

HigTree

**Erick Rodrigues Canterle¹ Pedro Vinicius Souza Coimbra²
Thiago Ferreira de França e Queiroz³ Vinícius de Sá Ferreira⁴
Daniel Garcia Leal Raymundo⁵**

¹²³⁴ICMC - Instituto De Ciências Matemáticas e de Computação
Universidade de São Paulo

⁵FCUP - Faculdade de Ciências da Universidade do Porto
Universidade do Porto

7 de novembro de 2025





Índices

1. Introdução
2. Estrutura de dados da HiGtree
3. Mínimos quadrados
4. Paralelismo
5. Aproximação de Derivadas em HiGs
6. Moving Least Squares (MLS)
7. Condições de Contorno
8. Verificação Numérica
9. Convergência em Malhas Non-Graded
10. Agradecimentos
11. Referências



Introdução



O que é *HgFlow/HigThree*?

O *HgFlow/HigTree* é uma biblioteca desenvolvida majoritariamente em C, que disponibiliza um conjunto de ferramentas voltadas à realização de simulações numéricas em domínios complexos. Esses domínios são discretizados por meio de uma estrutura de dados de malha cartesiana baseada em árvores generalizadas, denominada **HiG-Tree**. A biblioteca foi projetada com foco no paralelismo, empregando o padrão **MPI** (Message Passing Interface) / **Zoltan** para a distribuição de dados e da carga computacional.

Um pouco de História

Esta biblioteca foi desenvolvida em (informar data) sob a supervisão do professor Antonio Castelo Filho e **CIA**, com financiamento da Petrobras. O projeto contou com a colaboração de mais de dez alunos de pós-graduação, que contribuíram para seu aprimoramento e aplicação em diversas pesquisas, resultando na publicação de inúmeros artigos científicos, como ([Sousa et al.](#)), ([Castillo-Sánchez et al.](#)) e dissertações como ([Raymundo](#)).



Motivação para o desenvolvimento da HiG-Tree

- Malhas hierárquicas baseadas em árvore combinam:
 - Rapidez de discretizações cartesianas (diferenças finitas)
 - Flexibilidade e precisão do refinamento local
- **Desafio principal:** Adaptar o estêncil nas interfaces entre elementos de diferentes tamanhos
- Métodos tradicionais usam interpolações geométricas de alta ordem
- Estas interpolações são específicas por local e complexas em 3D

Contribuição da Biblioteca HiG-Tree



Nova Abordagem Proposta

- Método robusto baseado em **interpolação meshless** (Moving Least Squares - MLS)
- Computa pesos da aproximação por diferenças finitas em malhas hierárquicas
- Permite configurações complexas de malha
- Mantém a ordem de precisão do método
- Mais flexível que métodos baseados em interpolações geométricas



Dimensão do Projeto

Figura 1: Escopo do projeto (disponível em: <<https://github.com/antoniocastelofilho/HigFlow>>) mensurado pela quantidade de arquivos e pelo total de linhas de código.

github.com/AlDanial/cloc v 2.06 T=0.39 s (1047.9 files/s, 528029.0 lines/s)				
Language	files	blank	comment	code
C	166	13711	39091	111772
SVG	18	18	18	7605
YAML	22	99	18	7271
C/C++ Header	94	2540	4192	4926
TeX	8	675	551	4598
MATLAB	14	731	153	2306
Bourne Shell	33	92	8	2005
make	17	346	45	1282
C++	4	249	151	1018
CMake	20	158	199	697
Python	8	136	92	405
Mathematica	2	61	0	83
Markdown	2	45	0	69
XML	2	0	0	60
Windows Module Definition	1	17	0	55
Text	1	24	0	39
SUM:	412	18902	44518	144191



Estrutura de dados da HiGtree



A HiG-Tree (*Hierarchical Grid Tree*) é uma estrutura de dados em árvore usada em malhas adaptativas, que permite refinamento, dividindo o espaço recursivamente, economizando memória e tempo de processamento.

Nos casos 2D e 3D, chamamos de *quadtree* e *octree*, respectivamente.



Exemplo de HiG-Tree

8	16	18	20
	15	17	19
2	5	7	11 12 13 14
	4	6	10
1		3	9

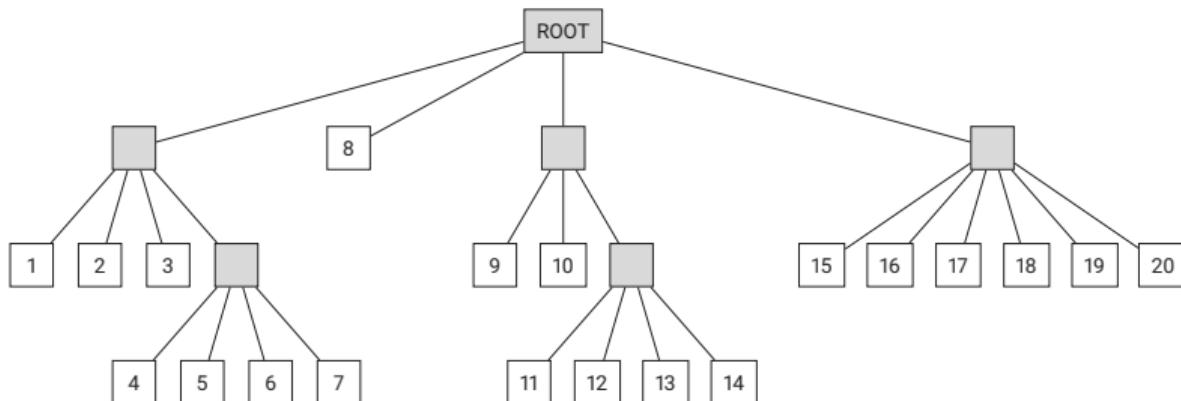


Figura 2: Exemplo de malha 2D e sua quadtree associada.
Fonte: Sousa et al.



