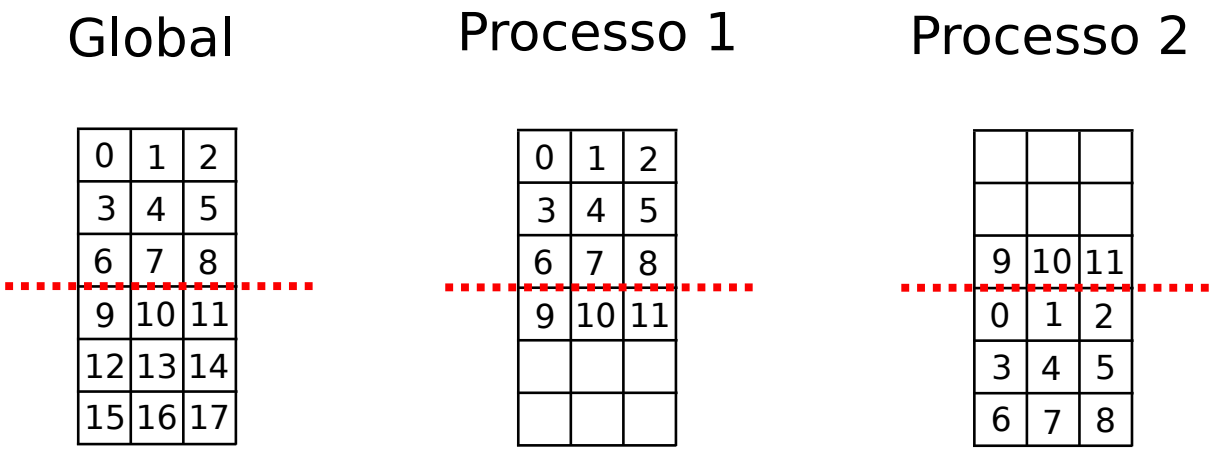
Originalmente, a HigTree armazenava uma cópia completa de todas os valores de uma propriedade em cada um dos processos MPI. Essa abordagem simplificava tanto a implementação da sincronização dos valores das propriedades entre processos, que poderia ser realizada através de uma operação de *broadcast*, através da chamada `MPI_Allgather()`, e também simplificava o processo de reinício da simulação, uma vez que era suficiente que um único processo escrevesse ou carregasse todos os valores de um arquivo, sendo este procedimento serial.

A desvantagem óbvia é o consumo de memória na execução. Seria razoável esperar que somente um valor de propriedade fosse armazenado para cada célula discreta do domínio, e portanto o consumo de memória das propriedades seria proporcional a n, onde n é o número total de células discretas do domínio. Em vez disso, o consumo era proporcional a $n \times P$, onde P é o número de processos MPI, pois cada valor era armazenado P vezes.

Uma desvantagem menos explícita é que o procedimento de sincronização utilizado não escala linearmente. A operação de *broadcast* tem custo computacional de $O(n \log P)$. Isso significa que, mesmo aumentando P e n na mesma proporção, de modo a manter $\frac{n}{P}$ constante, o custo da sincronização será necessariamente maior em um problema maior.

Implementei um mecanismo de distribuição das propriedades físicas, onde cada processo armazena somente os valores localizados nas árvores que compõem seu subdomínio local (incluindo o *fringe*). Para suportar a nova configuração, criei uma enumeração local dos elementos, separada da enumeração global, até então utilizada para associar cada

Figura 19 – Enumerações global e locais de células de um domínio distribuído.



elemento do domínio (célula ou faceta) a um valor de propriedade.

Tendo-se um elemento referenciado por um `sim_domain` ou `sim_facet_domain`, é possível recuperar seu identificador (*id*), que por sua vez pode ser usado para indexar um valor dentro da `distributed_property`, onde os valores são armazenados em um *array* dinâmico.

Passou-se a utilizar uma enumeração somente dos elementos locais do processo para identificar os valores das propriedades. Isso foi necessário pois uma enumeração local não segue necessariamente a enumeração global, pois os elementos do *fringe* são representados em mais de um processo simultaneamente, como mostrado na Figura 19. Na figura, a enumeração global das células do processo 1 poderia ser facilmente derivada de sua enumeração local, porém, este não é o caso no processo 2. Para o caso geral, se fez necessário a criação de um *array* para armazenar a relação entre a enumeração local e a global.

A enumeração global passou a ser utilizada somente para a criação de matrizes de sistemas lineares envolvendo o domínio completo, o que é necessário a vários métodos numéricos de solução de Navier-Stokes e de outras EDPs. Um *array* dentro de `psim_domain` ou `psim_facet_domain` permite se obter rapidamente o *id* global correspondente a partir de um *id* local.

Para a sincronização entre processos com subdomínios vizinhos (onde os elementos de se estendem para o *fringe* do outro), cada processo cria dois tipo de dados composto MPI para cada um de seus vizinhos: um de envio e outro de recebimento. Essa funcionalidade do MPI permite especificar precisamente quais e em que ordem valores do *array* local serão enviados, e em quais posições os dados recebidos serão armazenados no *array* local. O processo de sincronização se resume, então, a um par de chamadas `MPI_Isend()` e `MPI_Irecv()` para cada processo vizinho.

Ao contrário do *broadcast*, essa operação de comunicação é local: não importa

como o domínio se expanda para além da área de contato entre vizinhos, o custo dessa comunicação sempre será o mesmo, pois para cada processo, somente os seus vizinhos serão envolvidos na comunicação.

Essa propriedade se traduz em um custo de comunicação de $O(\frac{n}{P})$ (no caso ideal onde a topologia da conectividade entre subdomínios se sobrepõe à topologia da rede de computadores do *cluster* onde a simulação é executada). No que tange à sincronização das propriedades físicas, esse novo algoritmo permite que, ao menos em princípio, a simulação escale indefinidamente, enquanto P aumentar proporcionalmente a n.

Parte II

Modelagem e Solução do Escoamentos de Fluidos

8 Modelagem física e diferencial

Para tornar tratável o problema de simular um escoamento de fluído real, com todas as suas complexidades, são necessárias suposições e simplificações que delimitam o escopo do problema suportado. A seguir, estão descritas as hipóteses que caracterizam o modelo físico utilizado na implementação do Cyberex.

8.1 Hipóteses simplificadoras do modelo físico

Como em qualquer disciplina da física, muitas hipóteses são aceitas implicitamente pelo contexto, portanto a lista apresentada a seguir não é exaustiva. Entretanto, ela é suficiente para se delimitar a quais aplicações práticas de engenharia o presente trabalho é relevante. Esperamos que à medida que o desenvolvimento do Cyberex e da HigTree progridam dentro do projeto, algumas das hipóteses possam ser relaxadas ou removidas.

8.1.1 Hipótese do contínuo

A primeira hipótese feita é a hipótese do contínuo, onde supõe-se que o fluido é um meio contínuo, e que energia cinética é uma grandeza separada de energia térmica. Portanto fenômenos dependentes da natureza discreta do fluido, como movimento browniano ou efeitos de reentrada em aplicações aeroespaciais não podem ser modelados.

Esta é a principal hipótese feita para a dedução da equação do balanço de massa e da equação do balanço da quantidade de movimento linear de Cauchy, que descrevem o comportamento macroscópico do fluido. São elas, em notação indicial:

$$\frac{\partial \rho}{\partial t} + \frac{\partial (\rho u_i)}{\partial x_i} = 0 \tag{8.1}$$

$$\frac{\partial (\rho u_i)}{\partial t} + \frac{\partial (\rho u_j u_i)}{\partial x_j} - \frac{\partial \sigma_{ij}(\boldsymbol{u})}{\partial x_j} - \varphi_i = 0 \tag{8.2}$$

onde:

- ρ é a massa específica do fluido;
- σ_{ij} é o tensor de tensões de Cauchy;
- $\boldsymbol{u}$ é o vetor velocidade local;
- u_i é a i-ésima componente da vetor velocidade;
- t é o tempo;

- x_i é a i-ésima componente do vetor posição;
- φ_i é a i-ésima componente da força de corpo.

8.1.2 Incompressibilidade

Considera-se, também, que o escoamento é incompressível. Esta suposição é geralmente válida para problemas cujo número de Mach seja de até 0,3, onde a compressibilidade é desprezível. Isto simplifica as equações diferenciais do problema para:

$$\frac{\partial u_i}{\partial x_i} = 0 \tag{8.3}$$

$$\rho\frac{\partial u_i}{\partial t} + \rho\frac{\partial (u_j u_i)}{\partial x_j} - \frac{\partial \sigma_{ij}(\boldsymbol{u})}{\partial x_j} - \varphi_i = 0 \tag{8.4}$$

Poderíamos simplificar ainda mais o termo $\rho\frac{\partial (u_j u_i)}{\partial x_j}$ para a forma não divergente, que é matematicamente equivalente: $\rho u_i \cancelto{0}{\frac{\partial u_j}{\partial x_j}} + \rho u_j \frac{\partial u_i}{\partial x_j}$. Entretanto, essa simplificação não é utilizada pois, em sua forma discretizada, estes termos não são equivalentes, sendo a forma divergente mais estável numericamente.

8.1.3 Fluido newtoniano

Utilizamos o modelo de Stokes, de modo que supomos que o fluido seja newtoniano, portanto é um fluido cujas tensões cisalhantes são proporcionais às respectivas taxas de deformação.

Aplicando a separação entre pressão fluidostática p e o tensor deviatórico τ_{ij}:

$$\sigma_{ij} = -p\delta_{ij} + \tau_{ij}, \tag{8.5}$$

onde δ_{ij} é o delta de Kronecker, *i.e.*,

$$\delta_{ij} = \begin{cases} 1, & \text{se } i = j \\ 0, & \text{se } i \neq j \end{cases}, \tag{8.6}$$

o modelo de fechamento proposto por Stokes modela o tensor τ_{ij} como:

$$\tau_{ij} = \mu\left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i} + \frac{2}{3}\delta_{ij}\cancel{\frac{\partial u_k}{\partial x_k}}\right) + \delta_{ij}\lambda\cancel{\frac{\partial u_k}{\partial x_k}} = \mu\left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i}\right)., \tag{8.7}$$

onde μ é a viscosidade dinâmica do fluido, e λ é o segundo coeficiente da viscosidade.

Substituindo as Equações 8.5 e 8.7 na Equação 8.4, temos as equações de Navier-Stokes para escoamentos incompressíveis:

$$\rho\frac{\partial u_i}{\partial t} + \rho\frac{\partial (u_j u_i)}{\partial x_j} + \frac{\partial p}{\partial x_i} - \frac{\partial}{\partial x_j}\left[\mu\left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i}\right)\right] - \varphi_i = 0 \tag{8.8}$$

8.1.4 Escoamento monofásico

Fazemos a suposição de que o escoamento simulado é monofásico. Portanto, não há tensão interfacial e as propriedades físicas do fluido são contínuas.

8.1.5 Temperatura constante

Considera-se que o escoamento é isotérmico, e portanto sua temperatura é um valor constante, que não precisa ser modelado. Para isso, temos também de supor que a energia cinética convertida em energia térmica pelo termo viscoso é desprezível, e portanto não altera a temperatura do fluido e não pode influenciar o escoamento. Já havíamos feito a suposição de incompressibilidade, o que implica na densidade ρ do fluido ser constante. Agora podemos supor, por ser isotérmico, que a viscosidade dinâmica μ do fluido também é constante.

8.1.6 Geometrias ortogonais

Supõe-se que é possível aproximar o volume da região de interesse do escoamento por um domínio de EDPs cujos limites sejam hiperplanos perpendiculares aos eixos coordenados. Essa limitação é imposta porque atualmente a HigTree suporta somente domínios formados por um conjunto de hiper-retângulos.

8.1.7 Escalas pequenas desinteressantes

Existem várias técnicas para se lidar com a dificuldade inerente de simular escoamentos turbulentos. No presente trabalho, foi utilizada a simulação de grandes escalas (LES). Nos casos que assim necessitam, somente as grandes escalas da turbulência são computadas. Os efeitos das menores escalas são modelados através do termo de viscosidade turbulenta μ_t.

A utilização desta técnica é necessária somente pela limitação dos recursos computacionais exigidos para se computar diretamente todas as escalas da turbulência, o que é chamado de simulação numérica direta (DNS). Ela é opcional, pois o modelo não é necessário em escoamentos não turbulentos, ou no caso do usuário estar disposto a pagar o custo em tempo de processamento e espaço em memória necessários para se realizar uma DNS sobre um escoamento turbulento.

O termo da viscosidade dinâmica turbulenta μ_t, se usado, é somado à viscosidade dinâmica do fluido. Modificando a Equação 8.8 de acordo, temos:

$$\rho\frac{\partial u_i}{\partial t} + \rho\frac{\partial(u_j u_i)}{\partial x_j} + \frac{\partial p}{\partial x_i} - \frac{\partial}{\partial x_j}\left[(\mu+\mu_t)\left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i}\right)\right] - \varphi_i = 0 \tag{8.9}$$

Note que, no caso de LES ser utilizado (e portanto $\mu_t \neq 0$) a variável u_i representa uma velocidade filtrada, sem as flutuações das menores escalas — que não são explicitamente calculadas.

8.2 Modelo diferencial final

Aplicada todas as hipóteses definidas anteriormente, as EDPs que definem o problema são:

$$\begin{cases} \dfrac{\partial u_i}{\partial x_i} = 0 \\ \rho\dfrac{\partial u_i}{\partial t} + \rho\dfrac{\partial (u_j u_i)}{\partial x_j} + \dfrac{\partial p}{\partial x_i} - \dfrac{\partial}{\partial x_j}\left[(\mu+\mu_t)\left(\dfrac{\partial u_i}{\partial x_j} + \dfrac{\partial u_j}{\partial x_i}\right)\right] - \varphi_i = 0 \end{cases} \tag{8.10}$$

Para simplificar a solução do problema, as equações foram adimensionalizadas, de modo que o único parâmetro das EDPs seja o número de Reynolds do escoamento. Em um primeiro passo, dividimos a Equação 8.9 por ρ, de modo que a equação da quantidade de movimento linear dimesional é dada por:

$$\frac{\partial u_i}{\partial t} + \frac{\partial (u_j u_i)}{\partial x_j} + \frac{1}{\rho}\frac{\partial p}{\partial x_i} - \frac{\partial}{\partial x_j}\left[(\nu+\nu_t)\left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i}\right)\right] - \frac{\varphi_i}{\rho} = 0, \tag{8.11}$$

onde ν_t é a viscosidade cinemática turbulenta.

Então substituímos as variáveis da Equação 8.11 pelas suas versões adimensionais, dadas por:

$$\begin{aligned} u_i &= U\mathring{u}_i \\ p &= \rho U^2 \mathring{p} \\ \nu_t &= UL\mathring{\nu}_t \\ \varphi_i &= \frac{\rho U^2}{L}\mathring{\varphi}_i \\ x_i &= L\mathring{x}_i \\ t &= \frac{L}{U}\mathring{t} \end{aligned}$$

sendo que o número de Reynolds Re é dado por:

$$Re = \frac{UL}{\nu}. \tag{8.12}$$

O resultado é:

$$\frac{U^2}{L}\frac{\partial \mathring{u}_i}{\partial \mathring{t}} + \frac{U^2}{L}\frac{\partial (\mathring{u}_j \mathring{u}_i)}{\partial \mathring{x}_j} + \frac{U^2 \not{\rho}}{L \not{\rho}}\frac{\partial \mathring{p}}{\partial \mathring{x}_i} - \frac{\partial}{\partial \mathring{x}_j}\left[\frac{U(\nu + UL\mathring{\nu}_t)}{L^2}\left(\frac{\partial \mathring{u}_i}{\partial \mathring{x}_j} + \frac{\partial \mathring{u}_j}{\partial \mathring{x}_i}\right)\right] - \frac{\not{\rho} U^2}{\not{\rho} L}\mathring{\varphi}_i = 0 \tag{8.13}$$

Dividindo a Equação 8.13 por $\frac{U^2}{L}$, temos a versão adimensional final da equação do balanço da quantidade de movimento linear:

$$\frac{\partial \mathring{u}_i}{\partial \mathring{t}} + \frac{\partial (\mathring{u}_j \mathring{u}_i)}{\partial \mathring{x}_j} + \frac{\partial \mathring{p}}{\partial \mathring{x}_i} - \frac{\partial}{\partial \mathring{x}_j}\left[\left(\frac{1}{Re} + \mathring{\nu}_t\right)\left(\frac{\partial \mathring{u}_i}{\partial \mathring{x}_j} + \frac{\partial \mathring{u}_j}{\partial \mathring{x}_i}\right)\right] - \mathring{\varphi}_i = 0 \tag{8.14}$$

Ao se adimensionalizar também a Equação 8.3, temos o sistema de equações adimensionais que descreve o problema:

$$\begin{cases} \dfrac{\partial \mathring{u}_i}{\partial \mathring{x}_i} = 0 \\ \dfrac{\partial \mathring{u}_i}{\partial \mathring{t}} + \dfrac{\partial (\mathring{u}_j \mathring{u}_i)}{\partial \mathring{x}_j} + \dfrac{\partial \mathring{p}}{\partial \mathring{x}_i} - \dfrac{\partial}{\partial \mathring{x}_j}\left[\left(\dfrac{1}{Re} + \mathring{\nu}_t\right)\left(\dfrac{\partial \mathring{u}_i}{\partial \mathring{x}_j} + \dfrac{\partial \mathring{u}_j}{\partial \mathring{x}_i}\right)\right] - \mathring{\varphi}_i = 0 \end{cases} \tag{8.15}$$

Somente as equações e variáveis adimensionalizadas serão utilizadas pelo restante do presente trabalho. Por esse motivo, daqui em diante o símbolo sobrescrito $\circ$ será omitido.

9 Modelagem numérica

Para implementação dentro do Cyberex, as Equações 8.15 foram discretizadas por diferenças finitas em malha cartesiana deslocada, com velocidades nas faces e pressão e viscosidade turbulenta no centro das células, conforme mostrado na Figura 20. As fronteiras entre refinamentos e condições de contorno foram tratadas transparentemente, pelo mecanismo de interpolação da HigTree.

A equação da continuidade foi discretizada a partir do centro de cada célula, sendo os sobrescritos ℓ_i e h_i utilizados para representar as faces anterior e posterior, respectivamente, ao longo da coordenada x_i, conforme mostrado na Figura 21. Dela, obtém-se n equações discretas — onde n é o número de células do domínio — no seguinte formato:

$$\frac{u_i^{h_i}}{\Delta x_i} - \frac{u_i^{\ell_i}}{\Delta x_i} = 0 \tag{9.1}$$

Já as equações do balanço da quantidade de movimento foram discretizadas a partir das facetas, originando m equações discretas, onde m é o número total de facetas

Figura 20 – Localização das variáveis deslocadas na malha cartesiana, para o caso $D = 2$.

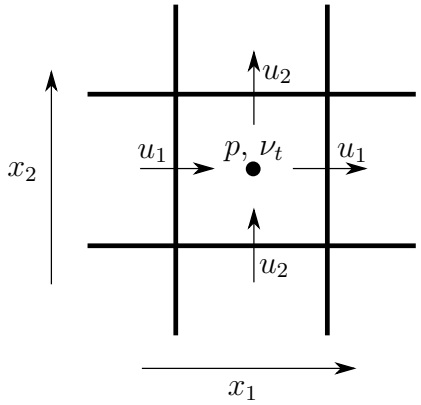
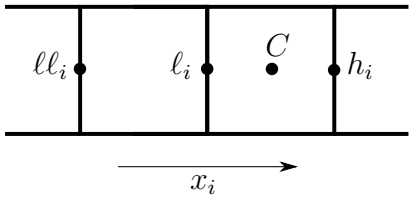
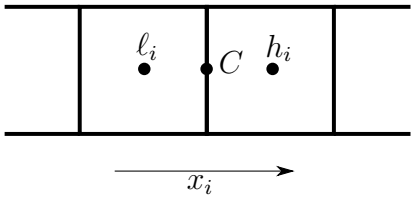


Figura 21 – Localização dos pontos relativos ao centro da discretização ao longo de uma coordenada espacial.

(a) Discretização centrada na célula.



(b) Discretização centrada na faceta.



no domínio:

$$\left(\frac{3u_i^C - 4u_i^{C*} + u_i^{C**}}{2\Delta t}\right) + \mathbb{D}_j^C(\breve{u}_i u_j) + \mathbb{D}_i^C(p)$$
$$- \left[\left(\frac{1}{Re} + \nu_t^{h_j}\right)\left(\frac{\mathbb{D}_j^{h_j}(u_i)}{\Delta x_j} + \frac{\mathbb{D}_i^{h_j}(u_j)}{\Delta x_j}\right) - \left(\frac{1}{Re} + \nu_t^{\ell_j}\right)\left(\frac{\mathbb{D}_j^{\ell_j}(u_i)}{\Delta x_j} + \frac{\mathbb{D}_i^{\ell_j}(u_j)}{\Delta x_j}\right)\right]$$
$$-\varphi_i(\boldsymbol{x}^C) = 0 \tag{9.2}$$

onde:

- $\mathbb{D}_i^C(\zeta) = \dfrac{\zeta^{h_i}}{\Delta x_i} - \dfrac{\zeta^{\ell_i}}{\Delta x_i}$;
- i subscrito corresponde à direção coordenada perpendicular à faceta de origem da discretização;
- C sobrescrito localiza o ponto na origem da discretização;
- $\ell\ell_i$, ℓ_i e h_i sobrescritos localizam as variáveis avaliadas, respectivamente, em $-\frac{3}{2}\Delta x_i$, $-\frac{1}{2}\Delta x_i$ e $\frac{1}{2}\Delta x_i$ a partir de C, ao longo da coordenada x_i, conforme ilustrado na Figura 21;
- $*$ e $**$ sobrescritos indicam, respectivamente, o valor da variável no último e no penúltimo passo de tempo;
- $\breve{u}$ indica o fluxo médio de uma faceta virtual naquela posição, e é dado pelo esquema advectivo utilizado.

