LUIZ FELIPE MARCHETTI DO COUTO

ARQUITETURA DE COMPUTAÇÃO PARALELA PARA RESOLUÇÃO DE PROBLEMAS DE DINÂMICA DOS FLUIDOS E INTERAÇÃO FLUIDO-ESTRUTURA

São Paulo

2016

LUIZ FELIPE MARCHETTI DO COUTO

ARQUITETURA DE COMPUTAÇÃO PARALELA PARA RESOLUÇÃO DE PROBLEMAS DE DINÂMICA DOS FLUIDOS E INTERAÇÃO FLUIDO-ESTRUTURA

Dissertação apresentada à Escola Politécnica da Universidade de São Paulo para obtenção do título de Mestre em Engenharia Civil

Área de Concentração: Engenharia Civil

Orientador: Prof. Dr. Paulo de Mattos Pimenta

São Paulo

2016

Catalogação-na-publicação

Couto, Luiz Felipe Marchetti do
Arquitetura de computação paralela para resolução de problemas de dinâmica dos fluidos e interação fluido-estrutura / L. F. M. Couto -- São Paulo, 2016.
238 p.

Dissertação (Mestrado) - Escola Politécnica da Universidade de São Paulo. Departamento de Engenharia de Estruturas e Geotécnica.

1.Multiprogramação e multiprocessamento 2.Computação gráfica 3.Interação fluido-estrutura 4.Dinâmica dos fluidos 5.Método dos elementos finitos I.Universidade de São Paulo. Escola Politécnica. Departamento de Engenharia de Estruturas e Geotécnica II.t.

LUIZ FELIPE MARCHETTI DO COUTO

ARQUITETURA DE COMPUTAÇÃO PARALELA PARA RESOLUÇÃO DE PROBLEMAS DE DINÂMICA DOS FLUIDOS E INTERAÇÃO FLUIDO-ESTRUTURA

Dissertação apresentada à Escola Politécnica da Universidade de São Paulo para obtenção do título de Mestre em Engenharia Civil

Banca examinadora:

Prof. Dr. Paulo de Mattos Pimenta
Orientador
Escola Politécnica da Universidade de São Paulo

Prof. Dr. Henrique Campelo Gomes
Escola Politécnica da Universidade de São Paulo

Prof. Dr. José Luis Drummond Alves
Universidade Federal do Rio de Janeiro

São Paulo, 02 de Junho de 2016.

À minha amada esposa Roberta, pela paciência e incentivo por todos estes anos.
Ao meu filho Felipe, o grande amor da minha vida.
Aos meus pais Celso e Marcia, pelo apoio incondicional.

AGRADECIMENTOS

Aos familiares, pelo apoio incondicional, compreensão e carinho.

Ao Prof. Dr. Paulo de Mattos Pimenta pelo apoio e confiança desde o início deste trabalho, pela amizade, pelas produtivas reuniões, pelos grandes ensinamentos nestes anos e por me apresentar ao mundo maravilhoso da Mecânica Computacional.

Ao Prof. Dr. Henrique Campelo Gomes pela amizade conquistada, pelas excelentes conversas e reuniões e "co-orientação", sempre trazendo novas ideias para este e futuros trabalhos e sem o qual nada disso seria possível.

Ao Prof. Dr. Alfredo Gay Neto pelas excelentes contribuições a este trabalho, pelo fornecimento do código do GIRAFFE, possibilitando a realização da co-simulação e proporcionado com isso a apresentação no Congresso de Mecânica Computacional (USNCCM) em San Diego, EUA.

Ao grupo do Laboratório de Mecânica Computacional, Campello, Edu, Paulo, Yuri, Jorginho, pelas sempre produtivas conversas, tornando cada dia neste Laboratório um grande aprendizado.

Ao Prof. Januário Pellegrino Neto pela amizade, confiança e principalmente pela indicação na pós-graduação da Escola Politécnica da Universidade de São Paulo, possibilitando conhecer o Laboratório de Mecânica Computacional e o excelente corpo técnico que o mesmo possui.

RESUMO

Um dos grandes desafios da engenharia atualmente é viabilizar soluções computacionais que reduzam o tempo de processamento e forneçam respostas ainda mais precisas. Frequentemente surgem propostas com as mais diversas abordagens que exploram novas formas de resolver tais problemas ou tentam, ainda, melhorar as soluções existentes. Uma das áreas que se dedica a propor tais melhorias é a computação paralela e de alto desempenho – HPC (*High Performance Computing*). Técnicas que otimizem o tempo de processamento, algoritmos mais eficientes e computadores mais rápidos abrem novos horizontes possibilitando realizar tarefas que antes eram inviáveis ou levariam muito tempo para serem concluídas. Neste projeto propõe-se a implementação computacional de uma arquitetura de computação paralela com o intuito de resolver, de forma mais eficiente, em comparação com a arquitetura sequencial, problemas de Dinâmica dos Fluidos e Interação Fluido-Estrutura e que também seja possível estender esta arquitetura para a resolução de outros problemas relacionados com o Método dos Elementos Finitos. O objetivo deste trabalho é desenvolver um algoritmo computacional eficiente em linguagem de programação científica C++ e CUDA – de propriedade da NVIDIA® – tendo como base trabalhos anteriores desenvolvidos no LMC (Laboratório de Mecânica Computacional) e, posteriormente, com a arquitetura desenvolvida, executar e investigar problemas de Dinâmica dos Fluidos e Interação Fluido-Estrutura (aplicando o método dos Elementos Finitos com Fronteiras Imersas e a solução direta do sistema de equações lineares com PARDISO) com o auxílio dos computadores do LMC. Uma análise de sensibilidade para cada problema é realizada de forma a encontrar a melhor combinação entre o número de elementos da malha de elementos finitos e o *speedup*, e posteriormente é feita uma análise comparativa de desempenho entre a arquitetura paralela a sequencial. Com uma única GPU conseguiu-se uma considerável redução no tempo para o *assembly* das matrizes globais e no tempo total da simulação.

Palavras-chaves: Interação Fluido-Estrutura, Elementos Finitos, High Performance Computing, CUDA.

ABSTRACT

One of the biggest challenges of engineering is enable computational solutions that reduce processing time and provide more accurate numerical solutions. Proposals with several approaches that explore new ways of solving such problems or improve existing solutions emerge. One of the biggest areas dedicated to propose such improvements is the parallel and high performance computing. Techniques that improve the processing time, more efficient algorithms and faster computers open up new horizons allowing to perform tasks that were previously unfeasible or would take too long to complete. We can point out, among several areas of interest, Fluid Dynamics and Interaction Fluid-Structure. In this work it is developed a parallel computing architecture in order to solve numerical problems more efficiently, compared to sequential architecture (e.g. Fluid Dynamics and Fluid-Structure Interaction problems) and it is also possible to extend this architecture to solve different problems (e.g. Structural problems). The objective is to develop an efficient computational algorithm in scientific programming language C ++, based on previous work carried out in Computational Mechanics Laboratory (CML) at Polytechnic School at University of São Paulo, and later with the developed architecture, execute and investigate Fluid Dynamics and Fluid-Structure Interaction problems with the aid of CML computers. A sensitivity analysis is executed for different problems in order to assess the best combination of elements quantity and speedup, and then a perfomance comparison. Using only one GPU, we could get a 10 times speedup compared to a sequential software, using the Finite Element with Immersed Boundary Method and a direct solver (PARDISO).