Células da HiG-Tree

```
typedef struct hig_cell {  
    Point lowpoint;  
    Point highpoint;  
    int numcells[DIM];  
    struct hig_cell * parent;  
    int posinparent;  
    struct hig_cell **children;  
    uniqueid ids[NUMIDS];  
} hig_cell;
```

$(x_{1min}, x_{2min}, \dots, x_{mmin})$

$(x_{1max}, x_{2max}, \dots, x_{mmax})$

(n_1, n_2, \dots, n_m)

Célula-pai

Posição em relação ao pai

Células-filho

Id de cada célula/faceta



Facetas da HiG-Tree

```
typedef struct hig_facet {  
    hig_cell *c;           Célula associada  
    int dim;              Cada faceta tem dimensão m-1  
    int dir;              Direção {0,1}  
} hig_facet;
```

dir=0: Facetas geradas pelas arestas que contém o lowpoint.
dir=1: Facetas geradas pelas arestas que contém o highpoint.

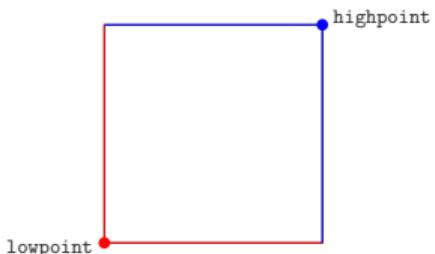


Figura 3: Facetas de uma célula 2D.



Visualização das malhas

Escrevemos a malha em formato AMR (*Adaptive Mesh Refinement*) como entrada, por exemplo:

```
0.0 8.0 -1.0 1.0      lowpoint highpoint = xmin xmax ymin ymax
2                      numlevels
0.05 0.05 1           (level 1) delta numpatches = delta_x delta_y numpatches
1 1 160 40            initcell patchsize
0.025 0.025 4         (level 2) delta numpatches = delta_x delta_y numpatches
1 1 320 16            initcell patchsize
1 65 320 16           initcell patchsize
1 17 16 48            initcell patchsize
305 17 16 48          initcell patchsize
```

então, geramos um arquivo de saída em formato VTK (*Visualization Toolkit*) com `higtree/src/higtree-io.c` e visualizamos com o ParaView.



Visualização das malhas

```
0.0 8.0 -1.0 1.0  
1  
0.05 0.05 1  
1 1 160 40
```

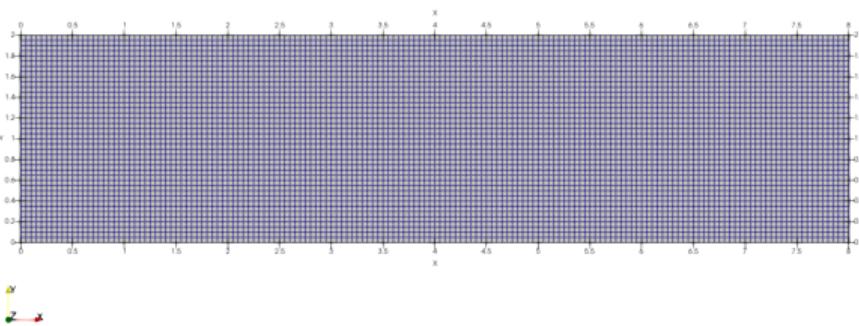


Figura 4: Exemplo de malha 2D com 1 nível de refinamento.



```
0.0 8.0 -1.0 1.0  
2  
0.05 0.05 1  
1 1 160 40  
0.025 0.025 4  
1 1 320 16  
1 65 320 16  
1 17 16 48  
305 17 16 48
```