Como a Equação 9.2 é discretizada a partir da faceta onde a variável u_i^C está localizada, esta é obtida diretamente. Todas as outras variáveis são obtidas através do módulo de interpolação da HigTree. Nos pontos onde a variável discreta está localizada na posição exata, esta é retornada imediatamente, e nenhuma interpolação é feita. Caso contrário, seu valor é interpolado a partir dos pontos vizinhos, usando um polinômio de grau dado no arquivo de configuração do Cyberex. Isso é feito transparentemente pela HigTree.

O termo $\varphi_i(\boldsymbol{x}^C)$ é dado no arquivo de configuração, componente por componente. Cada uma das D componentes pode ter um valor constante, ou ser dado por uma função do espaço (em um *script* em Python, por exemplo), e portanto sua avaliação, quando necessária, é obtida diretamente.

9.1 Termo temporal

O termo temporal é dado pelo esquema de Euler implícito de segunda ordem — como utilizado por Deng et al. (1994) — exceto no primeiro passo de tempo, onde não

existe valor de $\boldsymbol{u}$ no penúltimo passo de tempo. Neste caso específico, um esquema de primeira ordem é utilizado, e o termo temporal passa a ser:

$$\frac{u_i^C}{\Delta t} - \frac{u_i^{C*}}{\Delta t} \tag{9.3}$$

9.2 Esquemas advectivos

Para definir o valor de alguma componente da velocidade $\breve{u}$ ao longo da direção x_i, vários esquemas advectivos foram implementados no Cyberex. O mais simples deles, o esquema de diferenças centradas (CDS), é um esquema de segunda ordem dado por:

$$\breve{u}^C = \frac{u^{\ell_i} + u^{h_i}}{2} \tag{9.4}$$

O CDS possui fortes restrições quanto a sua estabilidade numérica, criando oscilações não físicas quando o número de Peclet local da malha é maior do que 2 (RUNCHAL, 1987). Essa restrição limita sua aplicabilidade em escoamentos com alto número de Reynolds. Como alternativa, foram implementados três outros esquemas de segunda ordem com viés a montante (esquemas *upwind*), que, por serem mais estáveis, podem ser utilizados em uma gama maior de situações.

Todos os esquemas *upwind* se identificam por darem um peso maior aos pontos à montante do escoamento na interpolação do valor do fluxo. A vantagem de esquemas *upwind* sobre esquemas centrados para problemas advectivos é bem documentada na literatura — ver, por exemplo, Leonard (1996). Sem perda de generalidade, os esquemas advectivos descritos a seguir supõem $u_j > 0$, e portanto o viés é pelos pontos à esquerda do centro.

O esquema *upwind* de segunda ordem (2UP) mais simples implementado é:

$$\breve{u}^C = \frac{3u^{\ell_i} - 1u^{\ell\ell_i}}{2} \tag{9.5}$$

A análise do 2UP mostra que, como sendo uma extrapolação linear, ele sofre de *over/undershots*, e portanto não é monotônico, o que pode causar oscilações (ALVES; OLIVEIRA; PINHO, 2003). Dois outros esquemas *upwind* que resolvem este problema foram implementados. O *Curved-Line Avection Method* (CLAM), que conforme Alves, Oliveira e Pinho (2003 apud LEER, 1974), é dado por:

$$\breve{u}^C = \begin{cases} \dfrac{2u^{h_i}u^{\ell_i} - \left(u^{\ell_i}\right)^2 - u^{h_i}u^{\ell\ell_i}}{u^{h_i} - u^{\ell\ell_i}}, & \text{se } 0 < \phi^C < 1 \\ u^{\ell_i}, & \text{caso contrário.} \end{cases} \tag{9.6}$$

Onde:

$$\phi^C = \frac{u^{\ell_i} - u^{\ell\ell_i}}{u^{h_i} - u^{\ell\ell_i}} \tag{9.7}$$

O CLAM é não linear no intervalo $0 < \phi^C < 1$, e isto o torna único esquema advectivo localmente não linear implementado. Quando multiplicado aos fatores do termo advectivo, pode tornar o sistema de equações ainda mais difícil de ser resolvido, dependendo de como for feito tratamento das não linearidades.

Introduzido em Alves, Oliveira e Pinho (2003), o esquema *Convergent and Universally Bounded Interpolation Scheme for the Treatment of Advection* (CUBISTA) é localmente linear. Ele aproxima por segmentos de reta a curva de interpolação do CLAM, e para isso também depende da relação ϕ^C. Ele também foi implementado dentro do Cyberex, e é dado por:

$$\breve{u}^C = \alpha \left(u^{\ell_i} - u^{\ell\ell_i} \right) + \beta \left(u^{h_i} - u^{\ell\ell_i} \right) + u^{\ell\ell_i} \tag{9.8}$$

onde

$$\begin{cases} \alpha = \dfrac{7}{4},\ \beta = 0, & \text{caso } 0 < \phi^C < \dfrac{3}{8} \\ \alpha = \dfrac{3}{4},\ \beta = \dfrac{3}{8}, & \text{caso } \dfrac{3}{8} \leq \phi^C \leq \dfrac{3}{4} \\ \alpha = \dfrac{1}{4},\ \beta = \dfrac{3}{4}, & \text{caso } \dfrac{3}{4} < \phi^C < 1 \\ \alpha = 1,\ \beta = 0, & \text{nos outros casos.} \end{cases} \tag{9.9}$$

9.3 Modelagem submalha

Conforme descrito na subseção 8.1.7, se LES for utilizado para tratar escoamentos turbulentos, é necessário modelar o efeito das pequenas escalas através do termo de viscosidade turbulenta ν_t. Seu cálculo é feito de maneira explícita, ou seja, é feito a partir do valor das velocidades no último passo de tempo conhecido. O campo ν_t é então utilizado na solução dos campos de velocidade e pressão do próximo passo de tempo. Os valores discretos deste campo de viscosidade turbulenta são armazenados no centro das células do domínio, e interpolados através da HigTree nas faces, onde são necessários.

No presente trabalho, foi utilizado o modelo de Smagorinsky, com a opção de se usar a atenuação de van Driest (simbolizado por At) para reduzir a viscosidade turbulenta nas regiões próximas às paredes, conforme dado por Ferziger e Peric (2002, p. 279-280). Pelo modelo de Smagorinsky, ν_t, já em sua forma adimensional, é dado por:

$$\nu_t = At^2 C_S^2 \Delta^2 \|S\|_F \tag{9.10}$$

sendo que C_S é a constante de Smagorinsky, dada como parâmetro de configuração pelo usuário, e S é o tensor taxa de deformação, calculado a partir do campo de velocidades conhecido. O Δ é um comprimento da ordem das menores escalas resolvidas, que é calculado

a partir da malha local como uma média geométrica das D dimensões da célula onde ν_t está sendo calculado:

$$\Delta = \left(\prod_{i=1}^{D} \Delta x_i \right)^{\frac{1}{D}}. \tag{9.11}$$

O tensor taxa de deformação S é dado por:

$$S_{ij} = \frac{1}{2} \left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i} \right) \tag{9.12}$$

que é discretizado no centro de cada célula do domínio como:

$$S_{ij} \approx \frac{1}{2} \left[\mathbb{D}_j^C(u_i) + \mathbb{D}_i^C(u_j) \right]. \tag{9.13}$$

O escalar $\|S\|_F$ é a norma de Frobenius do tensor S:

$$\|S\|_F = \sqrt{S_{ij} S_{ij}}. \tag{9.14}$$

Se uma função que determina a posição da parede mais próxima não for dada no arquivo de configuração do Cyberex, então $At = 1$, e a função de van Driest não é utilizada. Caso contrário, a função $P : \mathbb{R}^D \to \mathbb{R}^D$ que, dado um ponto do domínio, determina a posição da parede mais próxima, é utilizada para se calcular At, que definido como:

$$At = \left[1 - \exp\left(-\frac{L}{\nu} \sqrt{\frac{\nu U}{L}} \frac{|P(\boldsymbol{x}^C) - \boldsymbol{x}^C|}{A^+} \sqrt{\mathbb{D}_\perp^W(|\boldsymbol{u}_\parallel|)} \right) \right]^2, \tag{9.15}$$

onde W localiza o centro da célula do domínio mais próximo ao ponto na parede $P(\boldsymbol{x}^C)$, $\perp$ é a direção perpendicular à parede, $\boldsymbol{u}_\parallel$ é a componente da velocidade paralela à parede, e A^+ é um parâmetro constante que, segundo a literatura, deve valer aproximadamente 25.

Existem dificuldades teóricas com essa aplicação do modelo de van Driest. Como colocado por Ferziger e Peric (2002):

> *Although this modification* [van Driest damping] *produces the desired results, it is difficult to justify in the context of LES. An SGS* [subgrid-scale] *model should depend solely on the local properties of the flow and it is difficult to see how the distance from the wall qualifies in this regard.*

Além disso, a adimensionalização utilizada evidencia outro problema: a dependência sobre o adimensional $\frac{L}{\nu}\sqrt{\frac{\nu U}{L}}$. Teoricamente, segundo o modelo diferencial do DNS (sem a modelagem ν_t do LES), o resultado do escoamento depende somente do adimensional Re. A introdução de outro adimensional com influência no escoamento sugere que o parâmetro A^+, dado na literatura como 25, não poderia ser fixo para todos casos, mas sim modelado de modo a tornar a solução independente de $\frac{L}{\nu}\sqrt{\frac{\nu U}{L}}$.

Para tentar manter a configuração do caso dependente somente de um parâmetro, o número de Reynolds, no presente trabalho foi utilizada a aproximação $\frac{L}{\nu}\sqrt{\frac{\nu U}{L}} \approx Re\sqrt{\frac{1}{Re}}$,

que é exata no caso de $L = 1$ e $U = 1$. Tal aproximação é justificada pelo fato de que o papel matemático de $\frac{1}{Re}$ no modelo diferencial adimensional do escoamento é o mesmo de ν no modelo diferencial dimensional. Ao se utilizar essa aproximação, $\frac{1}{Re}$ também passa a fazer, na função de van Driest adimensional, o mesmo papel matemático do ν em sua formulação dimensional. Passamos então a calcular At como:

$$At \approx \left[1 - \exp\left(-\frac{Re|P(\boldsymbol{x}^C) - \boldsymbol{x}^C|}{A^+}\sqrt{\frac{1}{Re}\mathbb{D}_{\perp}^{W}(|\boldsymbol{u}_{\parallel}|)}\right)\right]^2. \tag{9.16}$$

10 Solução monolítica das variáveis primárias

O sistema não linear originado pela discretização dada no capítulo 9 é resolvido monoliticamente, de modo que todas as equações discretas foram agrupadas em um único sistema não linear totalmente implícito, resolvido como um todo. Dependendo do método de tratamento das não linearidades, um ou mais sistemas lineares precisam ser resolvidos. Os métodos utilizados são descritos na seção 10.1.

O formato dos sistemas lineares depende do método, mas todos eles compartilham a característica de que, na forma matricial, uma porção significativa da diagonal principal é zerada. Como mostra a Figura 22, as colunas referentes às variáveis da pressão nas linhas referentes às equações da continuidade formam um bloco esparso ao longo da diagonal principal. Essa característica torna não trivial a solução do sistema linear por métodos iterativos. O problema é abordado em mais detalhes na seção 10.2.

10.1 Tratamento das não linearidades

Foram implementados três métodos para o tratamento das não linearidades, presentes no termo advectivo: linearização, iteração de ponto fixo e método de Newton-Raphson generalizado para múltiplas funções e múltiplas incógnitas. A linearização é o método mais

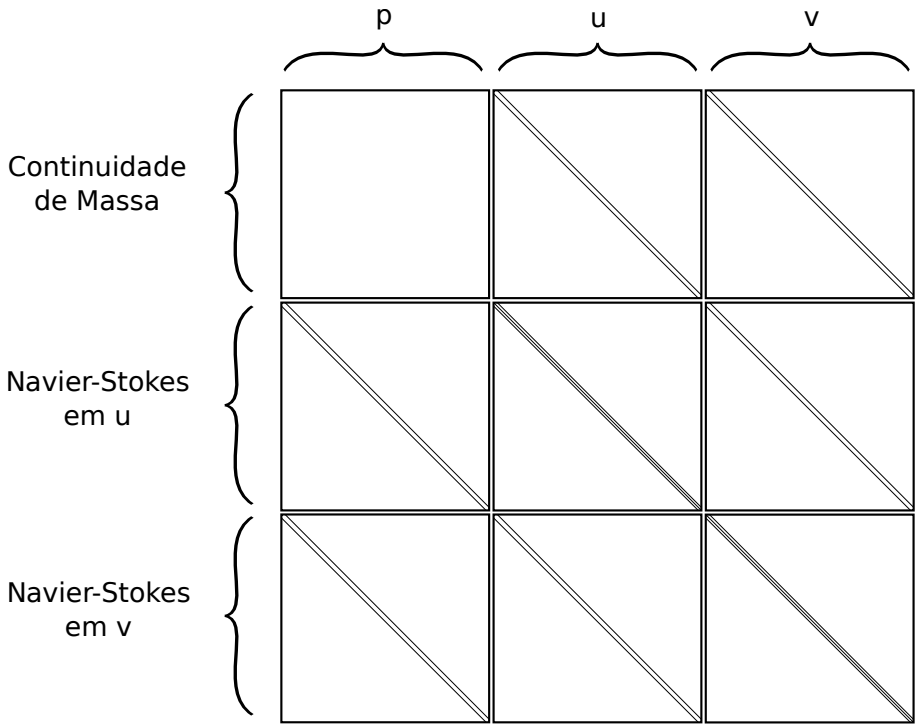


Figura 22 – Posição dos elementos não nulos de uma matriz monolítica 2D.

simples, onde $\breve{u}_i$ é a incógnita do esquema advectivo, e o valor já conhecido do passo de tempo anterior u_j^* é usado no lugar da incógnita u_j.

Os outros dois são métodos iterativos que aproximam a solução de um sistema não linear. Partem de um vetor aproximação inicial $\boldsymbol{s}^0$ e cada iteração sucessiva $\boldsymbol{s}^k$ se aproxima mais da solução exata desconhecida $\boldsymbol{s}$. Em ambos os casos, um sistema linear é resolvido por cada iteração. A diferença entre os métodos está na interpretação dos resultados e montagem do sistema linear, e na condição de parada.

O código foi organizado em uma hierarquia de classes da linguagem C++, onde a classe base `Monolithic` faz o controle do processo iterativo, que é comum a todos os métodos, enquanto chama funções virtuais implementadas nas classes específicas de cada um dos métodos, que derivam da classe `Monolithic`.

As funções implementadas por cada um dos métodos são:

`stop_due_func_norm()` que pode ser usada para definir uma condição de parada baseada no valores calculados para lado direito do sistema linear;

`stop_due_diff_norm()` que pode ser usada para definir uma condição de parada baseada na diferença entre $\boldsymbol{s}^k$ e $\boldsymbol{s}^{k-1}$;

`set_ns_eq()` é usada para definir uma equação (linha da matriz mais lado direito) discretizada a partir do centro de uma dada face — a equação da quantidade de movimento;

`set_mass_eq()` é usada para definir uma equação discretizada a partir do centro de uma célula — a equação da continuidade;

`update_solution()` é usada para finalizar uma iteração do sistema não linear a partir do resultado do sistema linear, calculando $\boldsymbol{s}^{k+1}$.

10.1.1 Método de Newton-Raphson

O método de Newton-Raphson permite aproximar numericamente uma raiz de uma função contínua a partir de um ponto inicial. Uma iteração do método é dada por:

$$s^{k+1} = s^k - \frac{f(s^k)}{f'(s^k)} \tag{10.1}$$

No presente trabalho, foi utilizada uma generalização do método para funções vetoriais. Sendo N é o número total de incógnitas no domínio, $\boldsymbol{f} : \mathbb{R}^N \to \mathbb{R}^N$ é dada por:

$$\boldsymbol{f}(\boldsymbol{s}) = \begin{bmatrix} f_1(\boldsymbol{s}) \\ f_2(\boldsymbol{s}) \\ \vdots \\ f_N(\boldsymbol{s}) \end{bmatrix} \tag{10.2}$$

e $\boldsymbol{f}'$ é sua matriz jacobiana, dada por:

$$\boldsymbol{f}'(\boldsymbol{s}) = \left[\begin{array}{cccc} \frac{\partial f_1}{\partial s_1} & \frac{\partial f_1}{\partial s_2} & \cdots & \frac{\partial f_1}{\partial s_N} \\ \frac{\partial f_2}{\partial s_1} & \frac{\partial f_2}{\partial s_2} & \cdots & \frac{\partial f_2}{\partial s_N} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_N}{\partial s_1} & \frac{\partial f_N}{\partial s_2} & \cdots & \frac{\partial f_N}{\partial s_N} \end{array}\right]\Bigg|_{\boldsymbol{s}} \tag{10.3}$$

então o método de Newton-Raphson pode ser generalizado para:

$$\boldsymbol{s}^{k+1} = \boldsymbol{s}^k - \boldsymbol{f}'(\boldsymbol{s}^k)^{-1}\boldsymbol{f}(\boldsymbol{s}^k) \tag{10.4}$$

O que requer a inversão da matriz jacobiana $\boldsymbol{f}'(\boldsymbol{s}^k)$. Ao invés de fazê-lo explicitamente, a equação pode ser escrita na forma de um sistema linear $A\boldsymbol{x} = \boldsymbol{b}$, que pode ser resolvido por um *solver* iterativo:

$$\boldsymbol{f}'(\boldsymbol{s}^k)\,[\boldsymbol{s}^k - \boldsymbol{s}^{k+1}] = \boldsymbol{f}(\boldsymbol{s}^k) \tag{10.5}$$

O método pode, então, ser utilizado diretamente na solução do sistema não linear composto pelas Equações 9.1 e 9.2. Especificamente, as funções f_1 a f_n foram definidas como sendo as equações da continuidade, e de f_{n+1} até f_N como sendo as equações da quantidade de movimento. Analogamente, as incógnitas s_1 a s_m correspondem aos valores discretos do campo de pressão, e s_{m+1} a s_N à velocidade. Ao montar a matriz jacobiana a partir dessa definição de $\boldsymbol{f}$, nota-se que a maioria dos seus elementos são nulos, e portanto é uma matriz tratável. Os elementos não nulos formam o padrão ilustrado na Figura 22.

É importante notar que as variáveis u e p usadas nas Equações 9.1 e 9.2 não possuem necessariamente correspondência direta com as incógnitas do vetor $\boldsymbol{s}$. Quando não há interpolação, cada valor de u e p corresponde diretamente a alguma variável s_β, mas quando o valor é obtido por interpolação, a propriedade é uma combinação linear das variáveis concretas. Portanto, de maneira genérica:

$$\begin{aligned} p(\boldsymbol{s}) &= \omega_1 s_1 + \omega_2 s_2 + \cdots + \omega_m s_m \\ u(\boldsymbol{s}) &= \omega_{m+1} s_{m+1} + \omega_{m+1} s_{m+2} + \cdots + \omega_N s_N \end{aligned}$$

onde os coeficientes ω_β são os pesos da interpolação calculados pela HigTree.

Pode-se, então, calcular a derivada parcial de cada equação discreta do sistema com relação a cada uma das incógnitas — estas derivadas parciais são os elementos da matriz jacobiana — utilizando-se a regra da cadeia para equações diferenciais parciais, que é dada por:

$$\frac{\partial f(g_1(\boldsymbol{s}), g_2(\boldsymbol{s}), \cdots)}{\partial s_\beta} = \frac{\partial f}{\partial g_1}\frac{\partial g_1}{\partial s_\beta} + \frac{\partial f}{\partial g_2}\frac{\partial g_2}{\partial s_\beta} + \cdots \tag{10.6}$$

Nos casos onde f_α corresponde a uma equação da continuidade:

$$f_\alpha = \frac{u_i^h}{\Delta x_i} - \frac{u_i^\ell}{\Delta x_i} \tag{10.7}$$

podemos calcular cada elemento da linha correspondente da matriz jacobiana como:

$$\frac{\partial f_\alpha}{\partial s_\beta} = \frac{\partial f_\alpha}{\partial u_i^h}\frac{\partial u_i^h}{\partial s_\beta} + \frac{\partial f_\alpha}{\partial u_i^\ell}\frac{\partial u_i^\ell}{\partial s_\beta} \tag{10.8}$$

$$= \frac{1}{\Delta x_i}\frac{\partial u_i^h}{\partial s_\beta} - \frac{1}{\Delta x_i}\frac{\partial u_i^\ell}{\partial s_\beta} \tag{10.9}$$

$$= \frac{1}{\Delta x_i}\omega_\beta^{u_i^h} - \frac{1}{\Delta x_i}\omega_\beta^{u_i^\ell} \tag{10.10}$$

Note que, na maioria das vezes, ambos $\omega_\beta^{u_i^h}$ e $\omega_\beta^{u_i^\ell}$ serão zero, pois para uma dada função f_α, a maioria dos s_β não farão parte nem do estêncil de u_i^h e nem de u_i^ℓ. Por este motivo, a linha da matriz é construída de maneira esparsa: a derivada parcial de f_α é calculada somente em relação às incógnitas que compões os termos presentes na equação — no caso, as incógnitas s_β que compõem os termos u_i^h e u_i^ℓ, conforme dado pela interpolação de HigTree.

Frequentemente uma variável da equação — como u_i^h na equação da continuidade — tem correspondência um para um com uma incógnita procurada. Isso acontece nos casos onde a interpolação não é necessária — como numa região de malha uniforme. Mesmo assim, o código-fonte foi escrito de modo a tratar o caso genérico, onde uma variável é dada como um estêncil de interpolação esparso. Para cada variável g_γ de uma função f_α, é calculado $\frac{\partial f_\alpha}{\partial g_\gamma}$. O resultado é então multiplicado ao estêncil de interpolação de g_γ, calculado pela HigTree a partir das variáveis base $\boldsymbol{s}$ (no caso de correspondência direta entre variável e incógnita, o estêncil tem somente um coeficiente de valor 1, correspondente à incógnita). Os estêncis de cada variável são calculados termo a termo, e ao final, todos os estêncis referentes a f_α são somados, formando a linha da matriz jacobiana.