Key-words: Fluid-Structure Interaction, Finite Elements, High Performance Computing, CUDA.

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

Tem-se como paradigma da engenharia nos tempos atuais a viabilização de soluções computacionais que otimizem o tempo de processamento e resultem em soluções mais precisas. Constantemente surgem propostas com as mais diversas abordagens que exploram novas formas de resolver tais problemas ou tentam, ainda, melhorar as soluções existentes. Uma das grandes áreas que se dedica a propor tais melhorias é a computação paralela e de alto desempenho – HPC (*High Performance Computing*).

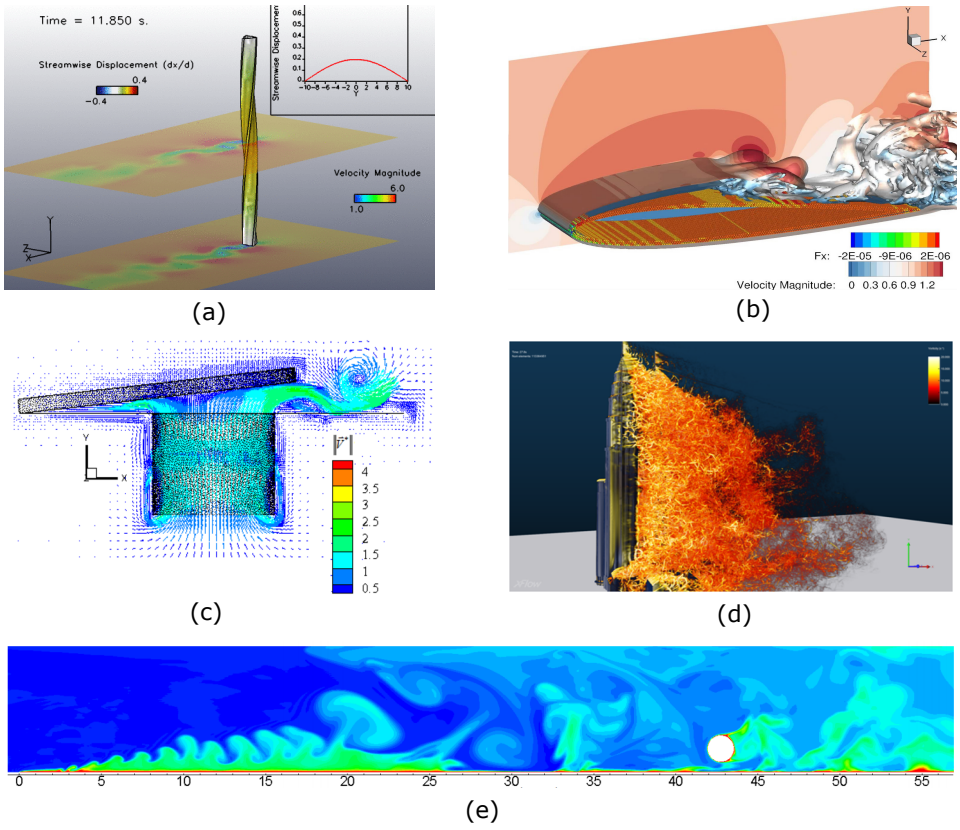
Técnicas que otimizem o tempo de processamento como algoritmos mais eficientes e computadores mais rápidos abrem novas possibilidades a realizar tarefas que antes eram inviáveis. Podemos citar, por exemplo, a dinâmica de fluidos e interação fluido-estrutura.

Segundo Gomes (2013), a interação fluido-estrutura (IFE) é um problema que envolve a Mecânica dos Fluidos e das Estruturas sendo que a solução de um domínio depende da solução do outro. Caracterizando-se, assim, um sistema acoplado. Se resolver um problema de mecânica dos fluidos com condições de contorno diversas já é uma tarefa difícil, um problema de IFE acrescenta como desafio a solução simultânea do sistema acoplado onde as condições de contorno na interface entre o fluido e a estrutura são desconhecidas a priori, pois dependem da solução do problema. Além disso, grande parte dos problemas de interesse para a engenharia envolve grandes deslocamentos da estrutura e convecção do fluido, sendo a IFE, por natureza, um problema fortemente não linear.

Ainda de acordo com Gomes (2013), uma dificuldade em simular computacionalmente problemas desta natureza está na alta velocidade do escoamento do fluido, que normalmente requer modelos complexos de turbulência (GAMNITZER, 2010). Adicione também o fato de que o tempo de simulação, em geral, precisa ser grande para despertar o fenômeno físico e, a incrementos de tempo pequenos, exigência de qualquer simulação numérica de escoamento de fluidos.

Esta combinação, naturalmente, demanda computação de alto desempenho que, muitas vezes, favorece a utilização de métodos para redução da ordem do problema (LIEU et al., 2006) e a utilização de computação paralela. As Figuras 1 (a), (b), (c), (d) e (e) mostram os resultados da simulação computacional de problemas de dinâmica dos fluidos e interação fluido-estrutura utilizando o método das Fronteiras Imersas e paralelizado na GPU.

Figura 1.1 – Exemplos de problemas de interação fluido-estrutura.



Fonte: (a) Análise da deformação de um riser (ACUSIM *software*) (b) Análise de turbulência de um aerofólio NACA 0012 (Cardiff *University Hydraulics*) (c) (GASCHE et al., 2012) (d) Aerodinâmica de edifícios (MANTIUM CAE) (e) (KANG et al., 2009)

1.1 Organização do trabalho

Este trabalho encontra-se organizado da seguinte forma:

No Cap. 2 é apresentado um breve resumo histórico da computação sequencial, evoluindo para a computação paralela e a utilização de placas gráficas para resolução de problemas numéricos. É apresentado os tipos de memórias disponíveis na GPU e suas características, o funcionamento da memória do tipo *shared* e como é realizado o *assembly* das matrizes globais utilizando a linguagem de programação CUDA.

O Cap. 3 apresenta a teoria da interação fluido-estrutura utilizando elementos finitos e o método das Fronteiras Imersas. Um novo método de integração para os elementos "cortados", como uma opção ao método do *Tessellation* é abordado.

O Cap. 4 aborda a teoria de vigas tridimensionais cinematicamente exatas para a solução de problemas que envolvem *risers*, sendo posteriormente utilizado para a co-simulação envolvendo dois *softwares* desenvolvidos no Laboratório de Mecânica Computacional da Escola Politécnica da Universidade de São Paulo, HPC_LMC e GIRAFFE.

No Cap. 5 é apresentado a implementação computacional do *software* desenvolvido

neste trabalho, apresentando os principais pacotes e classes, e uma breve apresentação do código paralelizado na GPU (No Apêndice A, encontra-se o código completo da paralelização utilizando a linguagem de programação CUDA).