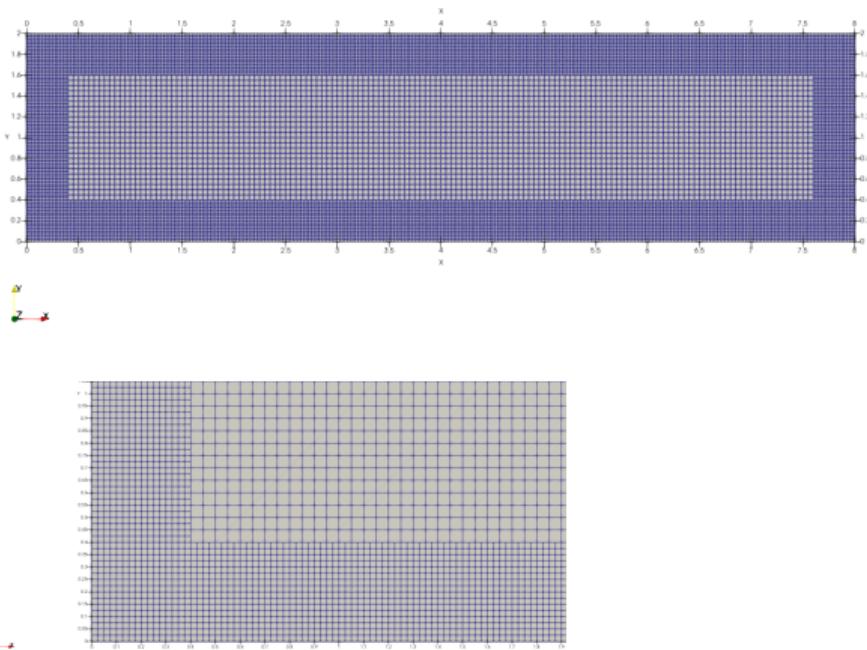


Figura 5: Exemplo de malha 2D com 2 níveis de refinamento.



```
0.0 8.0 -1.0 1.0  
3  
0.05 0.05 1  
1 1 160 40  
0.025 0.025 4  
1 1 320 16  
1 65 320 16  
1 17 16 48  
305 17 16 48  
0.0125 0.0125 4  
1 1 640 16  
1 145 640 16  
1 17 16 128  
625 17 16 128
```

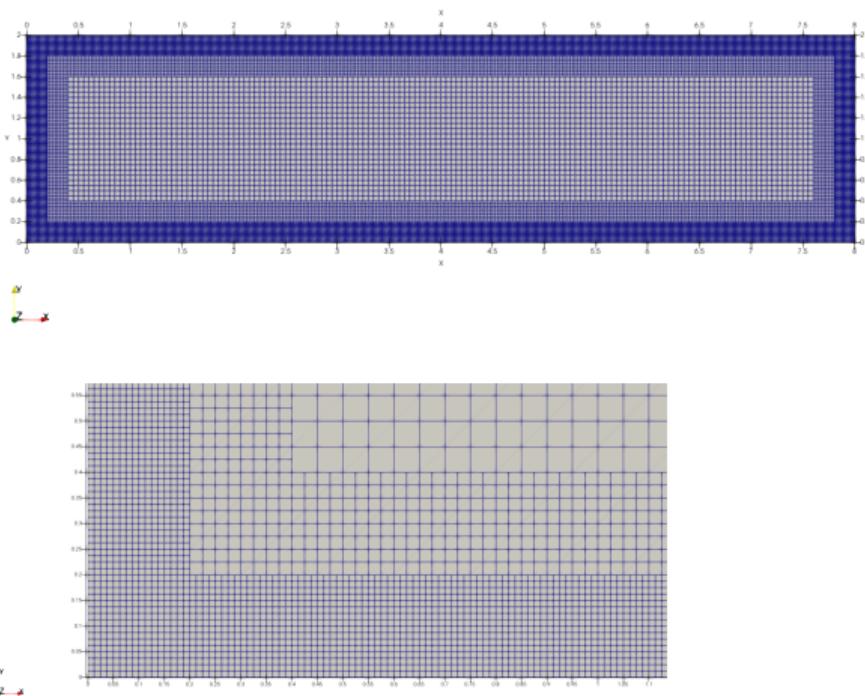


Figura 6: Exemplo de malha 2D com 3 níveis de refinamento.



```
0.0 8.0 -1.0 1.0  
3  
0.05 0.05 1  
1 1 160 40  
0.025 0.025 4  
1 1 320 16  
1 65 320 16  
1 17 16 48  
305 17 16 48  
0.0125 0.0125 4  
1 1 640 16  
1 145 640 16  
1 17 16 128  
625 17 16 128
```

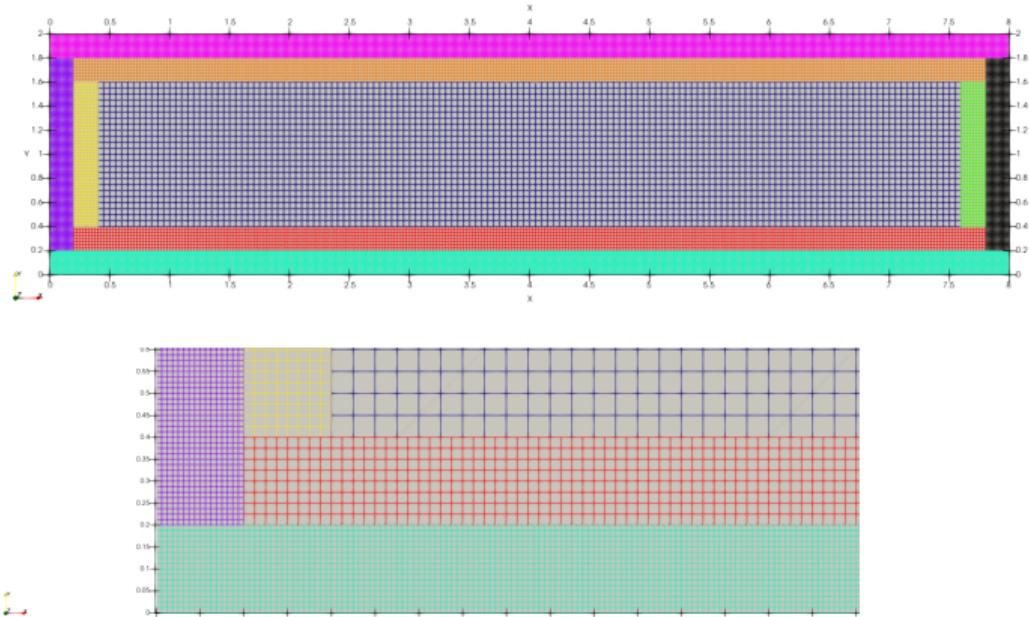


Figura 7: Exemplo de malha 2D com 3 níveis de refinamento.