No caso da continuidade, os termos $\frac{\partial f_\alpha}{\partial g_\gamma}$ computados são:

$$\frac{\partial}{\partial u_i^h}\frac{u_i^h}{\Delta x_i} = \frac{1}{\Delta x_i} \tag{10.11}$$

$$\frac{\partial}{\partial u_i^\ell}\frac{-u_i^\ell}{\Delta x_i} = \frac{-1}{\Delta x_i} \tag{10.12}$$

O método de Newton-Raphson foi implementado na classe `NewtonMethod`, derivada de `Monolithic`. Para cada equação da continuidade f_α, os valores de $\left.\frac{\partial f_\alpha}{\partial s_\beta}\right|_{\boldsymbol{s}^k}$ e $f_\alpha(\boldsymbol{s}^k)$ são calculados dentro da função `set_mass_eq()`, que é invocada uma vez por célula do domínio.

O mesmo procedimento é aplicado às equações da quantidade de movimento linear na função `set_ns_eq()`, porém as derivadas parciais são calculadas para cada termo da equação, e somadas ao final. As derivadas parciais em relação a cada variável são:

- Para o termo temporal:

$$\frac{\partial}{\partial u_i^C}\frac{3u_i^C}{2\Delta t} = \frac{3}{2\Delta t} \tag{10.13}$$

- Para o termo da pressão:

$$\frac{\partial}{\partial p^h}\frac{p^h}{\Delta x_i} = \frac{1}{\Delta x_i} \tag{10.14}$$

$$\frac{\partial}{\partial p^l}\frac{-p^l}{\Delta x_i} = \frac{-1}{\Delta x_i} \tag{10.15}$$

- Para o termo difusivo, temos que:

$$\frac{\partial \mathbb{D}_i^C(\zeta)}{\partial \zeta^{h_i}} = \frac{1}{\Delta x_i} \tag{10.16}$$

$$\frac{\partial \mathbb{D}_i^C(\zeta)}{\partial \zeta^{\ell_i}} = \frac{-1}{\Delta x_i}. \tag{10.17}$$

Considerando F como sendo o termo difusivo:

$$F = \left(\frac{1}{Re} + \nu_t^{h_j}\right)\left(\frac{\mathbb{D}_j^{h_j}(u_i)}{\Delta x_j} + \frac{\mathbb{D}_i^{h_j}(u_j)}{\Delta x_j}\right) - \left(\frac{1}{Re} + \nu_t^{\ell_j}\right)\left(\frac{\mathbb{D}_j^{\ell_j}(u_i)}{\Delta x_j} + \frac{\mathbb{D}_i^{\ell_j}(u_j)}{\Delta x_j}\right) \tag{10.18}$$

então temos que a derivada parcial de F em relação a cada uma das variáveis ζ envolvidas é dada por:

$$\begin{aligned}\frac{\partial F}{\partial \zeta} = &\left(\frac{1+\nu_t^{h_j}Re}{\Delta x_j Re}\right)\left(\frac{\partial \mathbb{D}_j^{h_j}(u_i)}{\partial \zeta} + \frac{\partial \mathbb{D}_i^{h_j}(u_j)}{\partial \zeta}\right) \\ &- \left(\frac{1+\nu_t^{\ell_j}Re}{\Delta x_j Re}\right)\left(\frac{\partial \mathbb{D}_j^{\ell_j}(u_i)}{\partial \zeta} + \frac{\partial \mathbb{D}_i^{\ell_j}(u_j)}{\partial \zeta}\right)\end{aligned} \tag{10.19}$$

- Para o termo advectivo, considerando que:

$$F = \frac{\breve{u}_i^{h_j}u_j^{h_j}}{\Delta x_j} - \frac{\breve{u}_i^{\ell_j}u_j^{\ell_j}}{\Delta x_j} \tag{10.20}$$

então temos:

$$\frac{\partial F}{\partial \breve{u}_i^{h_j}} = \frac{u_j^{h_j}}{\Delta x_j} \tag{10.21}$$

$$\frac{\partial F}{\partial \breve{u}_i^{\ell_j}} = \frac{-u_j^{\ell_j}}{\Delta x_j} \tag{10.22}$$

$$\frac{\partial F}{\partial u_j^{h_j}} = \frac{\breve{u}_i^{h_j}}{\Delta x_j} \tag{10.23}$$

$$\frac{\partial F}{\partial u_j^{\ell_j}} = \frac{-\breve{u}_i^{\ell_j}}{\Delta x_j} \tag{10.24}$$

$$\tag{10.25}$$

Os estêncis esparsos com os valores de $\frac{\partial \breve{u}_i^{h_j}}{\partial s_\beta}$ e $\frac{\partial \breve{u}_i^{\ell_j}}{\partial s_\beta}$ são obtidos diretamente da definição do esquema advectivo utilizado. Este procedimento também é baseado na regra da cadeia para derivadas parciais, pois o valor de $\breve{u}_i$ depende da avaliação de u_i em pontos específicos relativos à malha, que por sua vez dependem da interpolação das incógnitas básicas $\boldsymbol{s}$ dada pela HigTree.

As funções $f_\alpha(\boldsymbol{s}^k)$ são avaliadas a medida que cada linha da matriz é construída, aproveitando as mesmas interpolações que são usadas no cálculo das derivadas parciais. O vetor $\boldsymbol{f}(\boldsymbol{s}^k)$ é usado como lado direito do sistema linear — como mostrado na Equação 10.5. A norma infinita de $\boldsymbol{f}(\boldsymbol{s}^k)$ é usada como condição de parada, onde o processo iterativo do *solver* não linear é interrompido quando a norma é menor que uma dada tolerância, e portanto as Equações 9.1 e 9.2 estão satisfeitas dentro de uma margem de erro.

Ao final da iteração, na função `update_solution()`, a solução do sistema linear é subtraída de $\boldsymbol{s}^k$ para se obter $\boldsymbol{s}^{k+1}$, conforme a Equação 10.5.

10.1.2 Iteração de ponto fixo

Podemos linearizar a Equação 9.2, explicitando uma das velocidades multiplicadas no termo advectivo. Ao fazer isso, transformamos o problema no seguinte sistema linear:

$$\begin{cases} \dfrac{u_i^{h_i}}{\Delta x_i} - \dfrac{u_i^{\ell_i}}{\Delta x_i} = 0 \\ \left(\dfrac{3u_i^C - 4u_i^{C*} + u_i^{C**}}{2\Delta t}\right) + \mathbb{D}_j^C(\breve{u}_i^\dagger u_j) + \mathbb{D}_i^C(p) \\ \quad - \left(\dfrac{1}{Re} + \nu_t^{h_j}\right)\left(\dfrac{\mathbb{D}_j^{h_j}(u_i)}{\Delta x_j} + \dfrac{\mathbb{D}_i^{h_j}(u_j)}{\Delta x_j}\right) \\ \quad + \left(\dfrac{1}{Re} + \nu_t^{\ell_j}\right)\left(\dfrac{\mathbb{D}_j^{\ell_j}(u_i)}{\Delta x_j} + \dfrac{\mathbb{D}_i^{\ell_j}(u_j)}{\Delta x_j}\right) - \varphi_i(\boldsymbol{x}^C) = 0 \end{cases} \tag{10.26}$$

onde $\breve{u}_i^\dagger$ se refere ao valor mais recente conhecido para a variável (ou uma aproximação inicial), ao invés de ser uma incógnita. Sendo um sistema linear, este pode ser resolvido por um método iterativo, como descrito na seção 10.2. Na primeira iteração de ponto fixo, $u_i^{\beta\dagger}$ é o valor da velocidade no passo de tempo anterior (ou inicial, se for o início da simulação). Nas próximas iterações, $u_i^{\beta\dagger}$ é o valor calculado na iteração anterior.

Podemos escrever o sistema linear das Equações 10.26 no formato de uma multiplicação matriz-vetor $A\boldsymbol{s} = \boldsymbol{b}$. Neste caso, a solução é dada por $F(\boldsymbol{s}^\dagger) = [A(\boldsymbol{s}^\dagger)]^{-1}\boldsymbol{b} = \boldsymbol{s}$. Note que A é uma função de $\boldsymbol{s}^\dagger$, e portanto F também. A solução exata do sistema não linear se dá quando $\boldsymbol{s} = \boldsymbol{s}^\dagger$, ou seja, no ponto fixo da função F, tal que $\boldsymbol{s} = F(\boldsymbol{s})$. O método de iteração de ponto fixo baseia-se na suposição de que se resolvermos este sistema linear usando $\boldsymbol{u}_i^\dagger = \boldsymbol{u}_i^k$ (onde $\boldsymbol{u}_i^k \subset \boldsymbol{s}^k$), em sendo $\boldsymbol{s}^k$ uma aproximação para a solução exata do

sistema não linear, então $\boldsymbol{s}^{k+1}$ será uma aproximação ainda melhor. Logo, o procedimento iterativo pode ser descrito como $\boldsymbol{s}^{k+1} = F(\boldsymbol{s}^k)$.

Foi implementado na classe `FixedPointMethod`, derivada de `Monolithic`. Nesta classe, as funções `set_mass_eq()` e `set_ns_eq()` montam as linhas do sistema linear diretamente a partir da definição dada pelas Equações 10.26.

A função `update_solution()` atualiza os valores de pressão e velocidade para os valores resultados da solução do sistema linear, além de calcular a norma infinita da diferença entre a solução anterior e a atual. A condição de parada é implementada na função `stop_due_diff_norm()`, que simplesmente verifica se a norma calculada é maior que uma dada tolerância. Se for, as Equações 9.1 e 9.2, que formam o sistema não linear, estão satisfeitas dentro de uma tolerância de erro, pois $u_i^\dagger \approx u_i$.

É interessante notar que a variável escolhida para a linearização é uma velocidade na direção de x_i, o que deixa as velocidades perpendiculares u_j como incógnitas na matriz, e portanto exibe também o mesmo padrão de elementos não nulos mostrado na Figura 22. Isso foi feito pois, nos testes que executei durante a implementação, a convergência desta linearização se mostrou mais rápida e robusta do que a solução contrária. Quando a solução alternativa foi testada, explicitando as componentes u_j da velocidade, a solução entrava em um regime de oscilação, onde norma da diferença entre $\boldsymbol{s}^k$ e $\boldsymbol{s}^{k+1}$ permanecia constante e acima da tolerância requisitada.

10.1.3 Linearização

A linearização é a solução mais simples para o problema das não linearidades. Utilizando esta técnica, simplesmente supõe-se que a diferença entre o passo de tempo atual e o passo de tempo anterior é pequena o suficiente, de modo que u_j^* é utilizado no lugar de u_j no termo advectivo da Equação 9.2 para tornar o problema um sistema linear, e que o erro introduzido tem impacto desprezível na solução. Note que esta é uma linearização diferente da utilizada no método da iteração de ponto fixo, e foi escolhida meramente pelo fato de ser a solução comumente utilizada na literatura.

Mesmo com a diferença no termo linearizado, este método não é diferente o suficiente do método de ponto fixo para justificar a criação de sua própria classe derivada da classe `Monolithic`. Em vez disso, o método foi implementado como uma especialização de *template* da classe `FixedPointMethod`. Um parâmetro de *template* é usado para indicar se a técnica de linearização está sendo utilizada. Neste caso, a variável explicitada no termo advectivo é trocada, e a condição de parada muda de modo que somente uma iteração é executada.

10.2 Solução dos sistemas lineares

Em todos os métodos implementados é necessário resolver um sistema linear, cujos elementos não nulos da matriz estão representados na Figura 22. Por conter elementos nulos na diagonal principal, sua solução não é trivial. Nesta seção são descritas as várias alternativas testadas até a solução definitiva ser encontrada na decomposição de Schur.

10.2.1 Enumeração unificada

As linhas e colunas da matriz do sistema linear são identificadas de 0 até $N-1$, onde N é o tamanho da matriz quadrada. Na HigTree, cada processo é responsável por preencher um intervalo contínuo de linhas dentro da matriz. Uma configuração válida onde 4 processos preenchem uma matriz de 100 linhas poderia ser, por exemplo: o processo 0 responsável pelas 25 linhas a partir da linha 0, o processo 1 responsável pelas 25 linhas a partir da linha 25, e assim por diante. Cada processo guarda somente o início e o tamanho da área da matriz sob sua responsabilidade.

Originalmente na HigTree, somente uma propriedade física de cada vez poderia ser incógnita do sistema linear, então a criação do índice global de cada elemento era feito independentemente para cada propriedade física. Para ser possível construir um único sistema linear contendo todas as componentes da velocidade e pressão juntas, foi necessário implementar um algoritmo alternativo de enumeração global.

Foi criada na HigTree a função `psd_set_joint_gid()` que recebe como parâmetro n propriedades centradas nas células, e m propriedades centradas nas faces. Essa função, então, recalcula a enumeração global de modo que cada índice seja único dentre todas as propriedades fornecidas. No caso da solução monolítica, a pressão e as componentes da velocidade são passadas para essa função, gerando um índice que é único para cada valor dentre todos os valores de pressão e velocidade. Esses índices são agrupados por processo, portanto todos os pontos de pressão e velocidade dentro de um único processo formam uma sequência contínua dentro da sequência global.

10.2.2 Solução do sistema linear

As matrizes construídas pelos métodos monolíticos implementados são ocas: possuem uma submatriz nula ao longo da diagonal principal. Esta situação ocorre nas equações da continuidade, que apesar de não dependerem da pressão, estão dispostas nas linhas correspondentes às incógnitas da pressão no sistema não linear, conforme mostrado na Figura 22. Ter elementos nulos na diagonal principal não determina por si só a singularidade da matriz, e portanto não impede o sistema linear de ter solução única. Entretanto, a maioria dos precondicionadores iterativos e alguma implementações de *solver* direto não suportam elementos nulos na diagonal principal, o que limita as possibilidades da

aplicação do método. No presente trabalho, a princípio foi utilizado um método direto de decomposição LU, e mais tarde, um precondicionador baseado no complemento de Schur.

10.2.2.1 Decomposição LU

Inicialmente o sistema linear foi resolvido usando o método de decomposição LU do UMFPACK (DAVIS, 2004), disponibilizado pelo PETSc. Das várias implementações de decomposição LU nele disponíveis, essa foi a única testada que conseguiu resolver o sistema linear sem necessidade de outros ajustes na configuração do PETSc. Entretanto, este pacote não oferece paralelização, e portanto a execução estava limitada a um único processo.

A falta de suporte a execução paralela é uma limitação desta implementação específica, porém a própria técnica de decomposição LU possui limitações fortes, tanto em tempo de execução quando em memória necessária para a execução. A decomposição LU de uma matriz esparsa quase sempre gera matrizes densas, o que acarreta em um custo de memória da ordem de $O(N^2)$, onde N é o número de linhas da matriz. Já o tempo de execução é da ordem do custo de uma multiplicação de matrizes (BUNCH; HOPCROFT, 1974), que é tipicamente de $O(N^3)$. De fato, todo método direto geral de solução de sistemas lineares possui complexidades semelhantes, que é o principal limitante do tamanho do problema que pode ser resolvido com eles.

10.2.2.2 Manipulações algébricas sobre a matriz

Para tentar resolver o sistema linear usando um *solver* iterativo, que é a opção viável para problemas grandes, foi criada artificialmente equações para a pressão. Estas são somadas às equações da continuidade de modo à dar um valor não nulo à diagonal principal da pressão.

Para cada variável discreta da pressão p_i, cada equação da quantidade de movimento que possui um termo $c\,p_i$ é adiciona, caso c seja positivo, ou subtraída, caso c seja negativo, a uma equação artificial para p_i. Este procedimento, embora não garanta dominância do coeficiente de p_i, ao menos garante que seu valor não seja anulado durante a soma. Esta equação é então somada à linha do sistema linear correspondente à incógnita p_i, garantindo um elemento não nulo na diagonal principal da matriz.

Por se tratar de somas e subtrações entre equações de um sistema linear, a solução do sistema permanece a mesma, porém agora com a possibilidade de ser tratado por todos os métodos de solução que requerem uma diagonal não nula, que corresponde à vasta maioria dos *solvers*.

Este procedimento deixou a montagem da matriz mais cara, tanto pelas operações extras, quanto por aumentar o número de elementos não nulos da matriz esparsa,

mas em contrapartida permitiu utilizar o método de decomposição LU fornecido pelo MUMPS (AMESTOY et al., 2001), que apesar de ainda ser um método direto, suporta execução paralela e se mostrou mais estável que o UMFPACK.

Mesmo com estas modificações, ainda não foi possível utilizar um *solver* iterativo. O *solver* iterativo até então utilizado na HigTree era o BiCGSTAB precondicionado com *multigrid* algébrico, por se ter se mostrado o mais eficiente para resolver o sistema linear da pressão no método da projeção. Já no caso monolítico, mesmo sendo possível executá-lo com os zeros removidos da diagonal principal, este *solver* falha em convergir.

Outras alternativas tentadas, que funcionaram com o *solver* direto MUMPS, mas falharam em tornar o sistema bem condicionado o suficiente para ser resolvido iterativamente, foram:

- Reordenar os elementos da matriz com o algoritmo a garantir que nenhum elemento da diagonal principal seja nulo;
- Pré-multiplicar o sistema linear por A^T, computando explicitamente A^TA e $A^T\boldsymbol{b}$, de modo a resolver o sistema $A^TA\boldsymbol{x} = A^T\boldsymbol{b}$.

10.2.2.3 Método do complemento de Schur

Outra alternativa tentada foi a utilização de um método específico para solução de matrizes ocas, denominado complemento de Schur (ELMAN et al., 2008), que também já está implementado no PETSc. A configuração deste *solver* em particular não é trivial, e portanto, foi criado dentro da HigTree um novo tipo de *solver*, denominado `hollow`, já configurado com o melhor conjunto de opções encontrado até o momento para resolver o sistema linear que advém da formulação monolítica. A opção de *solver* `hollow` da HigTree nada mais é do que a opção `petsc`, porém já configurada para este tipo específico de problema, usando o complemento de Schur.

Dada uma matriz A decomposta em 4 quadrantes, não necessariamente de mesmo tamanho:

$$A = \begin{pmatrix} A_{00} & A_{01} \\ A_{10} & A_{11} \end{pmatrix}$$

o complemento de Schur S de A_{11} é:

$$S = A_{00} - A_{01}A_{11}^{-1}A_{10}.$$

A propriedade interessante de S é que, calculando-se apenas S^{-1} e A_{11}^{-1}, é possível obter A^{-1}:

$$A^{-1} = \begin{pmatrix} I & 0 \\ -A_{11}^{-1}A_{10} & I \end{pmatrix} \begin{pmatrix} S^{-1} & 0 \\ 0 & A_{11}^{-1} \end{pmatrix} \begin{pmatrix} I & -A_{01}A_{11}^{-1} \\ 0 & I \end{pmatrix}$$

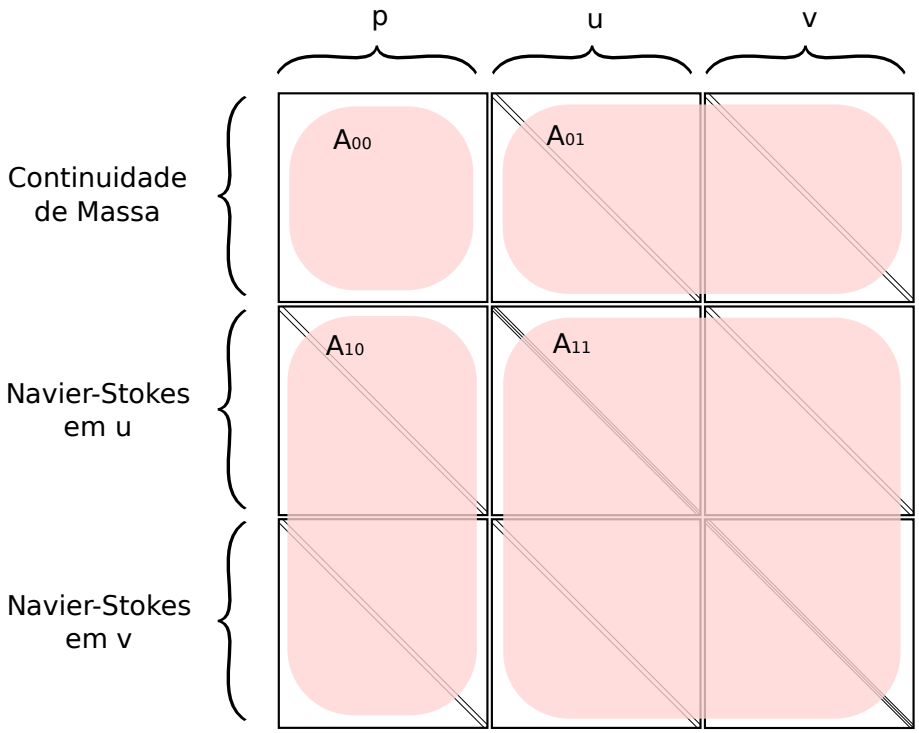


Figura 23 – Divisão da matriz monolítica 2D no método do complemento de Schur.

Se considerarmos A_{00} como sendo o bloco nulo ao longo da diagonal principal, o benefício do método se torna evidente, pois o bloco restante A_{11}, que precisa ser invertido, envolve somente os termos das velocidades das equações da quantidade de movimento (conforme mostrado na Figura 23) e possui dominância diagonal, portanto é numericamente bem comportado e fácil de resolver com métodos iterativos convencionais.

Na prática, como implementado pelo PETSc, o complemento de Schur em si, que é definido em função de A_{11}^{-1}, não é calculado. Em vez disso, é aproximado através do precondicionador especial LSC, proposto por Elman et al. (2005) e também implementado no PETSc. O modo como as inversões de A_{11} e S são feitas é configurável. Em A_{11} foi utilizado a já conhecida combinação de BiCGSTAB com *multigrid* algébrico. Já em S, o precondicionador utilizado também foi o *multigrid* algébrico, porém a solução total se mostrou um pouco mais rápida quando o GMRES fora utilizado como método de Krylov no lugar do BiCGSTAB. O PETSc é capaz de automaticamente detectar o bloco diagonal nulo na matriz, e fazer a subdivisão adequada ao método.

O tempo gasto na solução do sistema linear utilizando este método ficou comparável ao tempo gasto na solução da equação de Poisson no método da projeção, originalmente implementado na HigTree, e ordens de magnitude mais rápido do que utilizando a decomposição LU, sendo portanto, a solução viável encontrada para resolver problemas de grande porte.