O Cap. 6 trata de experimentos realizados por meio da simulação numérica de diferentes problemas de dinâmica dos fluidos e interação fluido-estrutura utilizando o método das Fronteiras Imersas, uma análise de sensibilidade para cada problema de modo a encontrar o melhor custo benefício em relação ao número de elementos em uma malha de elementos finitos e o *speedup*, e posteriormente uma comparação dos resultados com a literatura, e uma análise de desempenho.

Finalmente, o Cap. 7 apresenta as conclusões deste trabalho, e possíveis trabalhos futuros que podem ser realizados a partir deste.

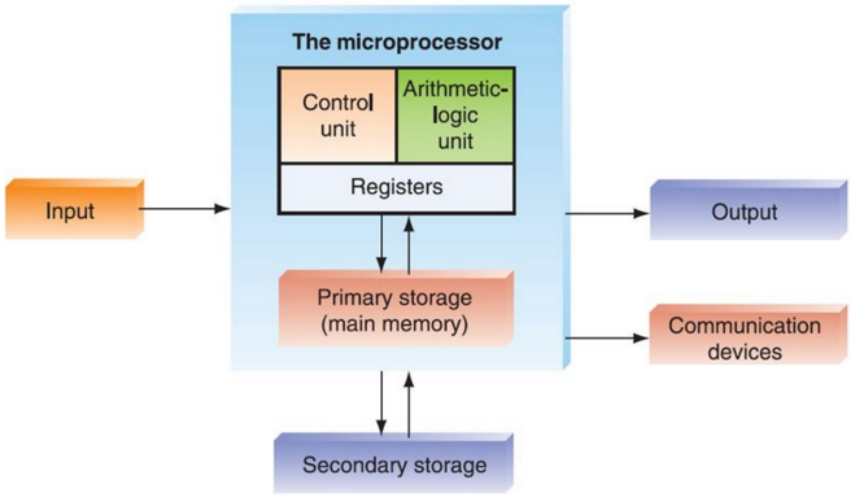
2 COMPUTAÇÃO PARALELA

Este capítulo apresenta uma breve história da computação sequencial e como foi possível utilizar diversas CPUs para a execução em paralelo. Apresenta o funcionamento básico dos diferentes tipos de memória disponíveis na GPU, o *assembly* das matrizes locais utilizando a tecnologia CUDA de propriedade da NVIDIA e o formato adequado para envio dos dados da CPU para a GPU, como os dados nodais, graus de liberdade, entre outros.

2.1 Computação sequencial

A computação sequencial tem sido utilizada por mais de 60 anos, desde John Von Neumann quando criou a computação digital nos anos 50. Ela é definida como um sistema que possui uma unidade de processamento chamada de CPU (*Central Processing Unit*[1]) e uma unidade de memória (Fig. 2.1 e Fig. 2.2). A velocidade de processamento de qualquer aplicação depende principalmente da taxa de execução das instruções, em ciclos por segundo (*clock*), e a taxa de transferência (*bandwidth*) entre a memória e a CPU.

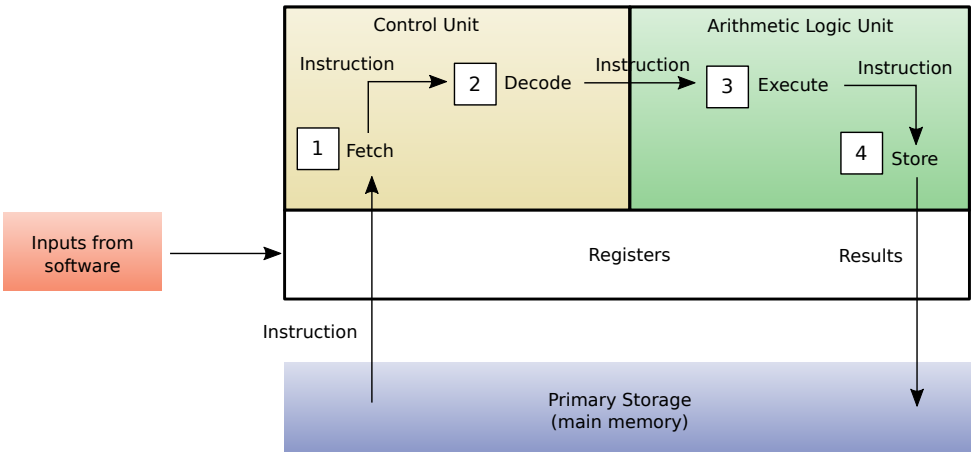
Figura 2.1 – Microprocessador.



Fonte: Disponível em: <http://www.csb.uncw.edu>.

[1]A CPU (*Central Processing Unit*) ou Unidade Central de Processamento, mais conhecido como processador, é composta pela ALU (*Arithmetic Logic Unit*), ou Unidade Lógica de Aritmética responsável pela operações matemáticas, lógicas e operações de decisão, e a CU (*Control Unit*), ou Unidade de Controle, que direciona todas as operações do processador.

Figura 2.2 – Funcionamento de um microprocessador.



Fonte: Disponível em: <http://www.csb.uncw.edu>.

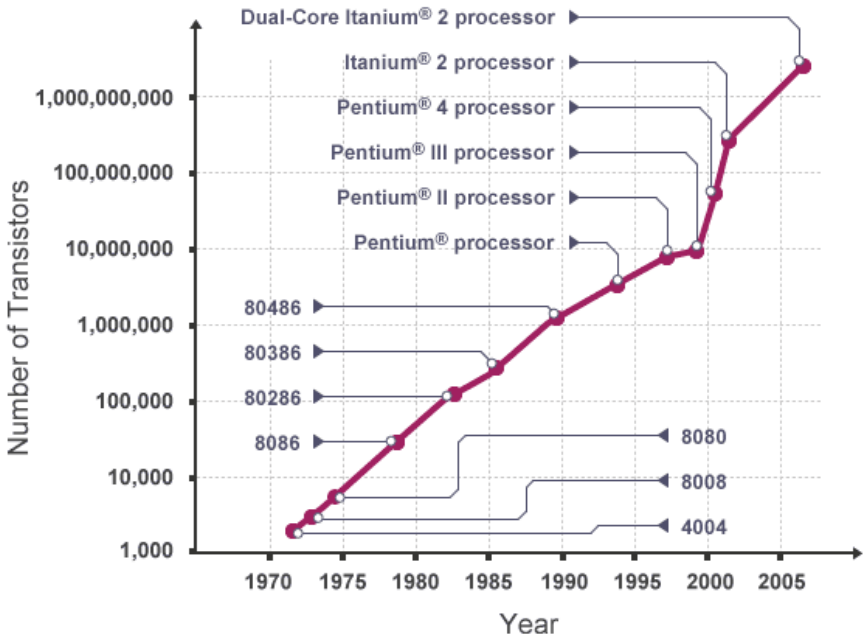
2.1.1 Gordon E. Moore

A Lei de Moore foi estabelecida por Gordon Earl Moore e diz que o número de transístores em um chip de um microprocessador dobrará a cada dois anos.

> "The complexity for minimum component costs has increased at a rate of roughly a fator of two per year. Certainly over the short term this rate can be expected to continue."

A Fig. 2.3 mostra a linha de tendência dos computadores ao longo dos anos (1970 a 2005).