Mínimos quadrados,
mínimos quadrados
ponderados, mínimos
quadrados móveis



Mínimos quadrados – LS

Seja f uma função e V um espaço vetorial $(n + 1)$ -dimensional gerado pelas funções $\varphi_0, \varphi_1, \dots, \varphi_n$. Deseja-se encontrar $F^* \in V$ com $F^* = \sum_{i=0}^n \alpha_i^* \varphi_i$ que melhor aproxima f , i.e., quer-se obter $Q = \min_{\alpha^*} \|f - F^*\|^2$.

Para encontrar F^* , basta calcular e resolver o sistema de equações normais

$$\begin{bmatrix} \langle \varphi_0, \varphi_0 \rangle & \cdots & \langle \varphi_0, \varphi_n \rangle \\ \vdots & \ddots & \vdots \\ \langle \varphi_n, \varphi_0 \rangle & \cdots & \langle \varphi_n, \varphi_n \rangle \end{bmatrix} \begin{bmatrix} \alpha_0^* \\ \vdots \\ \alpha_n^* \end{bmatrix} = \begin{bmatrix} \langle f, \varphi_0 \rangle \\ \vdots \\ \langle f, \varphi_n \rangle \end{bmatrix}$$





Mínimos quadrados ponderados – WLS

Vê-se que $\langle \cdot, \cdot \rangle_W$ dado por $\langle u, v \rangle_W = \sum_{i=0}^n w_i u_i v_i = u^\top W v$ com $W = \text{diag}(w_0, \dots, w_n)$ e $w_i > 0$ define um produto interno em \mathbb{R}^{n+1} .



Mínimos quadrados ponderados – WLS

Vê-se que $\langle \cdot, \cdot \rangle_W$ dado por $\langle u, v \rangle_W = \sum_{i=0}^n w_i u_i v_i = u^T W v$ com $W = \text{diag}(w_0, \dots, w_n)$ e $w_i > 0$ define um produto interno em \mathbb{R}^{n+1} .

O método dos mínimos quadrados ponderados generaliza LS ao considerar produtos internos da forma $\langle \cdot, \cdot \rangle_W$.



Mínimos quadrados ponderados – WLS

Vê-se que $\langle \cdot, \cdot \rangle_W$ dado por $\langle u, v \rangle_W = \sum_{i=0}^n w_i u_i v_i = u^T W v$ com $W = \text{diag}(w_0, \dots, w_n)$ e $w_i > 0$ define um produto interno em \mathbb{R}^{n+1} .

O método dos mínimos quadrados ponderados generaliza LS ao considerar produtos internos da forma $\langle \cdot, \cdot \rangle_W$.

Nota-se que LS é WLS com $W = I$.



Mínimos quadrados móveis – MLS

Consideremos f uma função discreta definida em $n + 1$ pontos. Seja V um espaço vetorial $(m + 1)$ -dimensional gerado por $\{\varphi_0, \dots, \varphi_m\}$ com $m < n$.



Mínimos quadrados móveis – MLS

Consideremos f uma função discreta definida em $n + 1$ pontos. Seja V um espaço vetorial $(m + 1)$ -dimensional gerado por $\{\varphi_0, \dots, \varphi_m\}$ com $m < n$. Quer-se aproximar f por uma função $F^* \in V$ por mínimos quadrados ponderados.



Mínimos quadrados móveis – MLS

Consideremos f uma função discreta definida em $n + 1$ pontos. Seja V um espaço vetorial $(m + 1)$ -dimensional gerado por $\{\varphi_0, \dots, \varphi_m\}$ com $m < n$. Quer-se aproximar f por uma função $F^* \in V$ por mínimos quadrados ponderados.

Definamos w uma função peso dada por

$$w(x) = \psi\left(\frac{|x - c|}{\sigma}\right), \quad \psi(t) = e^{-t^2},$$

onde c é o centro da gaussiana com raio controlado por σ .

Definamos $\langle \cdot, \cdot \rangle_{W(x)}$ um produto interno dado por

$$\langle u, v \rangle_{W(x)} = \sum_{i=0}^n w_i(x) u_i v_i,$$

onde

$$w_i(x) = \psi\left(\frac{|x - x_i|}{\sigma}\right)$$

e $W(x) = \text{diag}(w_0(x), w_1(x), \dots, w_n(x))$.



A melhor aproximação de f no sentido dos mínimos quadrados ponderados é dada pela função $F^* = \sum_{i=0}^m \alpha_i^* \varphi_i$, em que $(\alpha_0^*, \dots, \alpha_m^*)$ é solução do sistema linear $A(x)\alpha^*(x) = b(x)$, onde

$$A(x) = \begin{bmatrix} \langle \varphi_0, \varphi_0 \rangle_{W(x)} & \cdots & \langle \varphi_0, \varphi_n \rangle_{W(x)} \\ \vdots & \ddots & \vdots \\ \langle \varphi_n, \varphi_0 \rangle_{W(x)} & \cdots & \langle \varphi_n, \varphi_n \rangle_{W(x)} \end{bmatrix}$$

e

$$b(x) = \begin{bmatrix} \langle f, \varphi_0 \rangle_{W(x)} \\ \vdots \\ \langle f, \varphi_n \rangle_{W(x)} \end{bmatrix}.$$





Paralelismo



Tecnologias Empregadas

MPI

MPI (Message Passing Interface) é um padrão de comunicação amplamente utilizado em computação paralela e distribuída. Ela define um conjunto de rotinas e funções que permitem a troca de mensagens entre processos que podem estar executando em diferentes nós de um cluster ou em diferentes núcleos de um mesmo computador(<<https://www.open-mpi.org/>>).