11 Validação e Resultados

Utilizando problemas 2D com solução analítica conhecida, os métodos numéricos implementados foram verificados com testes de independência de malha. O *software* foi então utilizado para simular dois escoamentos sobre degrau descendente — um laminar 2D e um turbulento 3D — cujos resultados foram comparados com resultados dados pela literatura.

11.1 Testes de independência de malha

Três diferentes soluções analíticas 2D foram utilizadas para os testes de independência de malha: cavidade com tampa deslizante, *inflow* e *outflow* ortogonais, e uma variante dos vórtices de Taylor-Green para regime permanente. Em todos estes casos, o domínio é quadrado, de lados variando de 0 a 1. As execuções foram feitas em um computador de mesa, em paralelo com 4 processos MPI — um para cada unidade de processamento física disponível. Os testes englobaram as seguintes funcionalidades do *software*:

- malha uniforme e refinada;
- condições de contorno periódica, de Dirichlet e de Neumann;
- os vários esquemas advectivos implementados.

Para determinar o Δt de cada passo de tempo, foi utilizado um algoritmo estocástico simples. O usuário determina no arquivo de configuração os valores máximo, mínimo e inicial de um adimensional característico Ac. O Δt é calculado tal que, em todos os pontos da malha, tanto número de Courant quanto o número de difusão locais sejam no máximo Ac. Ou seja, Δt é calculado de modo a garantir que, em todas as facetas da malha:

$$Ac \geq \delta_{ij} \frac{u_j^C \Delta t}{\Delta x_j} \tag{11.1}$$

e que em todas as células da malha:

$$Ac \geq \frac{\Delta t}{Re[\min(\Delta x_1, \ldots, \Delta x_D)]^2}. \tag{11.2}$$

O valor de Ac inicial é usado no cálculo do Δt no primeiro passo de tempo, e o tempo de processamento t_P gasto na execução é medido. Ac é variado aleatoriamente em até 10% para mais ou para menos, e o novo valor é utilizado no próximo passo de tempo. Para cada passo de tempo executado, o valor de Ac utilizado é guardado, juntamente

com $\frac{\Delta t}{t_P}$: a taxa de progresso na simulação por tempo real gasto. As taxas de progresso dos 5 últimos passos de tempo são armazenadas, e o Ac da melhor delas é utilizado para determinar, aleatoriamente, o próximo valor que será utilizado.

Este algoritmo simples de otimização dinâmica caminha para o Ac que resulta na taxa de progresso localmente ótima. O procedimento estocástico, que varia aleatoriamente o valor de Ac, garante que ele acompanhe a evolução do ponto ótimo ao longo da simulação.

Os valores mínimo, máximo e inicial de Ac utilizados nessa execução foram, respectivamente, 0,01, 10^5 e 0,9. Como almejamos somente as soluções em regime permanente, desde que a evolução tenda a este regime, as soluções nos passos de tempo intermediários não precisam ter significado físico, e portanto não precisam obedecer à condição de Courant-Friedrichs-Lewy (CFL). Por isso, todos os casos foram executados utilizando o método de Newton-Raphson para tratamento das não linearidades, uma vez que o método não linear permite o uso de passos de tempo bem maiores do que o método linearizado, suportando um valor de Ac de até 10 ou mais.

O método de interação de ponto fixo também é um método não linear, e portanto também estável a altos valores de Ac. Porém, comparado ao método de Newton-Raphson, ele se mostrou consistentemente menos eficiente para atingir o mesmo resultado: apesar da construção da matriz ser mais simples, ele necessita de mais iterações — e portanto mais soluções do sistema linear — para convergir a cada passo de tempo. Por este motivo, ele não foi utilizado nas várias execuções necessárias aos testes de independência de malha.

Os problemas foram resolvidos em malhas progressivamente mais finas, refinadas na razão 1/2 em cada direção coordenada. Para cada malha, o escoamento foi desenvolvido a partir de um campo de velocidade nulo, até que o resíduo do sistema de equações já satisfizesse a condição parada logo no início do passo de tempo, antes da primeira iteração do sistema linear. Esta situação, onde a tolerância é satisfeita no início do passo de tempo, significa que a aproximação não precisa ser melhorada, e portanto o regime permanente fora atingido para aquela tolerância. O valor configurado para a tolerância em todos os casos foi de 10^{-8}, e cada passo de tempo geralmente converge usando de 2 a 5 iterações do sistema não linear. Este valor se mostrou pequeno o suficiente para que o erro da solução do sistema de equações fosse menor que o erro da discretização espacial da malha mais fina utilizada, não interferindo nos resultados dos testes de independência de malha.

Note que a tolerância do sistema não linear é somente uma das tolerâncias configuradas. As outras são as tolerâncias dos resíduos relativo e absoluto do sistema linear, resolvido pelo método do complemento de Schur. A melhor *performance* é atingida quando o *solver* linear faz poucas iterações (uma ou duas) por cada iteração do método não linear. Para isso, a tolerância do resíduo relativo foi configurada com um valor baixo, de 10^{-2}. Como no método de Newton-Raphson o lado direito do sistema linear é o próprio resíduo do sistema não linear, garantimos que a solução linear é só duas ordens de magnitude

melhor que a melhor aproximação conhecida, evitando o desperdício de tempo de processamento gasto para encontrar uma solução precisa para uma aproximação ruim. A tolerância absoluta é um limite de segurança que geralmente não é atingido durante a execução. Foi configurado como 10^{-9}.

11.1.1 Cavidade com tampa deslizante

O caso da cavidade com tampa deslizante é dado por Shih, Tan e Hwang (1989) e possui a seguinte solução (representada na Figura 24):

$$u(x,y) = 8q(x)r'(y) = 8(x^4 - 2x^3 + x^2)(4y^3 - 2y) \tag{11.3}$$

$$v(x,y) = -8q'(x)r(y) = -8(4x^3 - 6x^2 + 2x)(y^4 - y^2) \tag{11.4}$$

$$p(x,y,Re) = \frac{8}{Re}[Q(x)r'''(y) + q'(x)r'(y)] + 64Q_2(x)\{r(y)r''(y) - [r'(y)]^2\}, \tag{11.5}$$

onde

$$q(x) = x^4 - 2x^3 + x^2 \tag{11.6}$$

$$r(y) = y^4 - y^2 \tag{11.7}$$

$$Q(x) = \int q(x)dx = 0,2x^5 - 0.5x^4 + x^3/3 \tag{11.8}$$

$$Q_1(x) = q(x)q''(x) - [q'(x)]^2 = -4x^6 + 12x^5 - 14x^4 + 8x^3 - 2x^2 \tag{11.9}$$

$$Q_2(x) = \int q(x)q'(x)dx = 0,5[q(x)]^2 \tag{11.10}$$

$$R_1(y) = r(y)r'''(y) - r'(y)r''(y) = -24y^5 + 8y^3 - 4y. \tag{11.11}$$

O termo fonte é dado por:

$$\varphi_x = 0 \tag{11.12}$$

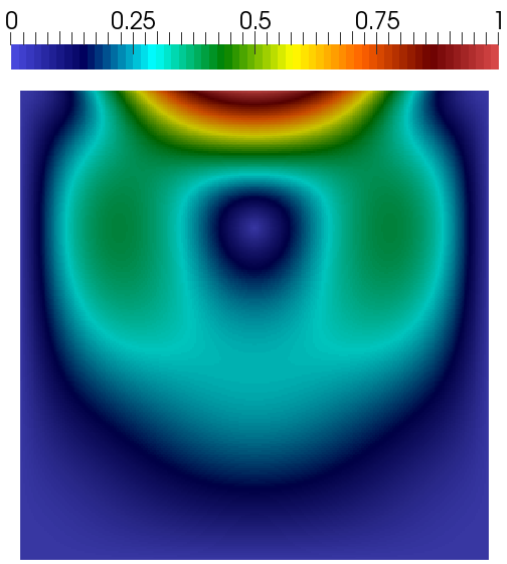
$$\begin{aligned}\varphi_y(x,y,Re) = {} & \frac{8}{Re}[24Q(x) + 2q'(x)r''(y) + q'''(x)r(y)] \\ & + 64[Q_2(x)R_1(y) - r(y)r'(y)Q_1(x)]\end{aligned} \tag{11.13}$$

O caso foi configurado com $Re = 200$, esquema advectivo CDS e condições de contorno de Dirichlet para velocidade em todas as direções. Nesta situação, onde não há condição de Dirichlet para a pressão, o Cyberex automaticamente impõe o valor da pressão para um dos pontos do domínio, de modo a dessingularizar o sistema de equações. Em casos como este, onde solução exata é conhecida, o valor de p para o ponto é usado. Nos casos onde o valor não é conhecido, zero é usado.

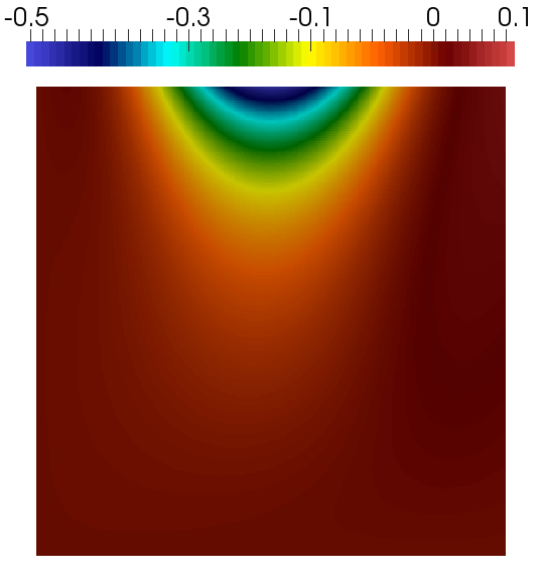
A norma do erro e a ordem de convergência calculada entre malhas progressivamente mais finas são dadas nas Tabelas 6 e 7. A ordem do erro tende a 2 — tanto para velocidade quanto para pressão — a medida que a malha é refinada. Este é o resultado esperado para

Figura 24 – Solução em regime permanente da cavidades 2D com solução analítica.

(a) Magnitude da velocidade.



(b) Pressão.



	Norma L_∞ do erro			Razão			Ordem		
Malha	p	u_1	u_2	p	u_1	u_2	p	u_1	u_2
32×32	3,84e-3	6,70e-3	6,14e-3	—	—	—	—	—	—
64×64	1,05e-3	1,80e-3	1,62e-3	3,65	3,73	3,79	1,87	1,90	1,92
128×128	2,72e-4	4,56e-4	4,13e-4	3,87	3,94	3,93	1,95	1,98	1,97
256×256	6,90e-5	1,14e-4	1,04e-4	3,94	3,99	3,99	1,98	2,00	1,99
512×512	1,74e-5	2,86e-5	2,59e-5	3,97	4,00	4,00	1,99	2,00	2,00

Tabela 6 – Progressão da norma L_∞ do erro na cavidade com tampa deslizante.

	Norma L_2 do erro			Razão			Ordem		
Malha	p	u_1	u_2	p	u_1	u_2	p	u_1	u_2
32×32	6,52e-4	1,42e-3	1,88e-3	—	—	—	—	—	—
64×64	1,72e-4	3,68e-4	4,95e-4	3,80	3,85	3,80	1,93	1,95	1,93
128×128	4,36e-5	9,30e-5	1,26e-4	3,94	3,95	3,94	1,98	1,98	1,98
256×256	1,09e-5	2,33e-5	3,16e-5	3,99	3,99	3,98	2,00	2,00	1,99
512×512	2,73e-6	5,83e-6	7,90e-6	4,00	4,00	4,00	2,00	2,00	2,00

Tabela 7 – Progressão da norma L_2 do erro na cavidade com tampa deslizante.

soluções monolíticas, que ao contrário de métodos segregados, garantem segunda ordem também para a pressão, e não só para a velocidade. Esse resultado atesta pela viabilidade e correta implementação do método de Newton-Raphson e do esquema advectivo CDS, além dos mecanismos já implementados na HigTree para imposição de condição de contorno de Dirichlet e interpolação, que é usada na discretização mesmo em malha uniforme.

11.1.2 *Inflow* e *outflow* ortogonais

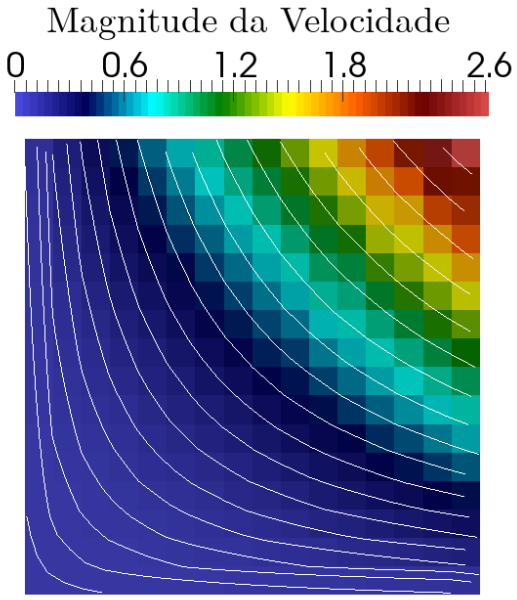
Outro caso manufaturado testado, também dado por Shih, Tan e Hwang (1989), é um domínio quadrado com *inflow* no topo e saída de massa à esquerda (veja as linhas de corrente na Figura 25). A solução é dada por:

$$u(x, y) = 2x^2y \tag{11.14}$$
$$v(x, y) = -2xy^2 \tag{11.15}$$
$$p(x, y) = x^2y^2 \tag{11.16}$$

e o termo fonte é:

$$\varphi_x(x, y, Re) = 4x^3y^2 + 2xy^2 - 4y/Re \tag{11.17}$$
$$\varphi_y(x, y, Re) = 4x^2y^3 + 2x^2y + 4x/Re \tag{11.18}$$

Figura 25 – Campo de velocidade e linhas de corrente para caso analítico com *inflow* e *outflow*.



O caso foi executado a $Re = 200$. Como condição de contorno, foi usado Dirichlet para as velocidades no contorno superior, onde há entrada de massa, e para as faces inferior e esquerda do domínio, onde não há fluxo de massa. Na face direita, foi utilizada Neumann

para as velocidades, onde há saída de massa. Os valores impostos são calculados a partir da solução conhecida.

Esta execução foi feita para testar uma técnica comum na simulação de escoamentos, que é utilizar uma condição de Neumann nas saídas de massa do domínio. Nelas, o valor das velocidades é parte da solução, e portanto, a princípio é desconhecido. Nessa situação, é comum impor derivada nula no contorno.

	Norma L_∞ do erro			Razão			Ordem		
Malha	p	u_1	u_2	p	u_1	u_2	p	u_1	u_2
16×16	8,26e-3	1,52e-2	1,73e-2	—	—	—	—	—	—
32×32	2,19e-3	3,60e-3	3,67e-3	3,78	4,24	4,73	1,92	2,08	2,24
64×64	5,67e-4	8,71e-4	8,72e-4	3,86	4,13	4,21	1,95	2,05	2,07
128×128	1,44e-4	2,15e-4	2,13e-4	3,93	4,05	4,09	1,97	2,02	2,03
256×256	3,64e-5	5,33e-5	5,27e-5	3,96	4,03	4,04	1,99	2,01	2,02
512×512	9,16e-6	1,33e-5	1,31e-5	3,98	4,01	4,02	1,99	2,01	2,01

Tabela 8 – Progressão da norma L_∞ do erro no caso com *inflow* e *outflow*.

	Norma L_2 do erro			Razão			Ordem		
Malha	p	u_1	u_2	p	u_1	u_2	p	u_1	u_2
16×16	2,70e-3	3,08e-3	3,75e-3	—	—	—	—	—	—
32×32	6,85e-4	6,61e-4	8,39e-4	3,95	4,67	4,47	1,98	2,22	2,16
64×64	1,74e-4	1,55e-4	2,02e-4	3,94	4,27	4,16	1,98	2,10	2,06
128×128	4,38e-5	3,74e-5	4,95e-5	3,97	4,13	4,07	1,99	2,05	2,03
256×256	1,10e-5	9,19e-6	1,23e-5	3,98	4,07	4,03	1,99	2,02	2,01
512×512	2,75e-6	2,28e-6	3,06e-6	3,99	4,03	4,02	2,00	2,01	2,01

Tabela 9 – Progressão da norma L_2 do erro no caso com *inflow* e *outflow*.

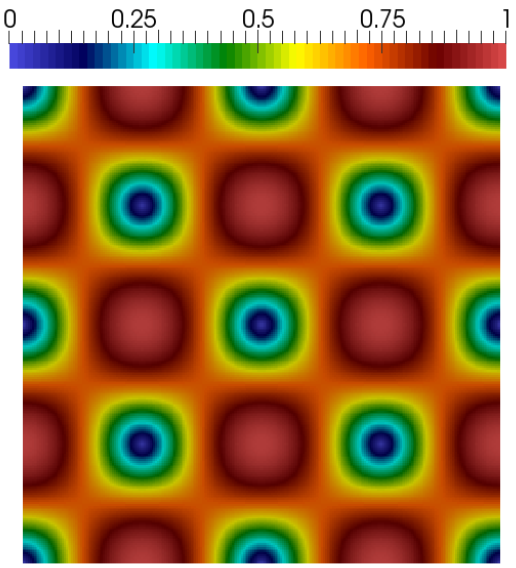
Como mostrado nas Tabelas 8 e 9, o programa consegue executar corretamente casos com condição de Neumann na saída de massa e obter ordem tendendo a dois. É interessante notar que a ordem tende a dois por baixo para a pressão, e por cima para as velocidades. A razão deste comportamento é desconhecida.

11.1.3 Taylor-Green permanente

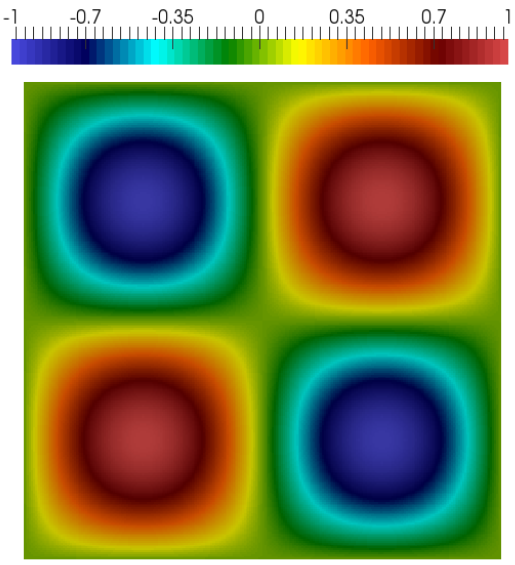
Outra solução analítica testada foi uma versão adaptada para regime permanente dos vórtices de Taylor-Green. Foram executados testes de independência de malha, onde se verificou a ordem 2 tanto para pressão quanto para velocidade.

Figura 26 – Solução em regime permanente dos vórtices de Taylor-Green.

(a) Magnitude da velocidade.



(b) Pressão.



A solução, mostrada na Figura 26, é dada por:

$$u(x,y) = \operatorname{sen}(2x\pi)\cos(2y\pi) \tag{11.19}$$

$$v(x,y) = -\cos(2x\pi)\operatorname{sen}(2y\pi) \tag{11.20}$$

$$p(x,y) = \operatorname{sen}(2x\pi)\operatorname{sen}(2y\pi), \tag{11.21}$$

sendo que os termos fontes incluídos nas equações da quantidade de movimento são:

$$\begin{aligned}\varphi_x(x,y,Re) = {} & 2\pi\cos(2x\pi)\operatorname{sen}(2x\pi)\operatorname{sen}^2(2y\pi) \\ & + 2\pi\cos(2x\pi)\operatorname{sen}(2y\pi) \\ & + 2\pi\cos(2x\pi)\operatorname{sen}(2x\pi)\cos^2(2y\pi) \\ & + \frac{8\pi^2\operatorname{sen}(2x\pi)\cos(2y\pi)}{Re}\end{aligned} \tag{11.22}$$

e

$$\begin{aligned}\varphi_y(x,y,Re) = {} & 2\pi\cos(2y\pi)\operatorname{sen}(2y\pi)\operatorname{sen}^2(2x\pi) \\ & + 2\pi\operatorname{sen}(2x\pi)\cos(2y\pi) \\ & + 2\pi\cos(2y\pi)\operatorname{sen}(2y\pi)\cos^2(2x\pi) \\ & - \frac{8\pi^2\cos(2x\pi)\operatorname{sen}(2y\pi)}{Re}.\end{aligned} \tag{11.23}$$

Quatro situações foram verificadas com esta solução:

- malha uniforme com condições de contorno periódicas em todas as direções;

- malha refinada também com condições de contorno periódicas;
- malha uniforme com condições de contorno de Dirichlet para velocidades e Neumann para pressão nos contornos em uma direção, e o inverso nos contornos da direção transversal;
- malha uniforme com todas as condições de contorno de Dirichlet para velocidades.

Nesta último caso, foi feita uma execução com cada um dos diferentes esquemas advectivos implementados. O Reynolds utilizado nestas execuções foi de $Re = 1$.

11.1.3.1 Execução em malha uniforme e periodicidade

A execução de Taylor-Green mais pura, em termos de número de interpolações, utilizou uma malha uniforme com domínio periódico em todas as direções. Por não ter refinamento, não possui interpolação entre níveis de malha, e por ser periódica, não utiliza interpolação na imposição das condições de contorno. A interpolação é usada somente para se obter as velocidades advectantes, no centro das células, e as componentes das velocidades paralelas às faces, onde são necessárias no esquema advectivo e no termo difusivo.

	Norma L_∞ do erro			Razão			Ordem		
Malha	p	u_1	u_2	p	u_1	u_2	p	u_1	u_2
32×32	1,13e-2	3,59e-3	3,24e-3	—	—	—	—	—	—
64×64	2,81e-3	8,29e-4	8,33e-4	4,03	4,33	3,89	2,01	2,11	1,96
128×128	7,06e-4	2,03e-4	2,03e-4	3,99	4,09	4,11	2,00	2,03	2,04
256×256	1,76e-4	5,05e-5	5,06e-5	4,01	4,01	4,01	2,00	2,00	2,00
512×512	4,40e-5	1,26e-5	1,26e-5	4,01	4,01	4,01	2,00	2,00	2,00

Tabela 10 – Progressão da norma L_∞ do erro em Taylor-Green uniforme e periódico.