Figura 2.3 – Lei de Moore.



Fonte: Disponível em: <http://www.bbc.co.uk/education/guides/z46s4wx/revision/6>.

Os microprocessadores baseados em um único CPU (i.e. família Intel Pentium e AMD) obtiveram um crescimento exponencial em velocidade de processamento e redução de custos nas últimas décadas. As CPUs chegaram a GFLOPs (*Giga Floating-point Operations per Second*) em desktops comuns e a centenas de GFLOPs em servidores em *cluster*[2].

2.2 Computação paralela

Os desenvolvedores acreditavam que o avanço em hardware levariam a um aumento de velocidade de processamento e execução dos softwares. Entretanto este crescimento tem diminuído desde 2003 devido ao alto consumo de energia e problemas de dissipação de calor nos processadores, limitando assim o aumento da frequência de *clock* e a eficiência em cada período de *clock* em uma única CPU.

Desta forma os fabricantes e produtores de microprocessadores mudaram a tecnologia interna da CPU para que pudessem ser utilizadas unidades de múltiplo processamento (*Multi-core processors*[3]). Esta mudança no *hardware* teve um grande impacto na comunidade de desenvolvedores de *softwares*. Isto porque tradicionalmente os aplicativos são desenvolvidos como programas sequenciais, como descrito por Von Neumann em 1945, e historicamente os usuários estão acostumados a esperar destes programas um avanço em performance com cada geração nova de *hardware* lançada no mercado.

2.2.1 Lei de Amdahl

Para que se possa compreender melhor esta lei é necessário apresentar alguns conceitos relativos a análise e métricas de *performance* ou desempenho de um *hardware* ou *software*.

2.2.1.1 *Latency* vs. *Throughput*

Os seguintes conceitos são importantes e muitas vezes contraditórios e são utilizados para decidir qual a melhor abordagem a ser aplicada a um determinado *hardware* ou *software*:

- *Latency* ou tempo de execução é o tempo necessário para que uma determinada tarefa leva para ser concluída. É medido em unidades de tempo ou períodos de *clock*.
- *Throughput* ou *bandwidth* é o número de tarefas que podem ser executadas em um determinado tempo. É medido em unidades de algo que está sendo produzido (I/O[4],

[2] O *cluster* consiste na interligação de diversos computadores para que juntos possam executar processamentos em paralelo. Foi desenvolvido na década de 60 pela IBM como uma forma de interligar mainframes e fornecer aos clientes uma solução viável comercialmente para processamentos paralelizados.

[3] Uma unidade de múltiplo processamento é um componente com duas ou mais unidades de processamento independentes.

[4] *Inputs* ou *outputs*.

iterações, memória transferida) por unidade de tempo.

Alguns exemplos:

- Frequência de *clock*: 100 MHz
- *Throughput* de um dispositivo: 640 Mbits/second

2.2.1.2 *Speedup*

Speedup é uma métrica de desempenho para analisar o quanto um determinado sistema obteve de *performance*, superior ou inferior, em relação a outro determinado sistema.

Exemplo:

- Sistema A executa determinada tarefa em 200 ciclos;
- Sistema B executa determinada tarefa em 350 ciclos;
- $350/200 = 1.75$, ou seja, o sistema B é 1.75 vezes mais rápido que o sistema A.

Em Molyneaux (2009) pode ser encontrada uma análise extensa de uma série de métricas de desempenho que podem ser utilizadas tanto em *hardware* como em *software*, como também em redes de computadores, entre outras aplicações.

2.2.1.3 *Gene M. Amdahl*

Em Amdahl (1967), Gene M. Amdahl diz:

> "For over a decade prophets have voiced the contention that the organization of a single computer has reached its limits and that truly signicant advences can be made only by interconnection of a multiplicity of computers in such a manner as to permit cooperative solution. Variously the proper direction has been pointed out as general purpose computers with a generalized interconnection of memories or as specialized computers with geometrically related memory interconnections and controlled by one or more instruction streams.
>
> Demonstration is made of the continued validity of the single processor approach and of the weaknesses of the multiple processor approach in terms of application to real problems and their attendant irregularities."

Esta citação, que tornou-se conhecida como "Lei de Amdahl", é frequentemente utilizada na computação paralela para prever o máximo *speedup* teórico que é obtido quando utiliza-se múltiplos processadores. Esta lei mostra que, a não ser que o *software* (ou parte do *software*) seja 100% eficiente na utilização de múltiplos processadores, o sistema se beneficiará cada vez menos quando adiciona-se mais processadores.

A seguinte equação mostra o *speedup* global do sistema utilizando-se a "Lei de Amdahl":

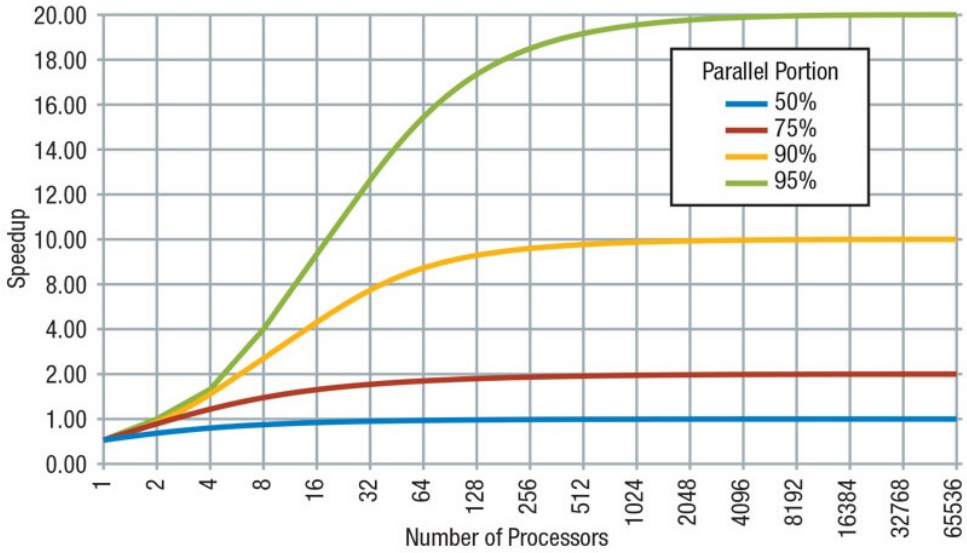
$$S(n) = \frac{1}{(1-P) + \frac{P}{n}} \tag{2.1}$$

onde:

- $S(n)$ é o *speedup* teórico;
- P é a fração do algoritmo que pode ser paralelizado;
- n é o número de processadores ou *threads*[5] utilizadas no processamento.

A Fig. 2.4 mostra o *speedup* teórico de um sistema paralelizado com relação ao número de processadores ou *threads* utilizadas neste sistema. Pode-se perceber claramente que mesmo que haja um aumento do número de processadores ou *threads* que serão teoricamente direcionadas para o sistema, o *speedup* teórico não aumenta, limitando assim este sistema.

Figura 2.4 – Gráfico do *speedup* teórico pelo número de processadores ou *threads* utilizadas em um sistema em função da fração do código-fonte que pode ser efetivamente paralelizado.