Zoltan

A biblioteca Zoltan, desenvolvida pelo Sandia National Laboratories, é um kit de ferramentas de software projetado para simplificar o desenvolvimento e otimizar o desempenho de aplicações paralelas, especialmente aquelas que lidam com dados não estruturados e adaptativos. Seu foco principal é fornecer serviços de gerenciamento de dados, com destaque para o balanceamento de carga dinâmico, uma tarefa crucial para garantir a eficiência de computadores paralelos (<<https://sandialabs.github.io/Zoltan/>>).

boost/geometry

É uma biblioteca da coleção **Boost C++ Libraries** voltada para o processamento geométrico genérico. Ela fornece um conjunto abrangente de ferramentas e algoritmos para manipular e analisar objetos geométricos de forma eficiente, permitindo trabalhar com diferentes tipos de coordenadas e sistemas de referência, mantendo alta flexibilidade e desempenho (<<https://www.boost.org/>>).



O *HigFlow/HigTree* é executada em paralelo com memória distribuída, utilizando a API **MPI** para comunicação entre processos. O domínio é particionado por meio do algoritmo **Recursive Coordinate Bisection (RCB)** da biblioteca **Zoltan**, garantindo balanceamento de carga. A comunicação entre fronteiras é tratada com regiões de sobreposição (fringe), definidas pela **boost/-geometry**, assegurando a troca correta de dados entre vizinhos ([Raymundo](#)).



Algoritmo Recursive Coordinate Bisection

```
FUNÇÃO RCB(Região, NumPartesDesejadas)
    SE NumPartesDesejadas == 1 ENTÃO
        Retorne Região // Já atingimos o número desejado de partes
    FIM SE

    // 1. Selecionar o eixo de corte
    // Escolha o eixo (X, Y ou Z) que corresponde à dimensão mais longa da caixa delimitadora da região
    Eixo = EscolherEixoMaisLongo(Região)

    // 2. Calcular o ponto de corte
    // Encontre o ponto de corte ao longo do eixo selecionado que divide o peso dos dados igualmente
    PontoCorte = CalcularPontoCorte(Região, Eixo)

    // 3. Dividir a região
    Região1, Região2 = DividirRegião(Região, Eixo, PontoCorte)

    // 4. Dividir recursivamente as sub-regiões
    // Determine como distribuir o número de partes entre as duas sub-regiões
    NumPartesRegião1, NumPartesRegião2 = DistribuirPartes(NumPartesDesejadas)

    Resultado1 = RCB(Região1, NumPartesRegião1)
    Resultado2 = RCB(Região2, NumPartesRegião2)

    // 5. Combinar os resultados
    Retorne Combinar(Resultado1, Resultado2)
FIM FUNÇÃO
```

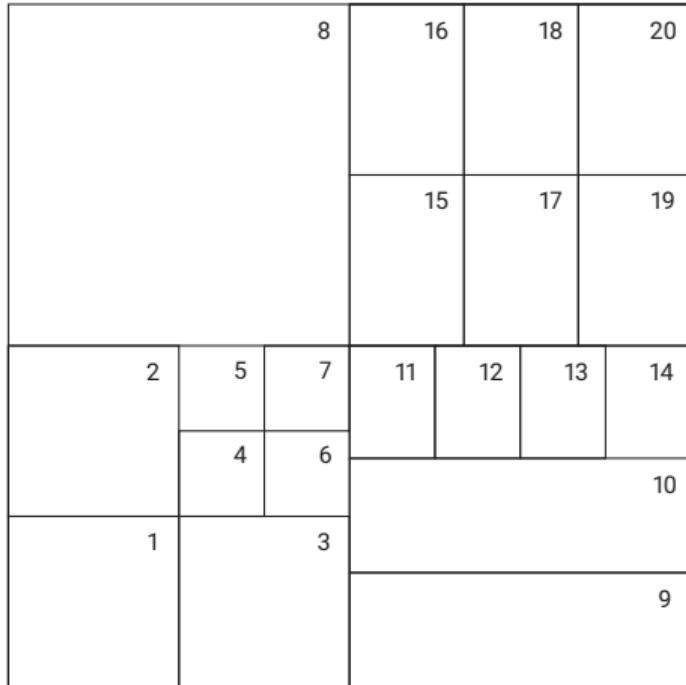


Figura 8: Exemplo de malha 2D e sua quadtree associada.

Fonte: Sousa et al.