	Norma L_2 do erro			Razão			Ordem		
Malha	p	u_1	u_2	p	u_1	u_2	p	u_1	u_2
32×32	5,00e-3	1,66e-3	1,61e-3	—	—	—	—	—	—
64×64	1,23e-3	4,03e-4	4,03e-4	4,07	4,12	4,00	2,02	2,04	2,00
128×128	3,06e-4	1,00e-4	1,00e-4	4,01	4,01	4,01	2,00	2,00	2,00
256×256	7,65e-5	2,51e-5	2,51e-5	4,01	4,00	4,00	2,00	2,00	2,00
512×512	1,91e-5	6,28e-6	6,28e-6	4,00	4,00	4,00	2,00	2,00	2,00

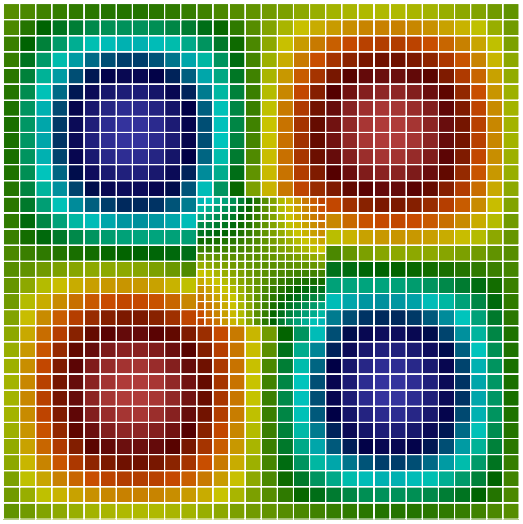
Tabela 11 – Progressão da norma L_2 do erro em Taylor-Green uniforme e periódico.

Conforme mostrado nas Tabelas 10 e 11, esta execução apresentou a ordem de convergência mais próxima do valor teórico permitido pelo método numérico, provavelmente por fazer uso uniforme e simétrico de interpolações.

11.1.3.2 Execução em malha refinada e periodicidade

Ao adicionar um refinamento local no caso com domínio periódico, é possível ver na magnitude do erro e na ordem de convergência a influência das interpolações introduzidas. A região refinada é um quadrado de lado $0{,}25L$ localizado no centro do domínio, conforme a Figura 27. O tamanho da malha dado nas Tabelas 12 e 13 se refere ao número de elementos na malha base, ou seja, nos elementos de malha que são filhos imediados da raiz da árvore do domínio. Na região refinada, cada elemento base é dividido em quatro.

Figura 27 – Refinamento local de malha usado em uma das soluções de Taylor-Green.



	Norma L_∞ do erro			Razão			Ordem		
Malha base	p	u_1	u_2	p	u_1	u_2	p	u_1	u_2
32×32	1,11e-1	3,00e-3	4,04e-3	—	—	—	—	—	—
64×64	3,09e-2	9,62e-4	1,04e-3	3,59	3,11	3,88	1,84	1,64	1,95
128×128	7,75e-3	2,32e-4	2,47e-4	3,98	4,15	4,23	1,99	2,05	2,08
256×256	1,94e-3	5,68e-5	6,01e-5	3,99	4,08	4,10	2,00	2,03	2,04

Tabela 12 – Progressão da norma L_∞ do erro em Taylor-Green com refinamento.

	Norma L_2 do erro			Razão			Ordem		
Malha base	p	u_1	u_2	p	u_1	u_2	p	u_1	u_2
32×32	2,75e-2	1,17e-3	1,80e-3	—	—	—	—	—	—
64×64	7,59e-3	3,55e-4	3,86e-4	3,63	3,30	4,66	1,86	1,72	2,22
128×128	1,88e-3	8,24e-5	8,69e-5	4,04	4,31	4,44	2,01	2,11	2,15
256×256	4,69e-4	1,99e-5	2,08e-5	4,01	4,14	4,17	2,01	2,05	2,06

Tabela 13 – Progressão da norma L_2 do erro em Taylor-Green com refinamento.

Olhando os resultados nas Tabelas 12 e 13, vê-se que a ordem de convergência é perturbada pelo refinamento local, especialmente nas malhas mais grosseiras, porém ela tende à ordem teórica de 2 a medida que as malhas são refinadas. Mesmo chegando a ordem 2, a presença de um bloco de refinamento local aumenta a norma do erro em uma ordem de magnitude, se comparado com a mesma malha base nas execuções sem o refinamento local — dada nas Tabelas 10 e 11.

11.1.3.3 Execução com condições de contorno mistas

Outro teste com condições de contorno diferentes para uma mesma variável também foi executado usando Taylor-Green em malha uniforme. Nesta execução, à esquerda e à direita foram usadas condições de contorno de Dirichlet para as velocidades e Neumann para a pressão. Em cima e na base, o contrário: condição de Neumann para as velocidades e Dirichlet para a pressão.

	Norma L_∞ do erro			Razão			Ordem		
Malha	p	u_1	u_2	p	u_1	u_2	p	u_1	u_2
16×16	1,42e-1	1,69e-2	1,39e-2	—	—	—	—	—	—
32×32	3,84e-2	3,81e-3	2,15e-3	3,70	4,44	6,46	1,89	2,15	2,69
64×64	1,06e-2	9,21e-4	5,33e-4	3,63	4,14	4,02	1,86	2,05	2,01
128×128	2,89e-3	2,28e-4	1,34e-4	3,66	4,04	3,99	1,87	2,01	2,00
256×256	7,77e-4	5,68e-5	3,34e-5	3,72	4,01	4,00	1,90	2,00	2,00

Tabela 14 – Progressão da norma L_∞ do erro em Taylor-Green com condições de contorno mistas.

	Norma L_2 do erro			Razão			Ordem		
Malha	p	u_1	u_2	p	u_1	u_2	p	u_1	u_2
16×16	4,27e-2	6,48e-3	4,35e-3	—	—	—	—	—	—
32×32	1,04e-2	1,50e-3	9,87e-4	4,10	4,32	4,41	2,03	2,11	2,14
64×64	2,57e-3	3,64e-4	2,40e-4	4,05	4,12	4,11	2,02	2,04	2,04
128×128	6,39e-4	9,03e-5	5,95e-5	4,02	4,03	4,04	2,01	2,01	2,01
256×256	1,59e-4	2,25e-5	1,48e-5	4,01	4,01	4,02	2,00	2,00	2,01

Tabela 15 – Progressão da norma L_2 do erro em Taylor-Green com condições de contorno mistas.

Os resultados para a norma L_∞, dados na Tabelas 14, foram similares aos resultados da execução com *inflow* e *outflow* ortogonais, que como este, também utilizou condição de Neumann para as velocidades e condição de Dirichlet para pressão em ao menos uma face do domínio. Similarmente aos resultados da normal L_∞ daquela execução — dados na Tabela 8 — a ordem tende a 2 por baixo para a pressão, e a 2 por cima para as velocidades. Já a norma L_2 tende a 2 por cima em todas as malhas, como mostrado na Tabela 15.

11.1.3.4 Execuções com vários esquemas advectivos

Para testar os esquemas advectivos, também foi utilizada a solução de Taylor-Green em malha uniforme, com condição de Dirichlet para as velocidades em todo o contorno. Os testes de independência de malha foram executados para todos os esquemas advectivos implementados: CDS (Tabelas 16 e 17), 2UP (Tabelas 18 e 19), CUBISTA (Tabelas 20 e 21) e CLAM (Tabelas 22 e 23).

	Norma L_∞ do erro			Razão			Ordem		
Malha	p	u_1	u_2	p	u_1	u_2	p	u_1	u_2
16×16	1,59e-1	7,91e-3	7,89e-3	—	—	—	—	—	—
32×32	4,36e-2	2,02e-3	2,02e-3	3,65	3,91	3,91	1,87	1,97	1,97
64×64	1,13e-2	5,07e-4	5,06e-4	3,84	3,99	3,99	1,94	2,00	2,00
128×128	2,89e-3	1,27e-4	1,27e-4	3,92	3,99	3,99	1,97	2,00	2,00
256×256	7,30e-4	3,18e-5	3,17e-5	3,96	4,00	4,00	1,99	2,00	2,00

Tabela 16 – Progressão da norma L_∞ do erro em Taylor-Green usando CDS.

	Norma L_2 do erro			Razão			Ordem		
Malha	p	u_1	u_2	p	u_1	u_2	p	u_1	u_2
16×16	4,84e-2	3,51e-3	3,51e-3	—	—	—	—	—	—
32×32	1,23e-2	8,72e-4	8,71e-4	3,93	4,03	4,03	1,97	2,01	2,01
64×64	3,08e-3	2,18e-4	2,18e-4	3,99	4,00	4,00	2,00	2,00	2,00
128×128	7,71e-4	5,45e-5	5,44e-5	4,00	4,00	4,00	2,00	2,00	2,00
256×256	1,93e-4	1,36e-5	1,36e-5	4,00	4,00	4,00	2,00	2,00	2,00

Tabela 17 – Progressão da norma L_2 do erro em Taylor-Green usando CDS.

	Norma L_∞ do erro			Razão			Ordem		
Malha	p	u_1	u_2	p	u_1	u_2	p	u_1	u_2
16×16	1,50e-1	7,35e-3	7,32e-3	—	—	—	—	—	—
32×32	4,25e-2	1,95e-3	1,95e-3	3,52	3,76	3,76	1,81	1,91	1,91
64×64	1,12e-2	4,99e-4	4,97e-4	3,79	3,92	3,92	1,92	1,97	1,97
128×128	2,88e-3	1,26e-4	1,25e-4	3,90	3,96	3,96	1,96	1,99	1,99
256×256	7,28e-4	3,16e-5	3,15e-5	3,95	3,98	3,98	1,98	1,99	1,99

Tabela 18 – Progressão da norma L_∞ do erro em Taylor-Green usando 2UP.

Todos os esquemas advectivos funcionaram conforme o esperado, apresentando resultados parecidos: tendem para um erro de ordem 2 a medida que a malha é refinada, sendo o pior resultado a norma L_∞ da pressão na transição da malha 16×16 para 32×32.

	Norma L_2 do erro			Razão			Ordem		
Malha	p	u_1	u_2	p	u_1	u_2	p	u_1	u_2
16×16	7,05e-2	3,26e-3	3,25e-3	—	—	—	—	—	—
32×32	1,84e-2	8,40e-4	8,39e-4	3,84	3,88	3,88	1,94	1,96	1,95
64×64	4,64e-3	2,14e-4	2,14e-4	3,96	3,93	3,93	1,99	1,97	1,97
128×128	1,16e-3	5,40e-5	5,39e-5	3,99	3,96	3,96	2,00	1,99	1,99
256×256	2,91e-4	1,36e-5	1,35e-5	3,99	3,98	3,98	2,00	1,99	1,99

Tabela 19 – Progressão da norma L_2 do erro em Taylor-Green usando 2UP.

	Norma L_∞ do erro			Razão			Ordem		
Malha	p	u_1	u_2	p	u_1	u_2	p	u_1	u_2
16×16	1,53e-1	7,45e-3	7,45e-3	—	—	—	—	—	—
32×32	4,32e-2	1,98e-3	1,97e-3	3,54	3,76	3,78	1,83	1,91	1,92
64×64	1,13e-2	5,03e-4	5,02e-4	3,81	3,94	3,92	1,93	1,98	1,97
128×128	2,89e-3	1,27e-4	1,26e-4	3,92	3,97	3,98	1,97	1,99	1,99
256×256	7,29e-4	3,17e-5	3,16e-5	3,96	3,99	3,99	1,99	2,00	2,00

Tabela 20 – Progressão da norma L_∞ do erro em Taylor-Green usando CUBISTA.

	Norma L_2 do erro			Razão			Ordem		
Malha	p	u_1	u_2	p	u_1	u_2	p	u_1	u_2
16×16	5,49e-2	3,34e-3	3,34e-3	—	—	—	—	—	—
32×32	1,40e-2	8,57e-4	8,57e-4	3,91	3,90	3,90	1,97	1,96	1,96
64×64	3,46e-3	2,16e-4	2,16e-4	4,05	3,96	3,96	2,02	1,99	1,99
128×128	8,52e-4	5,43e-5	5,43e-5	4,07	3,99	3,99	2,02	1,99	1,99
256×256	2,11e-4	1,36e-5	1,36e-5	4,04	3,99	3,99	2,01	2,00	2,00

Tabela 21 – Progressão da norma L_2 do erro em Taylor-Green usando CUBISTA.

11.2 Degrau descendente laminar 2D

O primeiro caso de validação executado é um degrau descendente laminar em duas dimensões. A definição do problema e os resultados de experimentação material utilizados na comparação são dados por Lee e Mateescu (1998). As medições relatadas foram feitas na seção média de um canal com $40L$ de largura, o que largo o suficiente para que o escoamento no meio do canal seja corretamente aproximado por uma simulação numérica 2D. O número de Reynolds do escoamento é 402,5, e as dimensões de sua geometria são dadas pela Figura 28.

A malha é composta por duas árvores retangulares: a seção de entrada, a montante do degrau, e o canal, a jusante. Possui três níveis de refinamento, na razão 1/2 por direção (ou 1/4 por elemento) entre níveis. Os elementos são quadrados, sendo que, no nível mais grosso, possui $\Delta x_i = 0{,}125L$. O nível mais fino se inicia a $0{,}25L$ antes do degrau, se estende ao longo de toda a seção de testes, termina em $40L$ a partir do degrau e ocupa toda a

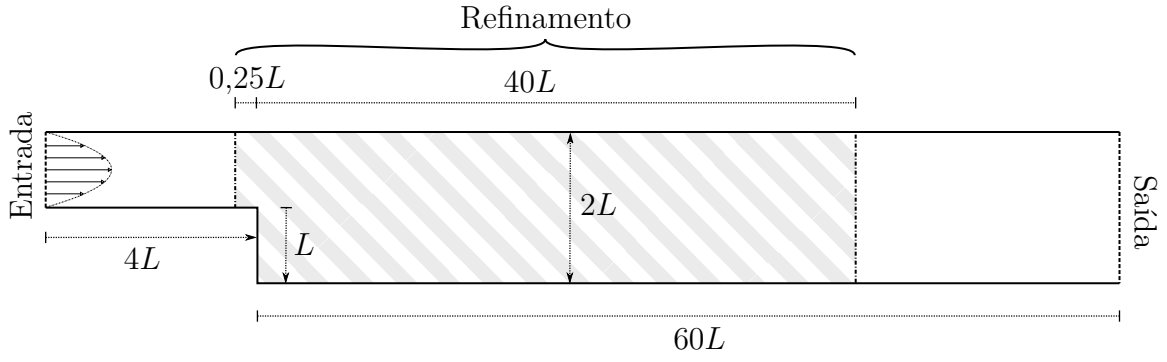
	Norma L_∞ do erro			Razão			Ordem		
Malha	p	u_1	u_2	p	u_1	u_2	p	u_1	u_2
16×16	1,50e-1	7,34e-3	7,33e-3	—	—	—	—	—	—
32×32	4,25e-2	1,95e-3	1,93e-3	3,54	3,76	3,79	1,82	1,91	1,92
64×64	1,12e-2	4,98e-4	4,95e-4	3,79	3,92	3,91	1,92	1,97	1,97
128×128	2,87e-3	1,26e-4	1,25e-4	3,90	3,96	3,95	1,96	1,98	1,98
256×256	7,27e-4	3,16e-5	3,15e-5	3,95	3,98	3,98	1,98	1,99	1,99

Tabela 22 – Progressão da norma L_∞ do erro em Taylor-Green usando CLAM.

	Norma L_2 do erro			Razão			Ordem		
Malha	p	u_1	u_2	p	u_1	u_2	p	u_1	u_2
16×16	5,55e-2	3,28e-3	3,28e-3	—	—	—	—	—	—
32×32	1,47e-2	8,41e-4	8,41e-4	3,77	3,90	3,90	1,92	1,96	1,97
64×64	3,72e-3	2,14e-4	2,14e-4	3,96	3,93	3,93	1,98	1,98	1,98
128×128	9,30e-4	5,39e-5	5,39e-5	4,00	3,96	3,96	2,00	1,99	1,99
256×256	2,32e-4	1,36e-5	1,35e-5	4,00	3,98	3,98	2,00	1,99	1,99

Tabela 23 – Progressão da norma L_2 do erro em Taylor-Green usando CLAM.

Figura 28 – Diagrama esquemático do degrau laminar 2D, fora de escala.

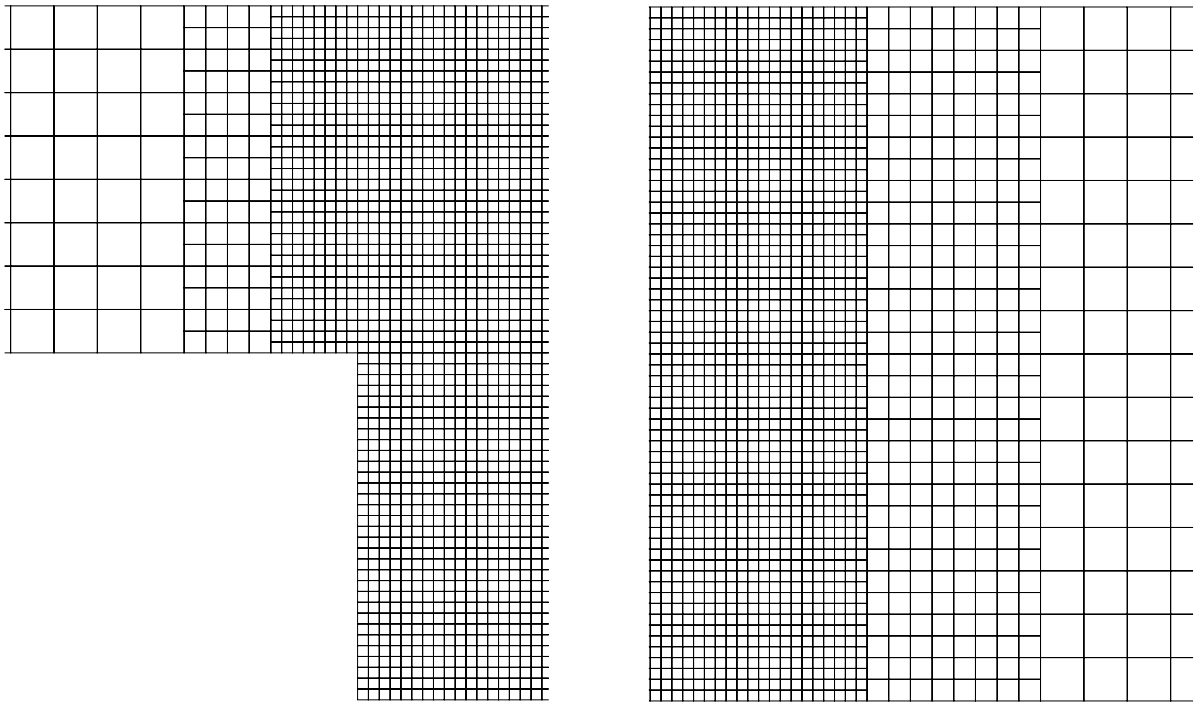


altura do domínio. O nível intermediário é usado somente na área de transição entre a malha mais grossa e a mais fina, como mostrado na Figura 29.

Para o campo de pressão é imposta uma condição de Neumann com derivada nula em todo o contorno do domínio, exceto na saída, onde o valor da pressão é imposto como zero em uma condição de Dirichlet. Para os campos de velocidade é usada uma condição de Neumann com derivada nula na saída e condição de Dirichlet em todo o resto do contorno. O valor das velocidades é imposto como zero nas paredes, que são as arestas superior, inferior e degrau. Na entrada, o valor é definido como a solução analítica para um escoamento desenvolvido em um canal, tal que sua velocidade média seja $1U$. Considerando

Figura 29 – Transição do refinamento na malha do degrau laminar 2D.

(a) A montante do degrau. (b) A jusante do degrau.



que a entrada é a linha em $x = 0$ e $1 < y < 2$, o valor na entrada é dado pela parábola:

$$u(0, y) = -6(y - 1)(y - 2) \tag{11.24}$$

$$v(0, y) = 0. \tag{11.25}$$

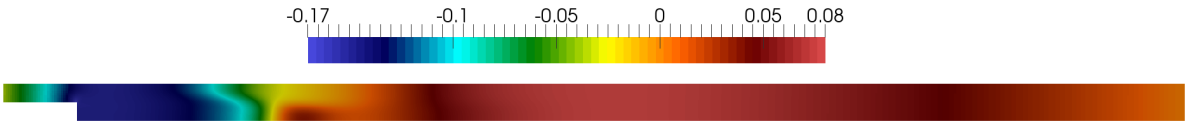
A simulação foi executada em paralelo em um computador de mesa, utilizando as quatro unidades de processamento disponíveis, até atingir o regime permanente com o método de Newton-Raphson. O parâmetros utilizados na execução foram:

- esquema advectivo: CUBISTA;
- tolerância da norma do resíduo não linear: 10^{-6};
- tolerância relativa do *solver* linear: 10^{-2};
- Ac inicial: 0,2;
- Ac mínimo: 0,05;
- Ac máximo: 10^4.

Ao contrário do que ocorrera nos casos de validação, o valor de Ac se estabilizou em torno de 1 (ao invés de aumentar até 10 ou mais), provavelmente devido à dificuldade de se

Figura 30 – Solução numérica em regime estacionário do degrau laminar 2D.

(a) Pressão.



(b) Magnitude da velocidade.

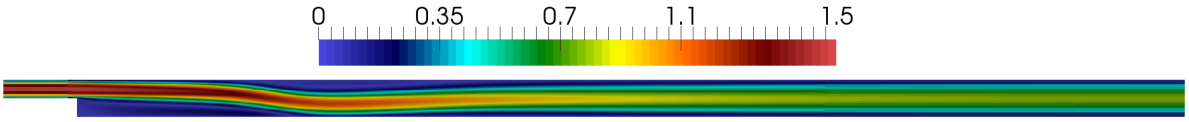


Tabela 24 – Comparação das distâncias de descolamento e recolamento do escoamento laminar por um degrau 2D.