Fonte: Disponível em <http://www.rtcmagazine.com/articles/view/103209>.

2.2.2 GPU

Nesta seção é apresentado um resumo dos diferentes tipos de memórias disponíveis na GPU. Na seção 2.2.2.2.3 é descrito o funcionamento do *assembly* das matrizes globais

[5] *Thread* é um processo que pode ser dividido em tarefas que são executadas concorrencialmente ou pelo sistema operacional (neste caso o processo é considerado como virtual) ou pelo *hardware* (neste caso o processo é físico).

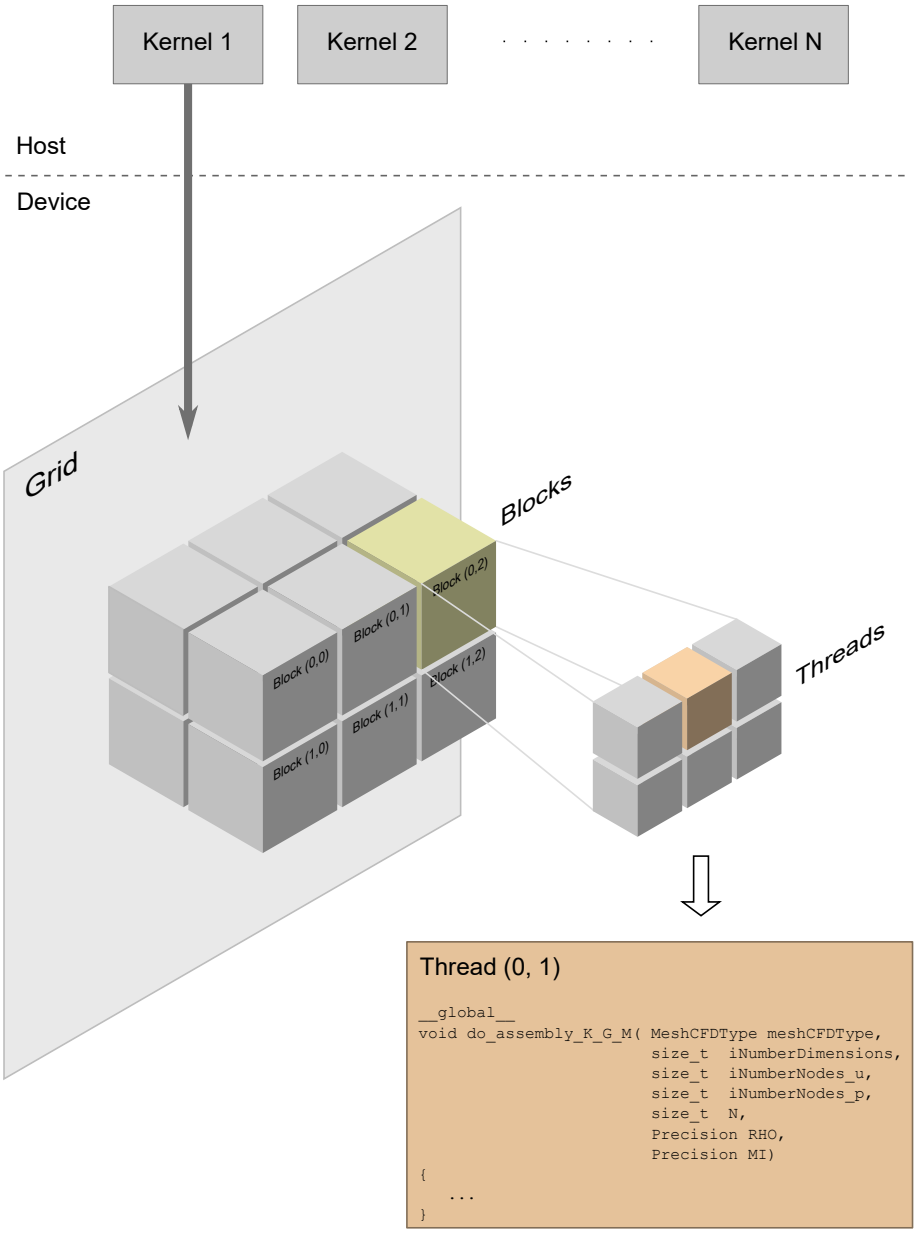
utilizadas no *software* desenvolvido neste trabalho. Em Farber (2011) pode-se encontrar uma completa descrição de todas as memórias disponíveis na GPU, como utilizá-las da melhor forma e alcançar o melhor desempenho nas aplicações GPGPU[6] e alguns exemplos que utilizam a linguagem de programação CUDA.

A Fig. 2.5 ilustra o funcionamento de um *kernel* e um exemplo de chamada na linguagem de programação CUDA, onde:

- *Kernel*: Processo a ser executado na GPU;
- *Host*: CPU;
- *Device*: GPU;
- *Grid*: Conjunto de blocos de *threads*;
- *Blocks*: Conjunto de *threads*;
- *Threads*: Execução de um *kernel* em paralelo.

[6]GPGPU (*General Purpose Graphics Processing Unit*): é a utilização de uma GPU (*Graphics Processing Unit*) ou placa gráfica não apenas para realizar tarefas de renderização gráfica, mas também para processamento de imagem, inteligência artificial, cálculo numérico, entre outras aplicações – normalmente executadas na CPU.

Figura 2.5 – *Grid* da GPU.



Fonte: O autor

2.2.2.1 Memória da GPU

As aplicações GPGPU têm disponíveis tanto memórias internas ao microprocessador da placa gráfica, como na própria placa. As memórias mais rápidas e mais escaláveis são as memórias do tipo *shared*. Sua única limitação é o tamanho disponível para armazenamento de informação (alguns KB) e somente as *threads* dentro de um bloco podem acessá-la.

A memória global é um sistema de memórias compartilhadas que podem ser acessadas por todas as *threads* da GPU. O tamanho disponível para armazenamento de informação, normalmente, é medido em GB, o que a torna a maior memória da GPU, mais

utilizada, mas a mais lenta.

A Tab. 2.1 mostra o *bandwidth* de diferentes memórias disponíveis na GPU.

Tabela 2.1 – *Bandwidth* das diferentes memórias disponíveis na GPU (FARBER, 2011).

Register Memory	$\approx 8000\,\mathrm{GB/s}$
Shared Memory	$\approx 1600\,\mathrm{GB/s}$
Global Memory	$170\,\mathrm{GB/s}$
Mapped Memory	$\approx 8\,\mathrm{GB/s}$ (unidirecional)

Pela Tab. 2.1 pode-se observar que a memória global deve ser muito bem utilizada para que se obtenha a máxima performance da aplicação GPGPU. Da mesma forma, as informações que serão armazenadas na memória compartilhada devem ser bem dimensionadas para que caibam nos poucos KB disponíveis.

2.2.2.2 Tipos de memórias

A Tab. 2.2 apresenta os diferentes tipos de memórias disponíveis na GPU e suas características.