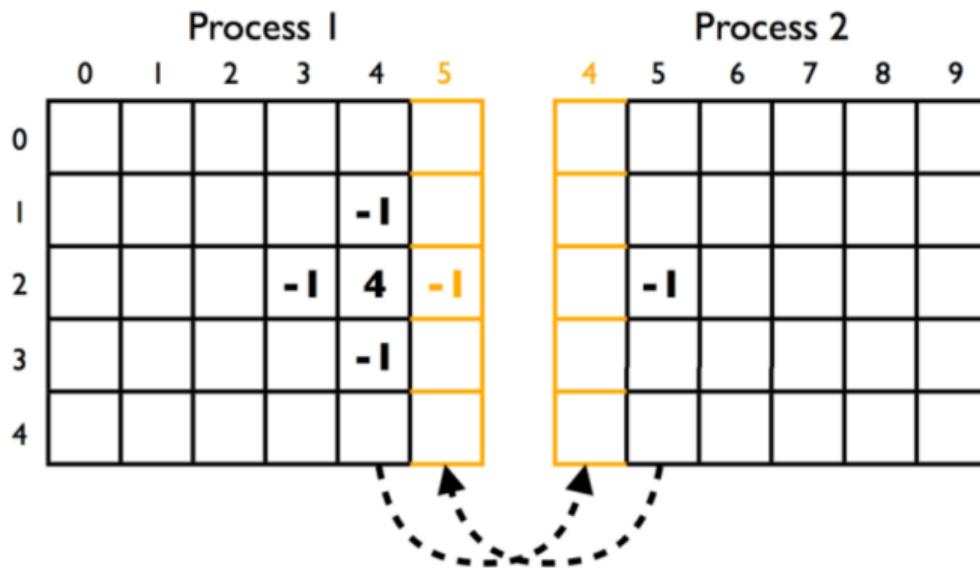


Figura 9: Exemplo visual de troca de informações em regiões fringe com stencil de laplaciano bidimensional.

fonte: [Raymundo \(2024\)](#)



Problema Modelo: Equação de Poisson

Equação de Poisson

$$\nabla^2 u(x, y) = f(x, y) \quad \text{em } \Omega \subset \mathbb{R}^2$$

Aproximação FD padrão:

$$\frac{1}{\delta_1^2}(U_l - 2U_c + U_r) + \frac{1}{\delta_2^2}(U_t - 2U_c + U_b) = f_c$$

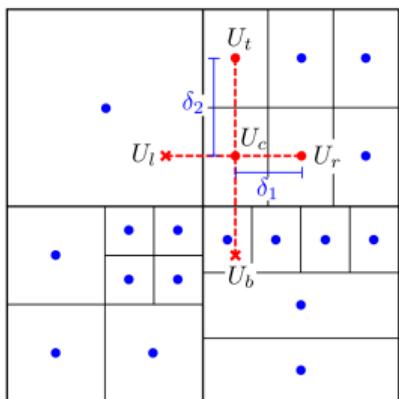


Figura 10: Estêncil FD de 2^a ordem



Interpolação Meshless

- Em malhas não uniformes, pontos como U_t e U_b não coincidem com a malha
- Necessidade de interpolação:

$$U_t = \sum_{k \in \mathcal{I}_t} w_k^t U_k, \quad U_b = \sum_{k \in \mathcal{I}_b} w_k^b U_k$$

- $\mathcal{I}_t = \{i_k, k = 1, \dots, N_l\}, \mathcal{I}_b = \{j_k, k = 1, \dots, N_b\}$: conjuntos de índices de vizinhos
- w_k^t, w_k^b : pesos computados via MLS



Aproximação Final

Sistema Linear Resultante

$$\frac{1}{\delta_1^2} \left(\sum_{k \in \mathcal{I}_t} w_k^t U_k - 2U_c + U_r \right) + \frac{1}{\delta_2^2} \left(U_t - 2U_c + \sum_{k \in \mathcal{I}_b} w_k^b U_k \right) = f_c$$

Forma Matricial

$$\sum_{k=1}^{N_u} A_k U_k = f_c, \quad c = 1, \dots, N_u$$

$$A\mathbf{u} = \mathbf{f}$$



Fundamentos do MLS

Aproximação

$$U(\mathbf{x}) = \sum_{j=1}^n c_j \Phi_j(\mathbf{x})$$

- Φ_j : funções de base polinomiais
- c_j : coeficientes determinados por mínimos quadrados
- Dado um conjunto de m pontos conhecidos $\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_m \in \mathbb{R}^d$, $m > n$, e m valores de função associados $u_1 = u(\mathbf{x}_1), u_2 = u(\mathbf{x}_2), \dots, u_m = u(\mathbf{x}_m)$, o procedimento para interpolar u em \mathbf{x} usando MLS é minimizar a função de erro:

$$E(\mathbf{c}) = \sum_{i=1}^m (U(\mathbf{x}_i) - u_i)^2 \lambda_i(\mathbf{x})$$

- Pesos: $\lambda_i(\mathbf{x}) = \frac{1}{\|\mathbf{x} - \mathbf{x}_i\|_2}$



Formulação Matricial

Problema de Mínimos Quadrados

$$E(\mathbf{c}) = \|WP\mathbf{c} - W\mathbf{u}\|_2^2$$

- W : matriz diagonal de pesos
- P : matriz de avaliação da base polinomial
- Solução via decomposição QR:

$$WP = Q \begin{bmatrix} R \\ 0 \end{bmatrix} = [Q_{\parallel} \quad Q_{\perp}] \begin{bmatrix} R \\ 0 \end{bmatrix}$$