Exp. Numéricos	*Re*	Altura	Rec. Inf.	Desc. Sup.	Rec. Sup.	Tam. Sup.
Presente trabalho	402,5	2	5,86	4,63	10,27	5,64
Lee e Mateescu (1998)	400	2	6,00	4,80	10,30	5,50
Gartling (1990)	400	2	6,10	4,85	10,48	5,63
Kim e Moin (1985)	400	2	6,00	—	—	5,75
Sohn (1988)	400	2	5,80	—	—	4,63
Exp. Materiais	***Re***	**Altura**	**Rec. Inf.**	**Desc. Sup.**	**Rec. Sup.**	**Tam. Sup.**
Lee e Mateescu (1998)	402,5	2	6,45	5,15	10,25	5,10
Armaly et al. (1983)	387,6	2,06	7,00	5,70	10,00	4,30

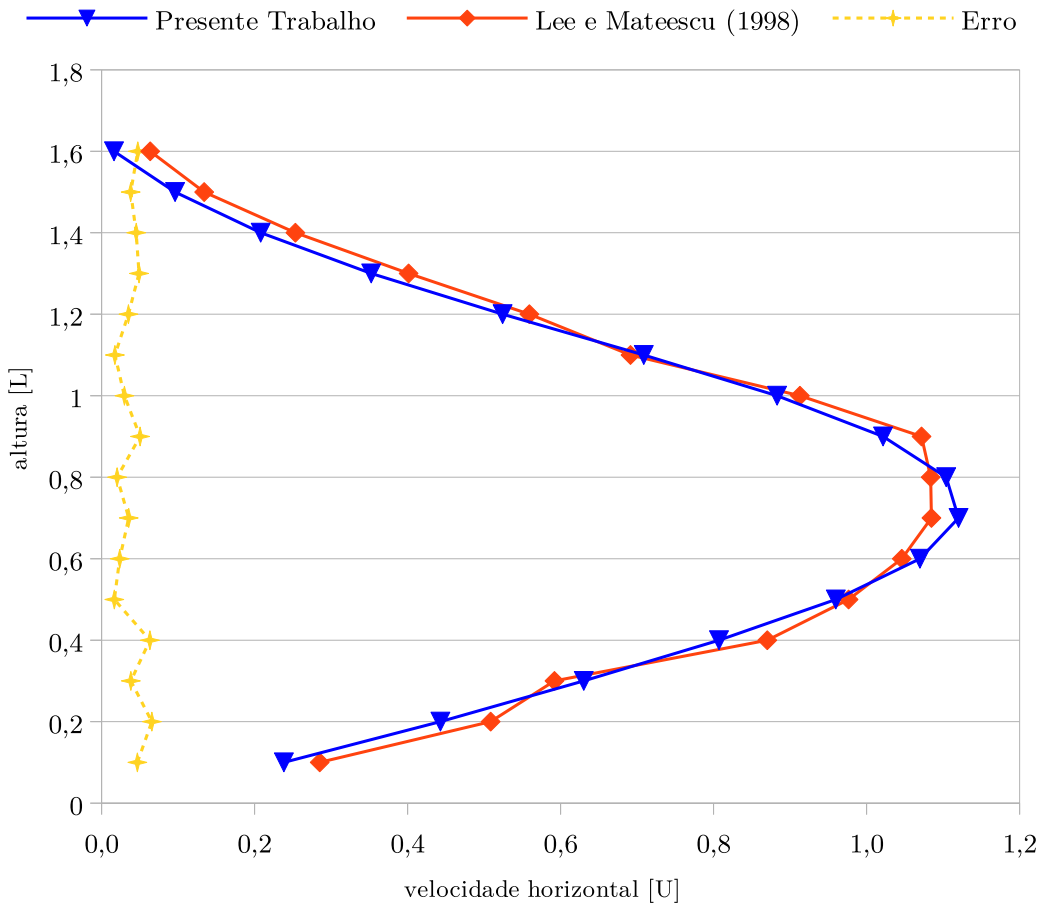
solucionar o sistema linear para um domínio alongado. Um Δt alto aumentava o número de interações necessárias ao *solver* linear para atingir a tolerância, o que impactava na razão $\frac{t_P}{\Delta t}$, penalizando valores mais altos de Ac na otimização do passo de tempo.

A solução final é mostrada na Figura 30. Ela possui duas recirculações características: uma na parte inferior, entre o degrau e base do canal, e outra na parte superior, onde ocorre um descolamento da camada limite. As distâncias de descolamento e recolamento das recirculações calculadas no presente trabalho são dadas, na Tabela 24, em comparação com valores relatados por outros trabalhos na literatura.

Os valores da velocidade u foram amostrados em duas seções do domínio: em $14L$ e $30L$ a partir do degrau, e então comparados com os valores fornecidos pelo artigo de referência (LEE; MATEESCU, 1998). A comparação dos perfis e o erro para as seções $14L$ e $30L$ são dados nas Figuras 31 e 32, respectivamente. O maior erro identificado é de 6,58% de U na seção $14L$, e de 7,41% na seção $30L$.

O grau de concordância dos resultados desta simulação com a literatura atesta a relevância física para um escoamento laminar dos métodos desenvolvidos e implementados

Figura 31 – Comparação dos resultados para as sondas em $14L$ do degrau laminar 2D.



no presente trabalho.

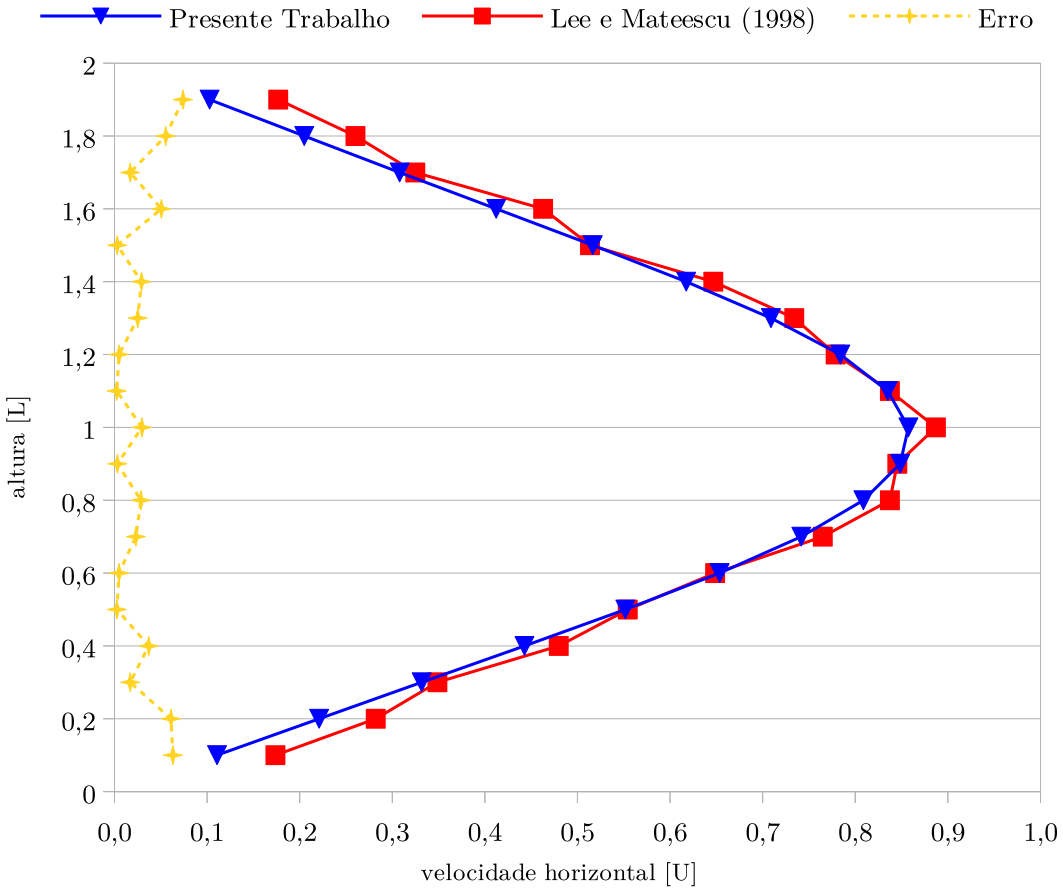
11.3 Degrau descendente turbulento 3D

Outro caso de validação estudado foi o escoamento turbulento 3D em um degrau descendente, conforme especificado por Jovic e Driver (1994) e numericamente estudado por Melo (2017), sendo o número de Reynolds do escoamento simulado igual a 4940,8. Os resultados da simulação foram comparados com a amostragem feita na experimentação material publicada no artigo de referência.

Conforme mostrado na Figura 33, o domínio da simulação foi feito de modo a representar metade do escoamento simétrico do experimento material de referência. O canal simulado tem espessura de $4L$ na direção z, variando de $-2L$ a $2L$ — que é suficiente para o desenvolvimento das estruturas turbilhonares (MELO, 2017 apud NETO et al.; SPODE; CAMPREGHER; NETO, 1993, 2005, p. 108) — sendo periódico nesta direção.

A geometria do degrau é dada pela diferença de altura entre as duas árvores que compõem o domínio, que estão alinhadas pela face superior. Nesta face, é imposta

Figura 32 – Comparação dos resultados para as sondas em $30L$ do degrau laminar 2D.



uma condição de contorno de simetria: Dirichlet com valor zero para a velocidade v, perpendicular à face, e Neumann com derivada nula para a pressão e as outras duas componentes da velocidade. Nas paredes, que delimitam o domínio por baixo, foram impostas condições de não deslizamento. Na saída foi imposta condição Neumann com derivada nula para as componentes da velocidade, e Dirichlet com valor zero para a pressão. Na entrada, foi colocado um perfil de velocidade que interpola linearmente os valores médios dados por Jovic e Driver (1994) a $-3{,}12L$ do degrau. Este perfil está representado em escala na Figura 33.

A malha possui quatro níveis de refinamento, que se estendem ao longo da extensão e profundidade do domínio, variando somente na vertical. O nível mais fino varia da parede inferior até $2L$ de altura, a partir daí a malha engrossa progressivamente até o limite superior do domínio, em $6L$. A células são cubos, que no nível mais grosso possuem arestas de $0{,}5L$. A razão de refinamento entre níveis é de $1/2$ por direção — o que resulta em $1/8$ do volume por elemento. O domínio total possui 1.082.368 elementos. A Figura 34 mostra uma visão frontal do refinamento utilizado.

Por ser um escoamento turbulento, o resultado comparado é uma média temporal do campo de velocidade. Ao se utilizar LES, o resultado obtido na simulação é uma evolução

Figura 33 – Diagrama esquemático do degrau 3D, fora de escala exceto pelo perfil de velocidade de entrada.

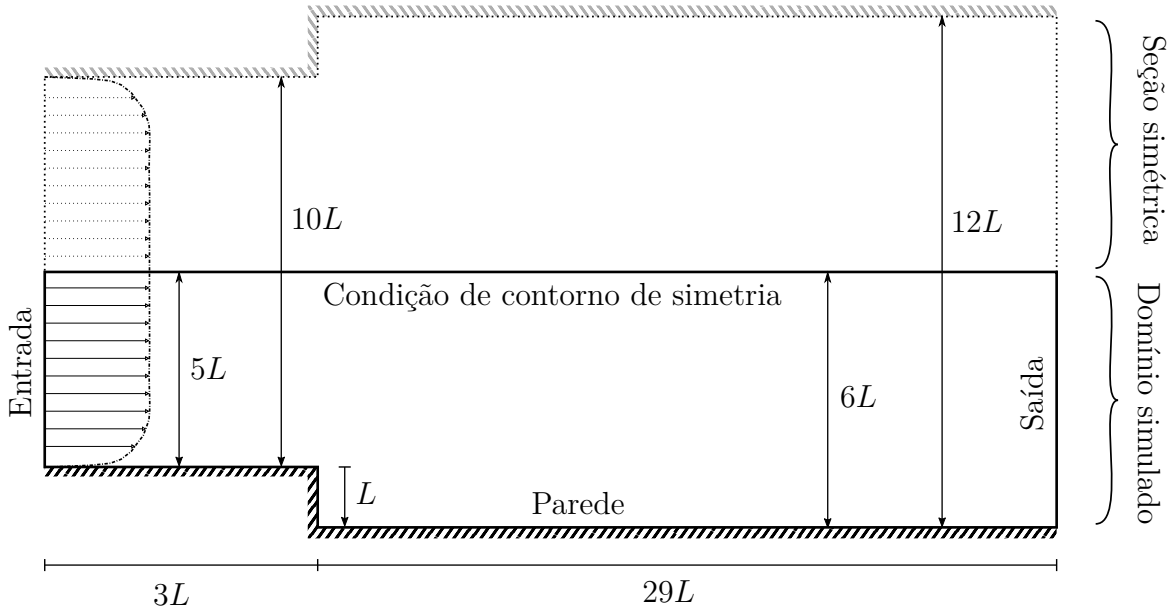
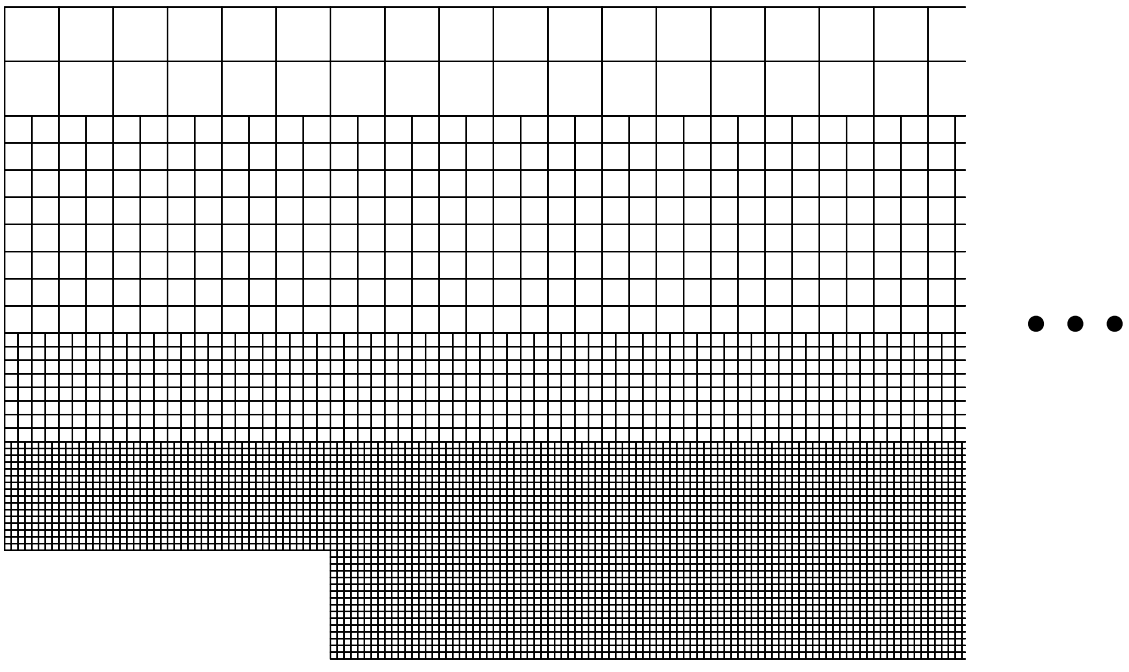


Figura 34 – Refinamento da malha domínio do degrau 3D.



temporal do campo de velocidade que compõe as grandes escalas, e precisa ser fisicamente consistente para que o cálculo da média seja correta. Por esse motivo, o valor de Ac foi limitado a 0,5, de modo a obedecer a condição de CFL. Como não havia necessidade de estabilidade a passos de tempo grandes, foi utilizada a técnica de linearização descrita na subseção 10.1.3 (ao invés do método não linear de Newton-Raphson) e portanto somente um sistema linear foi resolvido por passo de tempo.

Esta simulação foi executada no *cluster* de computação do MFLab, utilizando 16 processos MPI, onde alcançou o tempo de simulação de $t = 279{,}799L/U$, quando terminou o prazo de 1000 horas alocado para a execução desta tarefa. Outros parâmetros utilizados na execução foram:

- esquema advectivo: CUBISTA;
- constante de Smagorinsky: $C_S = 0{,}1$;
- tolerância relativa do *solver* linear: 10^{-6};
- tolerância absoluta do *solver* linear: 10^{-8};
- Ac inicial: 0,05;
- Ac mínimo: 0,025;
- Ac máximo: 0,5.

A simulação atingiu um regime estatisticamente permanente por volta de $t = 150$, como é possível ver no gráfico das componentes da velocidade amostradas em um ponto do domínio ao longo tempo, exibido em duas escalas diferentes na Figura 35. Já as Figuras 37-44 mostram as estruturas turbilhonares desenvolvidas em vários passos de tempo ao longo da simulação, que são consistentes com o escoamento turbulento esperado a este número de Reynolds.

As variáveis da simulação foram amostradas nas seções transversais $4L$, $10L$ e $19L$ a partir do degrau, com os pontos de amostragem dispostos em uma linha vertical em $z = 0$. A média temporal das componentes da velocidade foi calculada com os dados amostrados em $t \geq 150$, e comparados com os resultados relatados no artigo de referência.

Para se determinar o ponto de recolamento, também foi feita a média temporal de uma linha de amostragem ao longo do chão do canal a jusante do degrau, em $z = 0$, determinando a velocidade u média para cada faceta nesta linha. A média em $t \geq 150$ é dada no gráfico da Figura 36. A inversão do sinal da velocidade — que caracteriza o ponto de recolamento — se dá em $8{,}646L$ a partir do degrau. Este valor está fora da faixa relatada por Jovic e Driver (1994), que é de $6L \pm 15\%$.

Figura 35 – Progressão da velocidade do degrau 3D no ponto $x = 4L$ a partir do degrau, $y = 0{,}404L$ a partir do chão e $z = 0$.

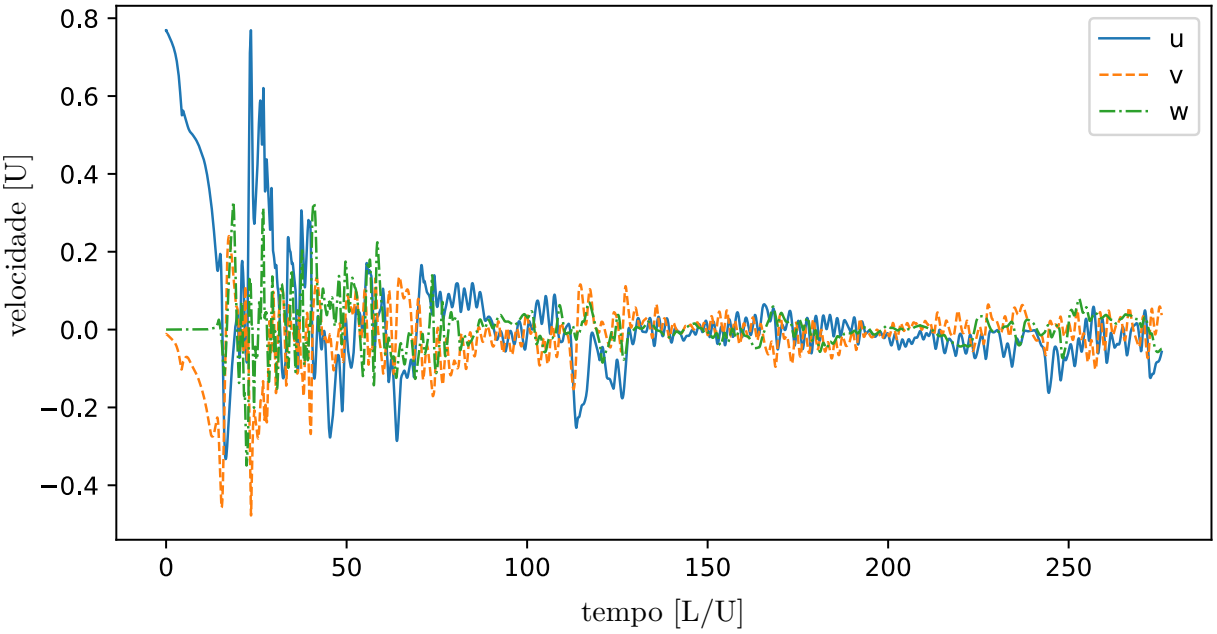


Figura 36 – Média da velocidade ao longo do fundo do canal do degrau 3D.

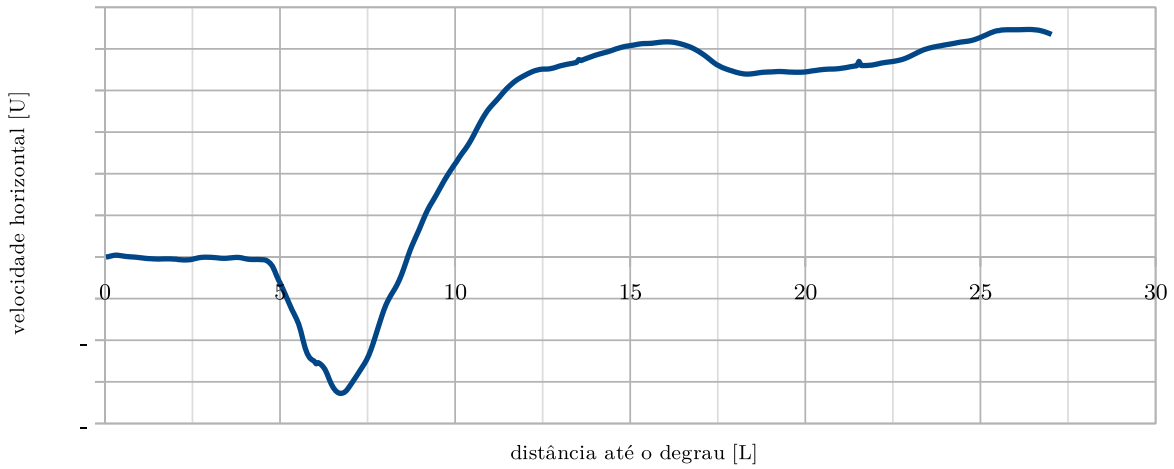


Figura 37 – Iso-Q das estruturas turbilhonares do degrau 3D em $t = 5L/U$.