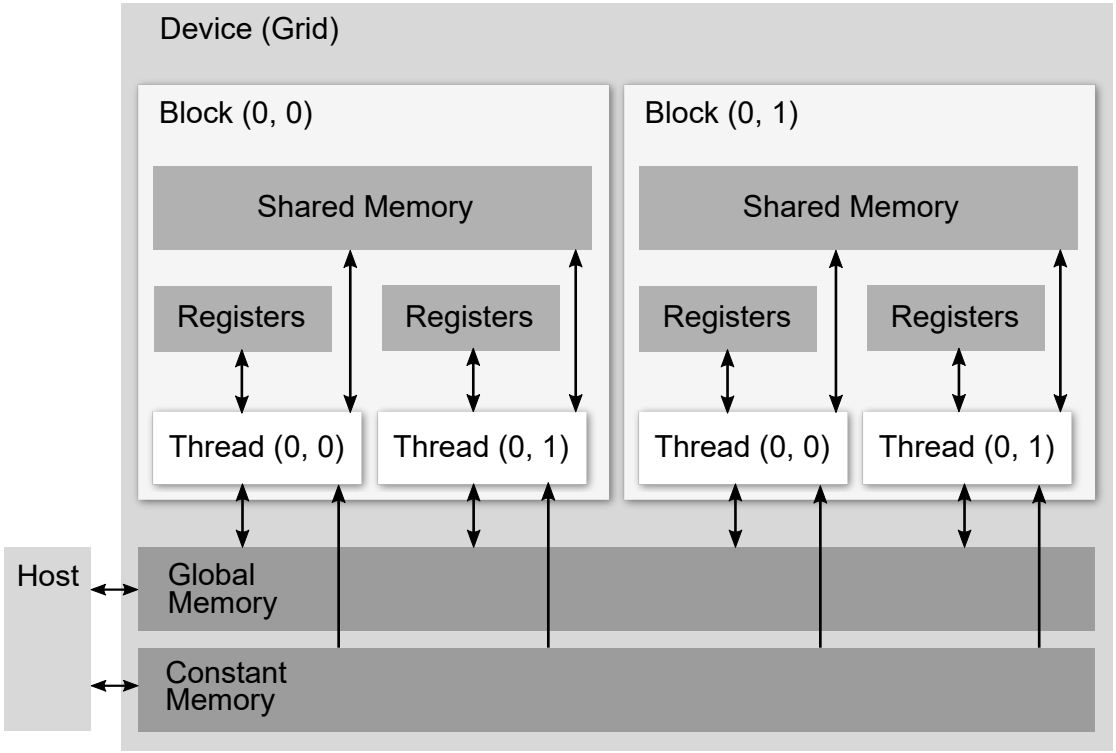
Tabela 2.2 – Tipos de memórias disponíveis na GPU e suas características (FARBER, 2011).

Memória	Localização[7],[8]	Acesso	Escopo
Register	*On-chip*	Leitura/Escrita	Uma *thread*
Local	*On-chip*	Leitura/Escrita	Uma *thread*
Shared	*On-chip*	Leitura/Escrita	Todas as *threads* de um bloco
Global	*Off-chip*	Leitura/Escrita	Todas as *threads* + *host*[9]
Constant	*Off-chip*	Leitura	Todas as *threads* + *host*
Texture	*Off-chip*	Leitura/Escrita	Todas as *threads* + *host*

A Fig. 2.6 mostra um diagrama esquemático dos diferentes tipos de memórias e a forma de transferência entre elas.

[7] On-chip: Interna ao microprocessador da placa gráfica.
[8] Off-chip: Placa gráfica.
[9] CPU.

Figura 2.6 – Diagrama esquemático das memórias internas da GPU e as transferência de dados entre elas.



Fonte: O autor

2.2.2.2.1 *Registers*

As memórias do tipo *registers* são as mais rápidas da GPU e são as únicas que possuem *bandwidth* e *latency* capazes de fornecer a máxima performance às aplicações GPGPU. Cada *kernel*[10] pode acessar somente 63 memórias do tipo *registers*, limitando assim a sua utilização por aplicações GPGPU. Este valor pode variar de 63 a 21 dependendo do número de *threads* executadas em paralelo.

2.2.2.2.2 *Local*

A memória do tipo *local* é utilizada quando a informação que se deseja armazenar não cabe em uma memória do tipo *register*.

2.2.2.2.3 *Shared*

A memória do tipo *shared* pode armazenar tanto 16 KB quanto 48 KB por bloco de *threads* e são organizadas em 32 grupos de 32 bits. Idealmente, 32 *threads* poderão acessar a memória compartilhada em paralelo sem perda de desempenho. Mas infelizmente,

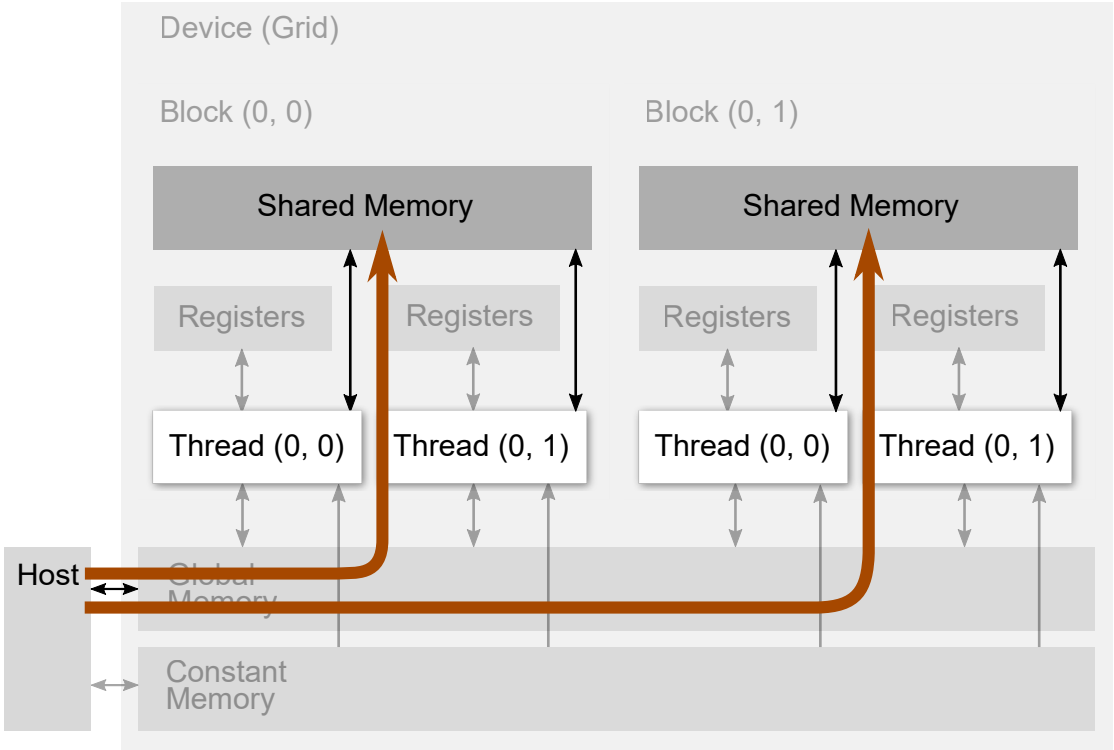
[10] *Kernel* é um programa executado dentro da GPU.

podem ocorrer conflitos de acesso quando múltiplas requisições são feitas por diferentes *threads* do mesmo bloco. Estas requisições podem ser tanto para o mesmo endereço ou para múltiplos endereços do mesmo grupo. Quando isto acontece, o *hardware* serializa as operações de memória, ou seja, se n *threads* acessam ao mesmo tempo o mesmo endereço de memória, então as n requisições serão realizadas sequencialmente, tornando o processo n vezes mais lento.