- Coeficientes ótimos:

$$\mathbf{c}(\mathbf{x}) = R^{-1} Q_{\parallel}^T W \mathbf{u}$$



Cálculo dos Pesos de Interpolação

Pesos w_i

$$U(\mathbf{x}) = \sum_{i=1}^m w_i(\mathbf{x}) u(\mathbf{x}_i)$$

$$\mathbf{w} = W Q_{\parallel} R^{-1} \Phi(\mathbf{x})$$

Propriedades Importantes

- Partição da unidade: $\sum_{i=1}^m w_i = 1$
- Recuperação exata de funções de base
- Invariância sob transformações rígidas (para polinômios completos)

Algoritmo MLS



Algoritmo 1: Cálculo dos coeficientes w_i

1. Dados: $\mathbf{x}_1, \dots, \mathbf{x}_m, \mathbf{x} \in \mathbb{R}^d, \Phi_1, \dots, \Phi_n$
2. Calcule $W = [\delta_{ij}/\lambda_i(\mathbf{x})], P = [\Phi_j(\mathbf{x}_i)]$
3. Decomposição QR de $WP \rightarrow R$ e Q_{\parallel}
4. Calcule $\Phi = (\Phi_1(\mathbf{x}), \dots, \Phi_n(\mathbf{x}))^T$
5. Resolva $R^T \mathbf{d} = \Phi$
6. Resolver o sistema triangular $R^T \mathbf{d} = \Phi \Rightarrow \mathbf{d} = R^{-T} \Phi$
7. Calcule $\mathbf{w} = W Q_{\parallel} \mathbf{d}$



Aspectos Práticos

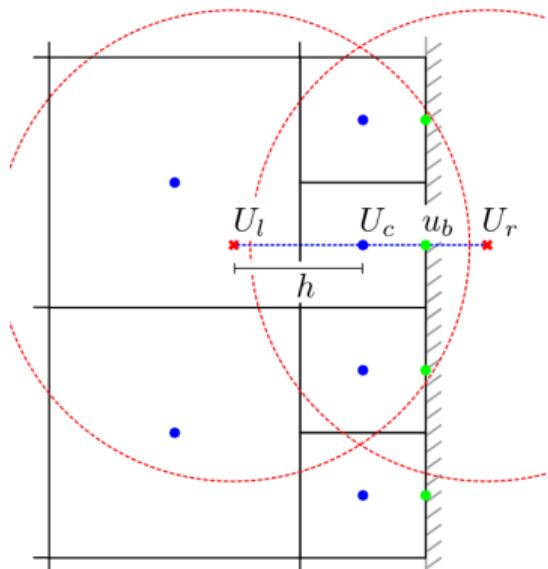
- **Raio de interpolação:** inicialmente $3 \times \max\{\delta_1, \dots, \delta_d\}$
- **Adaptatividade:** aumento do raio se R for singular
- **Cache:** pesos podem ser armazenados para malhas estáticas
- **Paralelização:** computações locais e independentes

Matrizes Resultantes

- Esparsas
- Não simétricas
- Solucionadores iterativos recomendados: GMRES, Bi-CG



Condições de Contorno Dirichlet

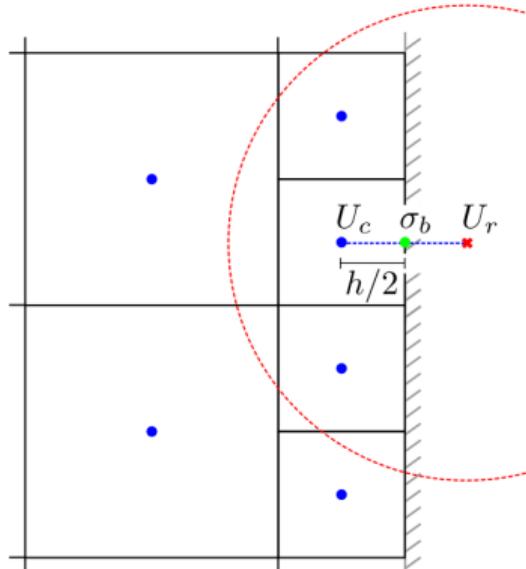


- Valores conhecidos nos pontos de fronteira
- Incorporados nas interpolações MLS
- Similar a técnicas FD padrão
- u_b imposto nos pontos verdes

Figura 11: Condição Dirichlet



Condições de Contorno Neumann



- $\frac{\partial u}{\partial x} = \sigma_b$ na fronteira direita
- Discretização por diferenças finitas:

$$U_r = U_c + h\sigma_b$$

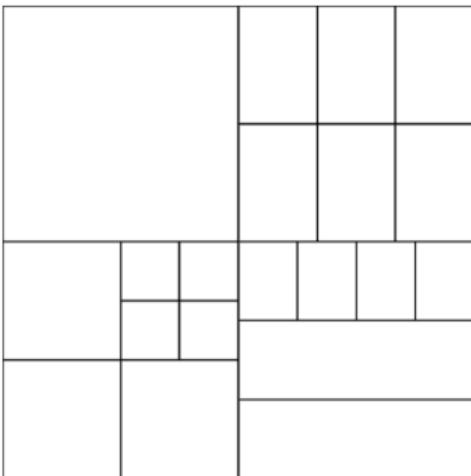
- h : tamanho das células menores
- Modificação do estêncil em U_c