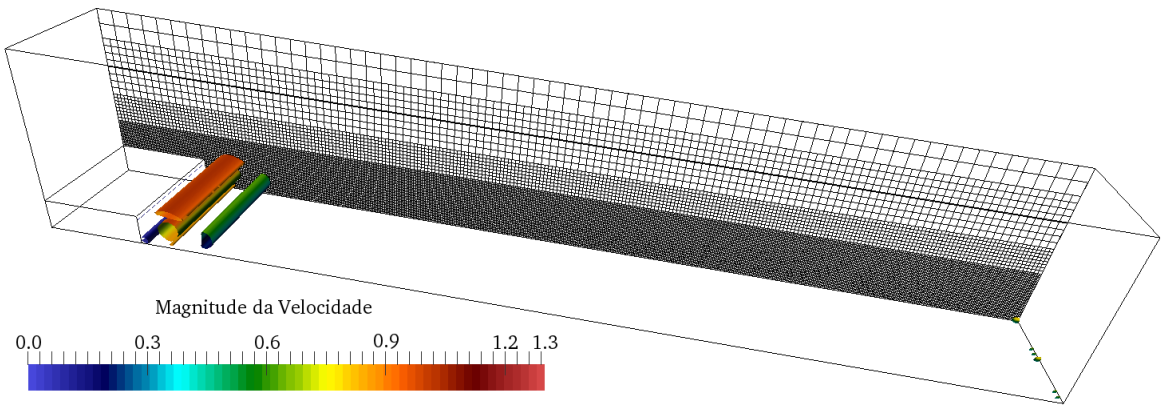


Figura 38 – Superfície iso-Q das estruturas turbilhonares do degrau 3D em $t = 25L/U$.

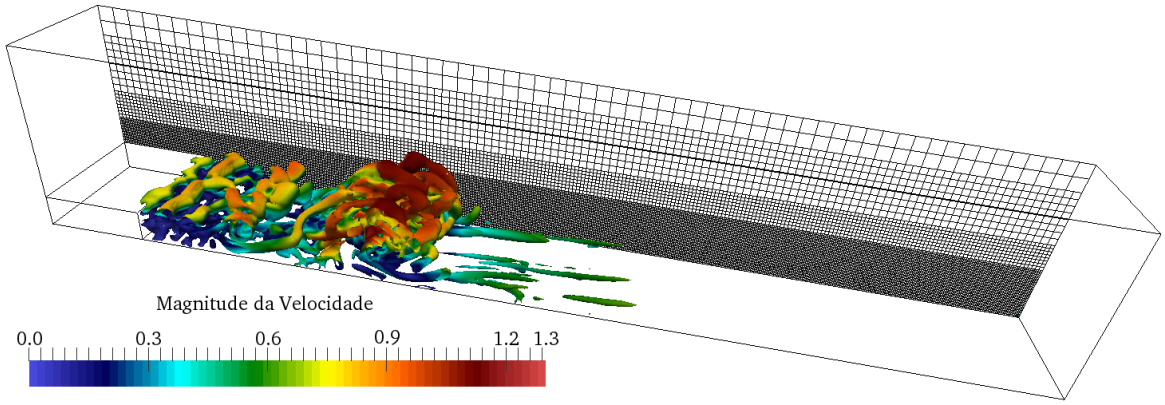


Figura 39 – Superfície iso-Q das estruturas turbilhonares do degrau 3D em $t = 50L/U$.

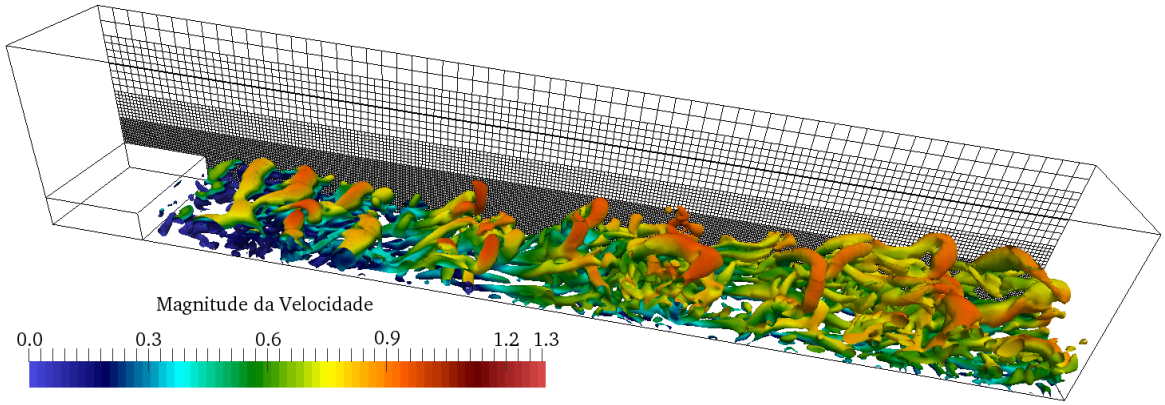


Figura 40 – Superfície iso-Q das estruturas turbilhonares do degrau 3D em $t = 75L/U$.

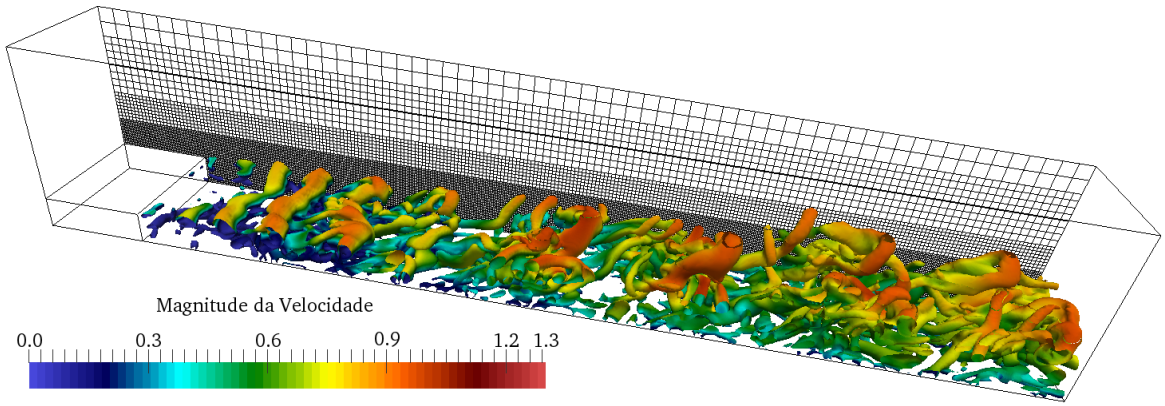


Figura 41 – Superfície iso-Q das estruturas turbilhonares do degrau 3D em $t = 100L/U$.

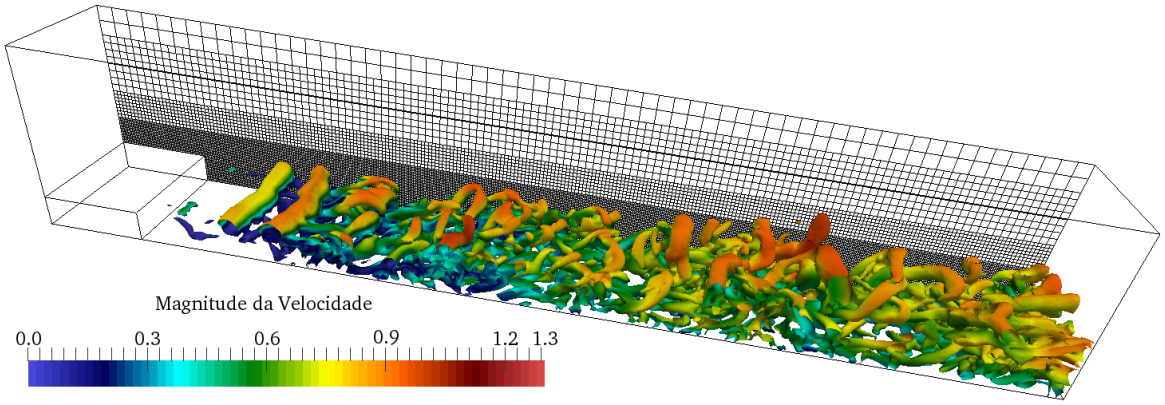


Figura 42 – Superfície iso-Q das estruturas turbilhonares do degrau 3D em $t = 150L/U$.

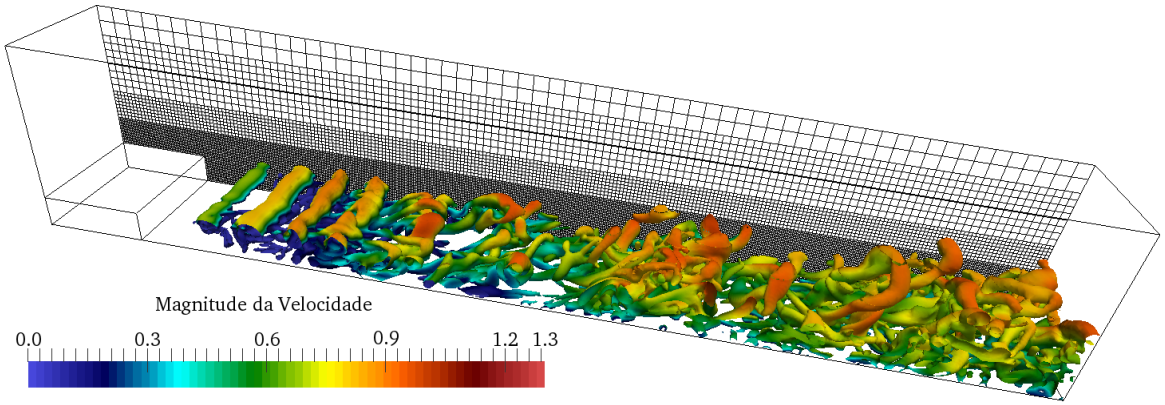


Figura 43 – Superfície iso-Q das estruturas turbilhonares do degrau 3D em $t = 200L/U$.

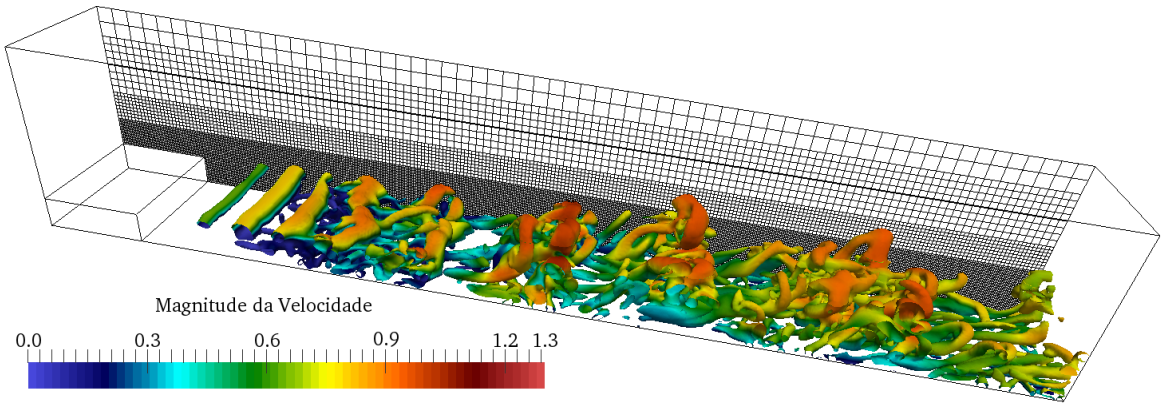


Figura 44 – Superfície iso-Q das estruturas turbilhonares do degrau 3D em $t = 250L/U$.

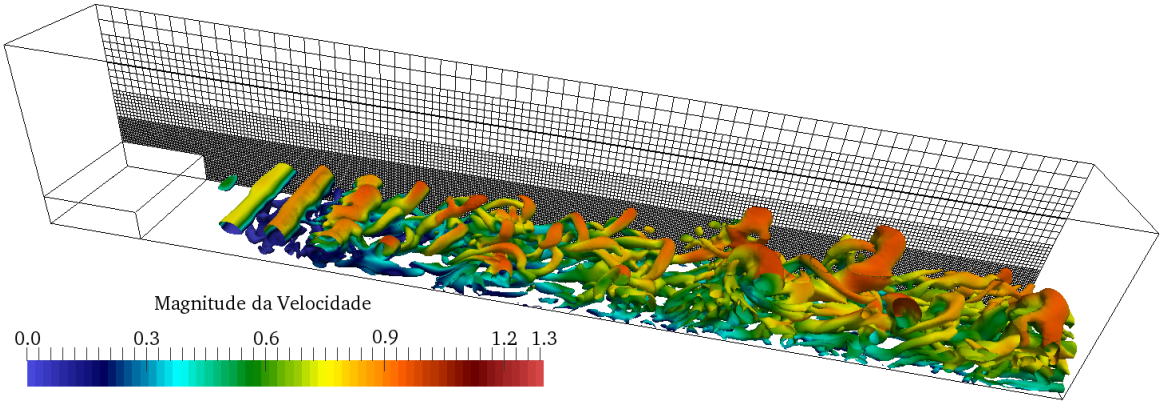
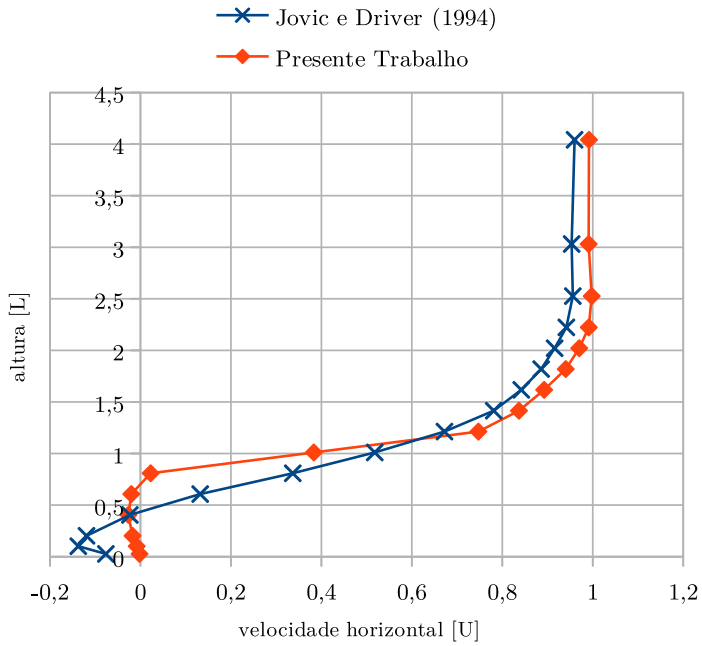


Figura 45 – Perfil médio de velocidade horizontal em $4L$ a partir do degrau 3D.



O comprimento de recolamento ruim se mostra também na comparação da média da velocidade nas sondas em $4L$ (Figura 45), sendo o resultado consistente com o experimento material nas sondas das seções $10L$ (Figura 46) e $19L$ (Figura 47), por serem mais distantes do degrau. O erro observado indica uma dificuldade de transição do escoamento laminar para o regime turbulento, o que pode ser explicado pela falta de ruído na condição de contorno de entrada.

Figura 46 – Perfil médio de velocidade horizontal em $10L$ a partir do degrau 3D.

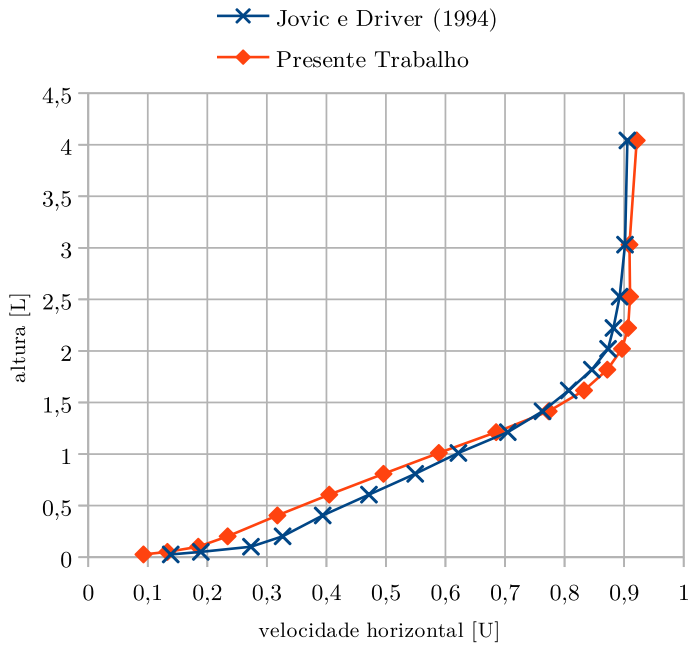
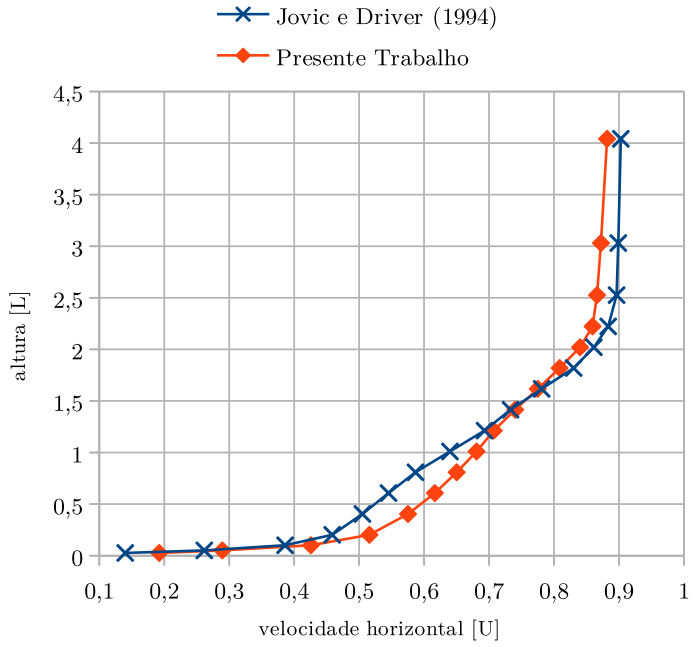


Figura 47 – Perfil médio de velocidade horizontal em $19L$ a partir do degrau 3D.



12 Conclusão

No presente trabalho apresentei minhas contribuições para o desenvolvimento da HigTree, que resultaram em mais flexibilidade e melhor *performance* para aplicações em paralelo. Em cima dela, desenvolvi uma aplicação de CFD para escoamentos newtonianos monofásicos incompressíveis, utilizando uma formulação baseada na solução monolítica das variáveis primárias: pressão e velocidade.

A HigTree ainda está em fase de desenvolvimento, e portanto ainda não provê — apesar de estar previsto em seu plano de desenvolvimento — algumas funcionalidades que seriam úteis em uma aplicação de CFD, como a alteração dinâmica do domínio com refinamentos e engrossamentos, rebalanceamento de carga e escrita e carregamento de dados de maneira independente do particionamento.

Para melhorar sua paralelização, tenho em perspectiva generalizar a estrutura de dados de árvore da HigTree para suportar formatos de malha não retangulares, o que permitira a utilização dos melhores algoritmos de particionamento disponíveis, que promovem uma melhor distribuição da carga de processamento e otimizam a comunicação entre os processos.

Mas mesmo em desenvolvimento, as principais funcionalidades da HigTree já estão prontas e são suficientes para o desenvolvimento de aplicações complexas de CFD. Sua arquitetura orientada a objetos provê abstrações que facilitam o raciocínio sobre o problema e simplificam a implementação, especialmente quanto às interpolações e imposição das condições de contorno. Não foi necessário nenhum tratamento especial ou verificação para impor as condições de contorno no modelo numérico, sendo tudo tratado transparentemente pela HigTree.

Com a aplicação desenvolvida, o Cyberex, foi possível mostrar a viabilidade de uma formulação monolítica para a solução paralela do problema de escoamentos em malha deslocada, aproveitando as funcionalidades e otimizações já desenvolvidas e disponíveis em bibliotecas de *software* modernas.

A abordagem monolítica desenvolvida é diretamente aplicável à discretização em malha deslocada bloco-estruturada do MFSim. Temos a perspectiva de implementá-la nesta ferramenta, onde será utilizada em escoamentos multifásicos, que requerem maior estabilidade numérica ao se trabalhar com as altas razões de propriedades físicas entra as fases do escoamento.

A solução não linear de escoamentos com o método de Newton-Raphson, que é estável com alto Δt, parece ser mais adequado para problemas onde o interesse é por

um resultado em regime permanente, que pode ser alcançado rapidamente se o regime transiente não for interessante. Por isso temos como perspectivas investigar a aplicação do método para modelagens baseadas em média de Reynolds, em microfluídica e escoamentos de Stokes. Em especial, quanto menor o número de Reynolds, maior parece ser o Δt suportado pelo método sem aumentar o número de iterações necessárias ao *solver* linear.

O desenvolvimento do Cyberex resultou em um código-fonte moderno, organizado de maneira modular e extensível, onde é fácil adicionar novos modelos e funcionalidades. Algumas perspectivas para futuros desenvolvimentos no Cyberex são:

- suporte à adição de ruído, para escoamentos turbulentos com LES;
- a adição de outros modelos submalha, como o modelo de Germano (LILLY, 1992);
- a implicitação do modelo submalha, de modo que a viscosidade turbulenta seja resolvida simultaneamente com as variáveis primárias pelo método não linear;
- a implementação genérica da função distância à parede — usada na função de van Driest — para que ela não precise ser dada pelo usuário;
- a adição de alternativas para a integração temporal, como o passo fracionado θ (TUREK, 1996);
- a implementação de *solvers* não lineares de convergência mais rápida que o método de Newton;
- a adição de precondicionadores alternativos para a estimativa do inverso do complemento de Schur, como o SIMPLE (ELMAN et al., 2008);
- portar os desenvolvimentos feitos em outras ferramentas do nosso grupo de pesquisa, de modo a centralizar os esforços de desenvolvimento.

Referências

AFTOSMIS, M. *Cart3D→flowCart→reorder*. 2004. Disponível em: <http://people.nas.nasa.gov/aftosmis/cart3d/flowCart_reorder.html>. Citado na página 59.