O grande desafio na utilização das memórias do tipo *shared* está no seu acesso. É necessário que as requisições sejam bem dimensionadas para que cada *thread* acesse o seu bloco de informação sem causar conflito de acesso e em uma única transação.

A Fig. 2.7 ilustra o acesso a esta memória a partir da CPU.

Figura 2.7 – Utilização da memória compartilhada para aumento do *speedup* de execução em paralelo.



Fonte: O autor

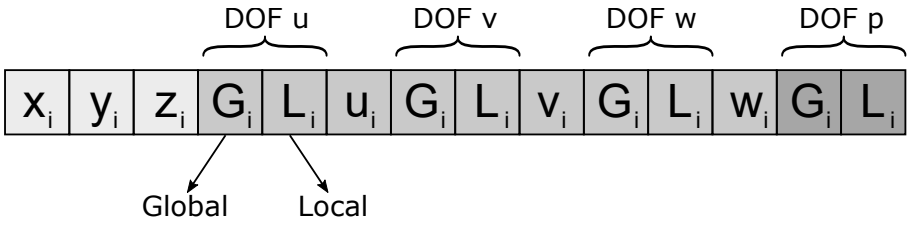
Neste trabalho utilizou-se este tipo de memória. Para que fosse evitado qualquer conflito de acesso, como descrito anteriormente, foi necessário definir as informações que seriam utilizadas pelo *assembly* das matrizes globais e de que forma cada *thread* iria acessar essas informações.

Para o *assembly* das matrizes globais são necessárias as seguintes informações para um problema tridimensional[11] (Fig. 2.8):

[11]Para um problema bidimensional é necessário retirar a coordenada z, os graus de liberdade globais e

- x, y e z: As coordenadas de cada nó da malha de elementos finitos;
- G, L: Os graus de liberdade globais e locais de cada nó da malha de elementos finitos;
- u, v, w: As velocidades dos nós da malha de elementos finitos nas direção x, y e z;
- p: A pressão dos nós da malha de elementos finitos.

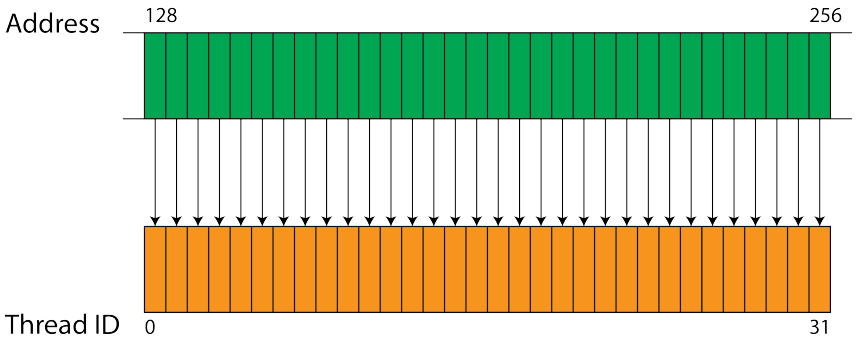
Figura 2.8 – Dados nodais para um problema tridimensional.



Fonte: O autor

Os dados nodais são enviados a memória compartilhada de forma contígua, assim, somente uma transação é necessária para que as 32 *threads* acessem as informações da memória compartilhada. Isto chama-se *memory coalescing*[12]. As Fig. 2.9, 2.10 e 2.11 ilustram estes acessos. O único acesso que será realizado em uma única transação é o sequencial e alinhado, e portanto, é o acesso que fornecerá o máximo desempenho.

Figura 2.9 – Acesso sequencial e alinhado.

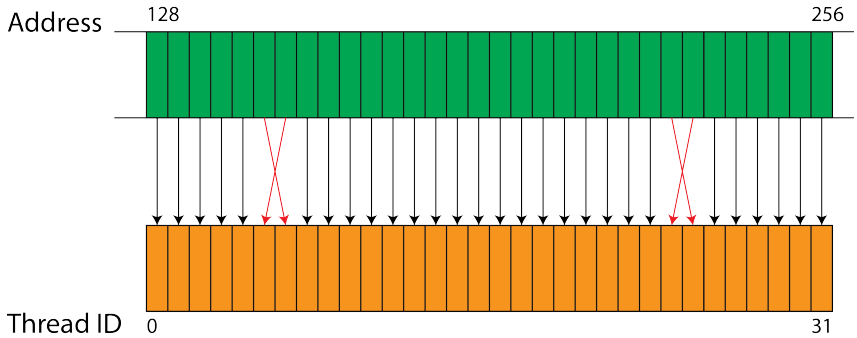


Fonte: Disponível em <https://cvw.cac.cornell.edu/gpu/coalesced?AspxAutoDetectCookieSupport=1>

locais e a velocidade na direção z.

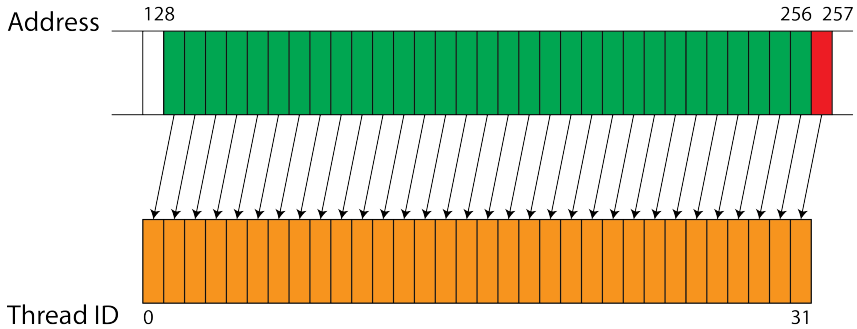
[12] *Memory coalescing*: combinar múltiplos acessos a memória em uma única transação.

Figura 2.10 – Acesso alinhado mas não sequencial.