Figura 12: Condição Neumann

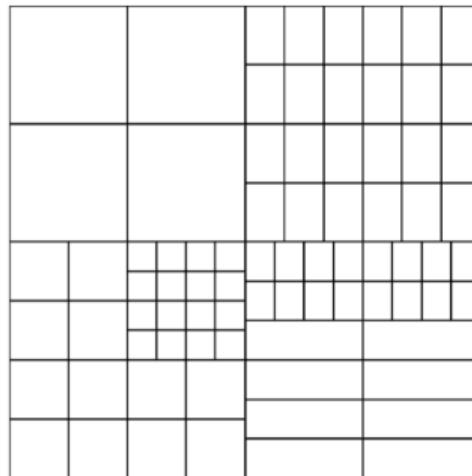


Verificação: Equação de Poisson em HiG Geral

- Domínio: $\Omega = [-1, 1]^2$
- Malha geral com "hanging nodes"
- Solução analítica: $u(x, y) = \sin(x) \cos(y)$
- Polinômios de 2º grau para MLS



(a) $h = 1/4$



(b) $h = 1/8$



Resultados de Convergência - Poisson

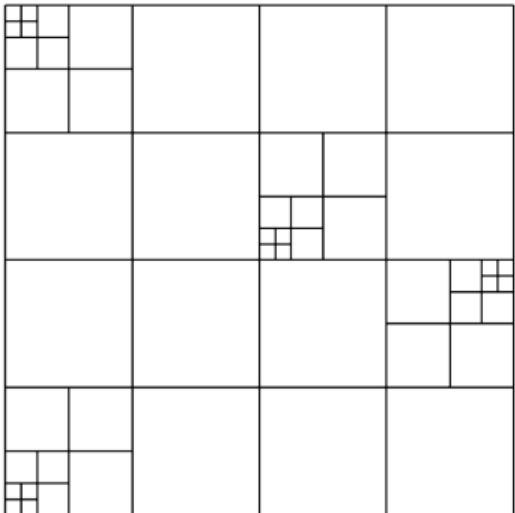
Tabela 1: Erros e ordem de convergência para Poisson em HiG geral

h	$\ u - u_h\ _{L_2}$	EOC	$\ u - u_h\ _{L_\infty}$	EOC
1/4	1.29e-01	–	1.21e-01	–
1/8	7.06e-03	4.1	8.66e-03	3.8
1/16	1.30e-03	3.3	1.66e-03	3.1
1/32	2.32e-04	3.0	3.14e-04	2.9

- Convergência de 2^a ordem confirmada
- Robustez mesmo com muitas interfaces não coincidentes
- Uso intensivo de interpolações em todas as interfaces



Malhas Non-Graded Quadtree



- Domínio: $[0, \pi]^2$
- Solução exata: $u(x, y) = e^{-x-y}$
- Condições Dirichlet
- Teste de convergência em malha não-graded
- Comparação com Min et al. [14] e Batty [12]

Figura 14: Malha non-graded para teste de Poisson



Resultados - Polinômios de 2º Grau

Tabela 2: Erros com interpolações de 2º grau

h	$\ u - u_h\ _{L_\infty}$	EOC	$\ \nabla u - \nabla u_h\ _{L_\infty}$	EOC
$\pi/32$	2.68e-2	–	2.57e-2	–
$\pi/64$	6.08e-3	2.1	1.07e-2	1.3
$\pi/128$	1.15e-3	2.3	3.18e-3	1.5
$\pi/256$	2.39e-4	2.3	1.03e-3	1.5

- 2ª ordem para a solução u_h
- Ordem reduzida (1.5) para o gradiente ∇u_h



Resultados - Polinômios de 3º Grau

Tabela 3: Erros com interpolações de 3º grau

h	$\ u - u_h\ _{L_\infty}$	EOC	$\ \nabla u - \nabla u_h\ _{L_\infty}$	EOC
$\pi/32$	1.08e-2	–	7.01e-3	–
$\pi/64$	1.80e-3	2.6	2.36e-3	1.6
$\pi/128$	3.18e-4	2.5	6.96e-4	1.7
$\pi/256$	7.65e-5	2.4	1.70e-4	1.8

- Melhora na ordem de convergência
- 2ª ordem para solução e gradiente com polinômios de ordem superior

Conclusões Parciais



Vantagens do Método Proposto

- **Robusto:** lida com malhas gerais hierárquicas
- **Flexível:** não requer interpolações geométricas específicas
- **Preciso:** mantém ordem de convergência esperada
- **Generalizável:** extensão direta para 3D e dimensões superiores
- **Eficiente:** computações locais, possibilidade de cache

Nossos sinceros agradecimentos ao Daniel. Sua ajuda foi indispensável para a realização deste trabalho



Muito Obrigado Pela Atenção

Dúvidas e Sugestões

Erick Rodrigues Canterle,
pedro.coimbra@usp.br,
queiroztff@usp.br ou
desaferreira@usp.br

Referências I

-  CASTILLO-SÁNCHEZ, H. A. et al. Numerical simulation of thixotropic-viscoelastic models for structured fluids in hierarchical grids. *Computers; Fluids*, Elsevier BV, v. 266, p. 106045, nov. 2023. ISSN 0045-7930.
-  RAYMUNDO, D. G. L. *Simulações numéricas de escoamentos bifásicos eletroosmóticos viscoelásticos em malhas hierárquicas*. Tese (Doutorado) – Universidade de São Paulo. Agência de Bibliotecas e Coleções Digitais, 2024.
-  SOUSA, F. et al. A finite difference method with meshless interpolation for incompressible flows in non-graded tree-based grids. *Journal of Computational Physics*, Elsevier BV, v. 396, p. 848–866, nov. 2019. ISSN 0021-9991.