ALVES, M. A.; OLIVEIRA, P. J.; PINHO, F. T. A convergent and universally bounded interpolation scheme for the treatment of advection. *International Journal for Numerical Methods in Fluids*, v. 41, n. 1, p. 47–75, 2003. Disponível em: <https://doi.org/10.1002/fld.428>. Citado 2 vezes nas páginas 97 e 98.

AMESTOY, P. R. et al. A fully asynchronous multifrontal solver using distributed dynamic scheduling. *SIAM Journal on Matrix Analysis and Applications*, v. 23, n. 1, p. 15–41, 2001. Disponível em: <https://doi.org/10.1137/S0895479899358194>. Citado na página 110.

ARCHAMBEAU, F.; MÉCHITOUA, N.; SAKIZ, M. Code Saturne: A Finite Volume Code for the computation of turbulent incompressible flows - Industrial Applications. *International Journal on Finite Volumes*, Institut de Mathématiques de Marseille, AMU, v. 1, n. 1, fev. 2004. Disponível em: <https://hal.archives-ouvertes.fr/hal-01115371>. Citado na página 42.

ARMALY, B. F. et al. Experimental and theoretical investigation of backward-facing step flow. *Journal of Fluid Mechanics*, Cambridge University Press, v. 127, p. 473–496, 1983. Disponível em: <https://doi.org/10.1017/S0022112083002839>. Citado na página 127.

AYACHIT, U. *The Paraview Guide: A Parallel Visualization Application*. New York: Kitware, Inc., 2015. ISBN 1-930934-29-7. Citado na página 50.

BAKER, A. H. et al. Scaling hypre's multigrid solvers to 100,000 cores. In: BERRY, M. W. et al. (Ed.). *High-Performance Scientific Computing*. London: Springer, 2012. cap. 13, p. 261–279. Disponível em: <https://doi.org/10.1007/978-1-4471-2437-5_13>. Citado na página 41.

BALAY, S. et al. *PETSc Users Manual*. [S.l.], 2015. Disponível em: <http://www.mcs.anl.gov/petsc>. Citado na página 31.

BARBI, F. *Experimentação numérica de bolhas em ascensão*. Tese (Doutorado) — Universidade Federal de Uberlândia, 2016. Disponível em: <https://repositorio.ufu.br/handle/123456789/18155>. Citado na página 35.

BERGER, M.; BOKHARI, S. A partitioning strategy for nonuniform problems on multiprocessors. *Computers, IEEE Transactions on*, C-36, n. 5, p. 570–580, May 1987. ISSN 0018-9340. Disponível em: <https://doi.org/10.1109/TC.1987.1676942>. Citado na página 57.

BRAY, T. (Ed.). *The JavaScript Object Notation (JSON) Data Interchange Format*. [S.l.], 2014. 1-16 p. Disponível em: <https://tools.ietf.org/html/rfc7159>. Citado na página 32.

BRAY, T. et al. *Extensible Markup Language (XML) 1.0*. 5. ed. [s.n.], 2008. World Wide Web Consortium, Recommendation REC-xml-20081126. Disponível em: <https://www.w3.org/TR/xml/>. Citado na página 31.

BUNCH, J. R.; HOPCROFT, J. E. Triangular factorization and inversion by fast matrix multiplication. *Mathematics of Computation*, American Mathematical Society, v. 28, n. 125, p. 231–236, 1974. ISSN 00255718, 10886842. Disponível em: <https://doi.org/10.1090/S0025-5718-1974-0331751-8>. Citado na página 109.

CARETTO, L.; CURR, R.; SPALDING, D. Two numerical methods for three-dimensional boundary layers. *Computer Methods in Applied Mechanics and Engineering*, v. 1, n. 1, p. 39 – 57, 1972. ISSN 0045-7825. Disponível em: <https://doi.org/10.1016/0045-7825(72)90020-5>. Citado 3 vezes nas páginas 35, 36 e 39.

CATALYUREK, U.; AYKANAT, C. Hypergraph-partitioning-based decomposition for parallel sparse-matrix vector multiplication. *IEEE Trans. Parallel Distrib. Syst.*, IEEE Press, Piscataway, NJ, USA, v. 10, n. 7, p. 673–693, 1999. ISSN 1045-9219. Disponível em: <https://doi.org/10.1109/71.780863>. Citado na página 56.

CHEN, Z.; PRZEKWAS, A. A coupled pressure-based computational method for incompressible/compressible flows. *Journal of Computational Physics*, v. 229, n. 24, p. 9150 – 9165, 2010. ISSN 0021-9991. Disponível em: <https://doi.org/10.1016/j.jcp.2010.08.029>. Citado 3 vezes nas páginas 35, 38 e 39.

CHEVALIER, C.; PELLEGRINI, F. Pt-scotch: A tool for efficient parallel graph ordering. *Parallel computing*, Elsevier, v. 34, n. 6, p. 318–331, 2008. Disponível em: <https://doi.org/10.1016/j.parco.2007.12.001>. Citado 2 vezes nas páginas 55 e 58.

CHILDS, H. et al. VisIt: An End-User Tool For Visualizing and Analyzing Very Large Data. In: *High Performance Visualization–Enabling Extreme-Scale Scientific Insight*. [S.l.: s.n.], 2012. p. 357–372. ISBN 978-1439875728. Citado na página 50.

CLARKE l. J.; MARK, E. Enhancements to the extensible data model and format (xdmf). In: *2007 DoD High Performance Computing Modernization Program Users Group Conference*. [s.n.], 2007. p. 322–327. Disponível em: <https://doi.org/10.1109/HPCMP-UGC.2007.30>. Citado na página 31.

DAMASCENO, M. M. R. *Desenvolvimento de uma modelagem para escoamentos reativos em malhas adaptativas do tipo bloco-estruturada*. Tese (Doutorado) — Universidade Federal de Uberlândia, 2018. Disponível em: <https://dx.doi.org/10.14393/ufu.te.2018.771>. Citado na página 35.

DARWISH, M.; MOUKALLED, F. A fully coupled navier-stokes solver for fluid flow at all speeds. *Numerical Heat Transfer, Part B: Fundamentals*, Taylor & Francis, v. 65, n. 5, p. 410–444, 2014. Disponível em: <https://doi.org/10.1080/10407790.2013.869102>. Citado 4 vezes nas páginas 35, 36, 38 e 39.

DARWISH, M.; SRAJ, I.; MOUKALLED, F. A coupled finite volume solver for the solution of incompressible flows on unstructured grids. *Journal of Computational Physics*, v. 228, n. 1, p. 180 – 201, 2009. ISSN 0021-9991. Disponível em: <https://doi.org/10.1016/j.jcp.2008.08.027>. Citado 2 vezes nas páginas 38 e 39.

DAVIS, T. A. Algorithm 832: Umfpack v4.3—an unsymmetric-pattern multifrontal method. *ACM Trans. Math. Softw.*, ACM, New York, NY, USA, v. 30, n. 2, p. 196–199, jun. 2004. ISSN 0098-3500. Disponível em: <https://doi.org/10.1145/992200.992206>. Citado na página 109.

DENG, G. B. et al. A new fully coupled solution of the navier-stokes equations. *International Journal for Numerical Methods in Fluids*, v. 19, n. 7, p. 605–639, 1994. Disponível em: <https://doi.org/10.1002/fld.1650190705>. Citado 5 vezes nas páginas 35, 36, 38, 39 e 96.

DEUTSCH, P. *GZIP file format specification version 4.3*. [S.l.], 1996. 1-12 p. Disponível em: <https://tools.ietf.org/html/rfc1952>. Citado na página 50.

DEVINE, K. et al. Zoltan data management services for parallel dynamic applications. *Computing in Science and Engineering*, v. 4, n. 2, p. 90–97, 2002. Disponível em: <https://doi.org/10.1109/5992.988653>. Citado na página 55.

DEVINE, K. et al. Parallel hypergraph partitioning for scientific computing. In: *Parallel and Distributed Processing Symposium, 2006. IPDPS 2006. 20th International*. [s.n.], 2006. p. 10 pp.–. Disponível em: <https://doi.org/10.1109/IPDPS.2006.1639359>. Citado na página 58.

ELMAN, H. et al. A taxonomy and comparison of parallel block multi-level preconditioners for the incompressible navier–stokes equations. *Journal of Computational Physics*, v. 227, n. 3, p. 1790 – 1808, 2008. ISSN 0021-9991. Disponível em: <https://doi.org/10.1016/j.jcp.2007.09.026>. Citado 4 vezes nas páginas 37, 39, 110 e 138.

ELMAN, H. et al. Block preconditioners based on approximate commutators. *SIAM J. Sci. Comput.*, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, v. 27, n. 5, p. 1651–1668, nov. 2005. ISSN 1064-8275. Disponível em: <https://doi.org/10.1137/040608817>. Citado 2 vezes nas páginas 37 e 111.

FALGOUT, R. D.; YANG, U. M. hypre: A library of high performance preconditioners. In: SLOOT, P. M. A. et al. (Ed.). *Computational Science — ICCS 2002*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2002. p. 632–641. ISBN 978-3-540-47789-1. Citado na página 47.

FELDMAN, M. Top500 meanderings: Supercomputers for weather forecasting have come a long way. *TOP500.org*, Apr 2017. Disponível em: <https://www.top500.org/news/top500-meanderings-supercomputers-for-weather-forecasting-have-come-a-long-way/>. Citado na página 23.

FERZIGER, J. H.; PERIC, M. *Computational Methods for Fluid Dynamics*. Berlin: Springer-Verlang, 2002. ISBN 978-3-540-42074-3. Citado 2 vezes nas páginas 98 e 99.

GAHVARI, H. et al. Modeling the performance of an algebraic multigrid cycle using hybrid mpi/openmp. In: *2012 41st International Conference on Parallel Processing*. [s.n.], 2012. p. 128–137. ISSN 0190-3918. Disponível em: <https://doi.org/10.1109/ICPP.2012.41>. Citado na página 41.

GARTLING, D. K. A test problem for outflow boundary conditions—flow over a backward-facing step. *International Journal for Numerical Methods in Fluids*, v. 11, n. 7, p. 953–967, 1990. Disponível em: <https://doi.org/10.1002/fld.1650110704>. Citado na página 127.

GUO, X. et al. Developing hybrid openmp-mpi parallelism for fluidity – next generation geophysical fluid modelling technology. In: *CUG 2012 – Greengineering the Future: Online Proceedings*. Stuttgart, Germany: Cray User Group, 2012. Disponível em: <http://www.hector.ac.uk/cse/distributedcse/reports/fluidity-icom02/fluidity-icom02/index.html>. Citado na página 41.

HAVERKORT, H. J. An inventory of three-dimensional hilbert space-filling curves. *CoRR*, abs/1109.2323, 2011. Disponível em: <http://arxiv.org/abs/1109.2323>. Citado na página 57.

HÜBBE, N.; KUNKEL, J. Reducing the hpc-datastorage footprint with mafisc—multidimensional adaptive filtering improved scientific data compression. *Computer Science - Research and Development*, v. 28, n. 2, p. 231–239, May 2013. ISSN 1865-2042. Disponível em: <https://doi.org/10.1007/s00450-012-0222-4>. Citado na página 50.

JOVIC, S.; DRIVER, D. M. *Backward Facing Step Measurements at Low Reynolds Number, Re_h = 5000*. [S.l.], 1994. 1-24 p. Disponível em: <https://ntrs.nasa.gov/archive/nasa/casi.ntrs.nasa.gov/19940028784.pdf>. Citado 3 vezes nas páginas 128, 129 e 131.

KARYPIS, G.; KUMAR, V. Parallel multilevel series k-way partitioning scheme for irregular graphs. *Siam Review*, SIAM, v. 41, n. 2, p. 278–300, 1999. Disponível em: <https://doi.org/10.1109/SUPERC.1996.183537>. Citado na página 58.

KARYPIS, G.; SCHLOEGEL, K. *ParMETIS - Parallel Graph Partitioning and Sparse Matrix Ordering Library - Version 4.0*. Minneapolis, MN, 2011. Disponível em: <http://glaros.dtc.umn.edu/gkhome/fetch/sw/parmetis/manual.pdf>. Citado na página 54.

KIM, J.; MOIN, P. Application of a fractional-step method to incompressible navier-stokes equations. *Journal of Computational Physics*, v. 59, n. 2, p. 308 – 323, 1985. ISSN 0021-9991. Disponível em: <https://doi.org/10.1016/0021-9991(85)90148-2>. Citado na página 127.

LEE, T.; MATEESCU, D. Experimental and numerical investigation of 2-d backward-facing step flow. *Journal of Fluids and Structures*, v. 12, n. 6, p. 703 – 716, 1998. ISSN 0889-9746. Disponível em: <https://doi.org/10.1006/jfls.1998.0166>. Citado 2 vezes nas páginas 124 e 127.

LEER, B. van. Towards the ultimate conservative difference scheme. ii. monotonicity and conservation combined in a second-order scheme. *Journal of Computational Physics*, v. 14, n. 4, p. 361 – 370, 1974. ISSN 0021-9991. Disponível em: <https://doi.org/10.1016/0021-9991(74)90019-9>. Citado na página 97.

LEONARD, B. P. Bounded higher-order upwind multidimensional finite-volume convection-diffusion algorithms. In: MINKOWYCZ, W. J.; SPARROW, E. M. (Ed.). *Advances in Numerical Heat Transfer*. Washington, DC: Taylor & Francis, 1996. v. 1, cap. 1, p. 1–57. ISBN 978-1560324416. Citado na página 97.

LILLY, D. K. A proposed modification of the germano subgrid-scale closure method. *Physics of Fluids A: Fluid Dynamics*, v. 4, n. 3, p. 633–635, 1992. Disponível em: <https://doi.org/10.1063/1.858280>. Citado na página 138.

LIMA, R. S. d. *Desenvolvimento e implementação de malhas adaptativas bloco-estruturadas para computação paralela em mecânica dos fluidos*. Tese (Doutorado) — Universidade Federal de Uberlândia, 2012. Disponível em: <https://repositorio.ufu.br/handle/123456789/14727>. Citado 4 vezes nas páginas 35, 42, 54 e 68.

MATTERN, F. Global quiescence detection based on credit distribution and recovery. *Inf. Process. Lett.*, Elsevier North-Holland, Inc., Amsterdam, The Netherlands, The Netherlands, v. 30, n. 4, p. 195–200, fev. 1989. ISSN 0020-0190. Disponível em: <https://doi.org/10.1016/0020-0190(89)90212-3>. Citado na página 77.

MELO, R. R. da S. *Modelagem e Simulação de Escoamentos Turbulentos com Efeitos Térmicos, Utilizando a Metodologia da Fronteira Imersa e Malha Adaptativa*. Tese (Doutorado) — Universidade Federal de Uberlândia, 2017. Disponível em: <https://repositorio.ufu.br/handle/123456789/18667>. Citado 2 vezes nas páginas 35 e 128.

MPI FORUM. *MPI: A Message-Passing Interface Standard – Version 3.1*. Knoxville, TN, USA, 2015. Disponível em: <https://www.mpi-forum.org/docs/mpi-3.1/mpi31-report.pdf>. Citado na página 25.

NETO, A. S. et al. A numerical investigation of the coherent vortices in turbulence behind a backward-facing step. *Journal of Fluid Mechanics*, Cambridge University Press, v. 256, p. 1–25, 1993. Disponível em: <https://doi.org/10.1017/S0022112093002691>. Citado na página 128.

OLUKOTUN, K.; HAMMOND, L. The future of microprocessors. *Queue*, ACM, New York, NY, USA, v. 3, n. 7, p. 26–29, set. 2005. ISSN 1542-7730. Disponível em: <http://doi.org/10.1145/1095408.1095418>. Citado na página 40.

PATANKAR, S.; SPALDING, D. A calculation procedure for heat, mass and momentum transfer in three-dimensional parabolic flows. *International Journal of Heat and Mass Transfer*, v. 15, n. 10, p. 1787 – 1806, 1972. ISSN 0017-9310. Disponível em: <https://doi.org/10.1016/0017-9310(72)90054-3>. Citado na página 35.

PATANKAR, S. V. A calculation procedure for two-dimensional elliptic situations. *Numerical Heat Transfer*, Taylor & Francis, v. 4, n. 4, p. 409–425, 1981. Disponível em: <https://doi.org/10.1080/01495728108961801>. Citado na página 37.

PERNICE, M.; TOCCI, M. D. A multigrid-preconditioned newton–krylov method for the incompressible navier–stokes equations. *SIAM Journal on Scientific Computing*, v. 23, n. 2, p. 398–418, 2001. Disponível em: <https://doi.org/10.1137/S1064827500372250>. Citado 3 vezes nas páginas 35, 37 e 39.

PILKINGTON, J. R.; BADEN, S. B. *Partitioning with Spacefilling Curves*. [S.l.], 1994. Disponível em: <http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.31.847>. Citado na página 57.

RHIE, C. M.; CHOW, W. L. Numerical study of the turbulent flow past an airfoil with trailing edge separation. *AIAA journal*, v. 21, n. 11, p. 1525–1532, 1983. Disponível em: <https://doi.org/10.2514/3.8284>. Citado na página 36.

RUNCHAL, A. K. Condif: A modified central-difference scheme for convective flows. *International Journal for Numerical Methods in Engineering*, v. 24, n. 8, p. 1593–1608, 1987. Disponível em: <https://doi.org/10.1002/nme.1620240814>. Citado na página 97.

SCHNEIDER, G. E.; RAW, M. J. Control volume finite-element method for heat transfer and fluid flow using colocated variables – 1. computational procedure. *Numerical Heat Transfer*, Taylor & Francis, v. 11, n. 4, p. 363–390, 1987. Disponível em: <https://doi.org/10.1080/10407788708913560>. Citado na página 36.

SCHROEDER, W.; MARTIN, K.; LORENSEN, B. *The Visualization Toolkit*: An object-oriented approach to 3d graphics. [S.l.]: Kitware, 2006. ISBN 9781930934191. Citado na página 30.

SHANG, Z. Impact of mesh partitioning methods in cfd for large scale parallel computing. *Computers & Fluids*, v. 103, p. 1 – 5, 2014. ISSN 0045-7930. Disponível em: <https://doi.org/10.1016/j.compfluid.2014.07.016>. Citado na página 42.

SHIH, T. M.; TAN, C. H.; HWANG, B. C. Effects of grid staggering on numerical schemes. *International Journal for Numerical Methods in Fluids*, v. 9, n. 2, p. 193–212, 1989. Disponível em: <https://doi.org/10.1002/fld.1650090206>. Citado 2 vezes nas páginas 115 e 117.

SIMÃO, A. S. et al. Desenvolvimentos matemáticos e numéricos em escoamentos bifásicos aplicados a processos de refino. Não publicado. 2014. Citado na página 24.

SOHN, J. L. Evaluation of fidap on some classical laminar and turbulent benchmarks. *International Journal for Numerical Methods in Fluids*, v. 8, n. 12, p. 1469–1490, 1988. Disponível em: <https://doi.org/10.1002/fld.1650081202>. Citado na página 127.

SPODE, C.; CAMPREGHER, R.; NETO, A. D. Parallel simulation of turbulent flow in a backward-facing step. In: *Proceedings of the COBEM 2005 : 18th international congress of mechanical engineering*. Ouro Preto, Minas Gerais, Brazil: Associação Brasileira de Engenharia e Ciências Mecânicas, 2005. ISBN 9788585769260. Citado na página 128.

SUTTER, H.; LARUS, J. Software and the concurrency revolution. *Queue*, ACM, New York, NY, USA, v. 3, n. 7, p. 54–62, set. 2005. ISSN 1542-7730. Disponível em: <http://doi.org/10.1145/1095408.1095421>. Citado na página 40.

THE HDF GROUP. *Hierarchical Data Format, version 5*. 1997–2018. Disponível em: <http://www.hdfgroup.org/HDF5/>. Citado na página 31.

TUREK, S. A comparative study of time-stepping techniques for the incompressible navier-stokes equations: From fully implicit non-linear schemes to semi-implicit projection methods. *International Journal for Numerical Methods in Fluids*, v. 22, n. 10, p. 987–1011, 1996. Disponível em: <https://doi.org/10.1002/(SICI)1097-0363(19960530)22:10%3C987::AID-FLD394%3E3.0.CO;2-7>. Citado 4 vezes nas páginas 35, 37, 39 e 138.

VANKA, S. P. Block-implicit multigrid solution of navier-stokes equations in primitive variables. *Journal of Computational Physics*, v. 65, n. 1, p. 138 – 158, 1986. ISSN 0021-9991. Disponível em: <https://doi.org/10.1016/0021-9991(86)90008-2>. Citado 2 vezes nas páginas 36 e 39.

WACHEM, B. G. van; GOPALA, V. R. A coupled solver approach for multiphase flow calculations on collocated grids. In: WESSELING, E. O. P.; PÉRIAUX, J. (Ed.). *ECCOMAS CFD 2006: Proceedings of the European Conference on Computational Fluid Dynamics*. Egmond aan Zee, Netherlands: Delft University of Technology, 2006. ISBN 90-9020970-0. Disponível em: <http://resolver.tudelft.nl/uuid:f7d8a939-d357-48a2-9a3a-b98be80f2711>. Citado 4 vezes nas páginas 33, 35, 38 e 39.

WEILAND, M. et al. Mixed-mode implementation of petsc for scalable linear algebra on multi-core processors. *CoRR*, abs/1205.2005, 2012. Disponível em: <http://arxiv.org/abs/1205.2005>. Citado na página 41.

WILLIAMS, R. D. Performance of dynamic load balancing algorithms for unstructured mesh calculations. *Concurrency: Pract. Exper.*, John Wiley and Sons Ltd., Chichester, UK, v. 3, n. 5, p. 457–481, out. 1991. ISSN 1040-3108. Disponível em: <https://doi.org/10.1002/cpe.4330030502>. Citado na página 57.

XIAO, C.-N.; DENNER, F.; WACHEM, B. G. van. Fully-coupled pressure-based finite-volume framework for the simulation of fluid flows at all speeds in complex geometries. *Journal of Computational Physics*, v. 346, p. 91 – 130, 2017. ISSN 0021-9991. Disponível em: <https://doi.org/10.1016/j.jcp.2017.06.009>. Citado 2 vezes nas páginas 35 e 39.