Fonte: Disponível em <https://cvw.cac.cornell.edu/gpu/coalesced?AspxAutoDetectCookieSupport=1>

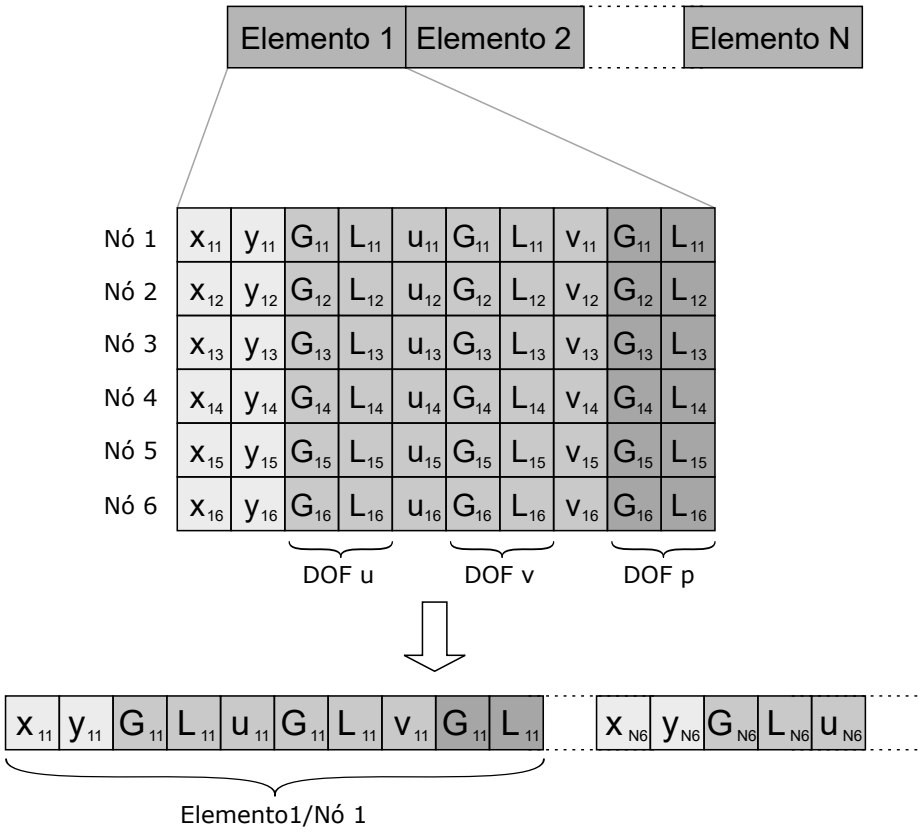
Figura 2.11 – Acesso desalinhado.



Fonte: Disponível em <https://cvw.cac.cornell.edu/gpu/coalesced?AspxAutoDetectCookieSupport=1>

A Fig. 2.12 ilustra a forma como os dados nodais são enviados para a memória compartilhada para posterior acesso contíguo.

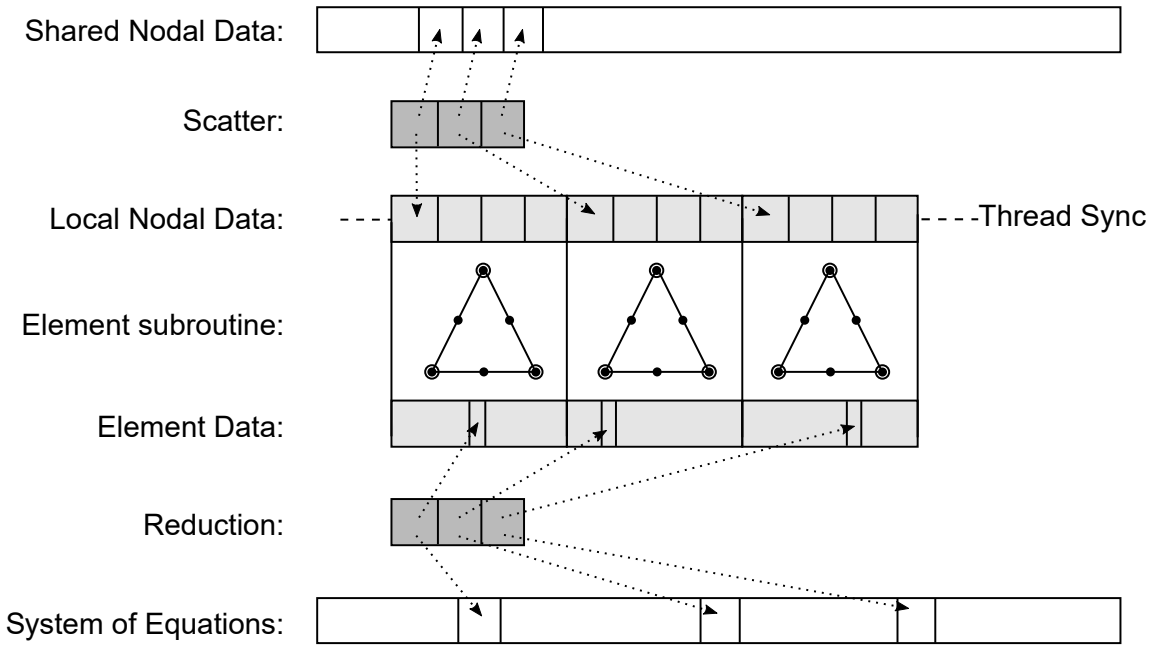
Figura 2.12 – Acesso contíguo aos dados nodais.



Fonte: O autor

A Fig. 2.13 apresenta um diagrama esquemático (adaptado de Cecka et al. (2011) para utilização neste trabalho) de como o *assembly* é realizado na GPU e posteriormente na CPU.

Figura 2.13 – *Assembly* das matrizes locais na GPU e posteriormente na CPU.



Fonte: O autor (adaptado de Cecka et al. (2011) para utilização neste trabalho)

O diagrama da Fig. 2.13 é executado da seguinte forma:

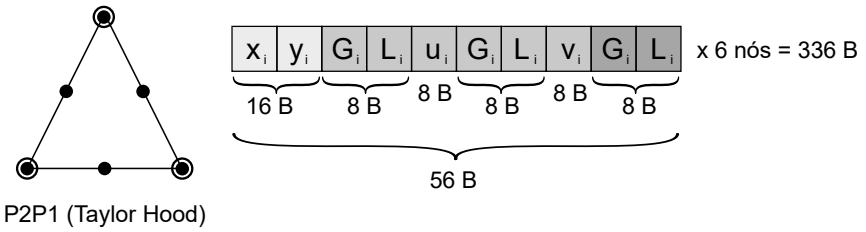
Tabela 2.3 – Algoritmo de *assembly* das matrizes globais.

Algoritmo *Assembly* das matrizes globais	
1:	***Shared Nodal Data*, *Scatter* e *Local Nodal Data***: Primeiramente o *kernel* separa as informações da memória do tipo *shared* em um *array* local de forma que cada thread tenha acesso a sua informação de forma contígua;
2:	***Thread Sync***: Realiza-se uma operação de sincronização[13];
3:	***Element subroutine***: Executam-se as subrotinas de cálculo dos elementos finitos (Cap. 3);
4:	***Element Data*, *Reduction* e *System of Equations***: Altera os dados da memória do tipo *shared* (valores dos graus de liberdade globais para as velocidades nas direções x, y e z e pressão) para que possa ser enviada para a memória do tipo *global* e posteriormente para a CPU para finalização do *assembly* das matrizes globais.

Para que se possa utilizar a memória compartilhada é necessário que os dados nodais caibam em 16 ou 48 KB. Cada elemento finito possui uma certa quantidade de nós e com isso uma certa quantidade de *bytes*. Utilizando-se precisão dupla são necessários 336, 504, 720 e 1944 B para os elementos do tipo P2P1, Q2Q1, tetraédrico de 10 nós e hexaédrico de 27 nós, respectivamente, como mostram as Fig. 2.14, 2.15, 2.16 e 2.17.

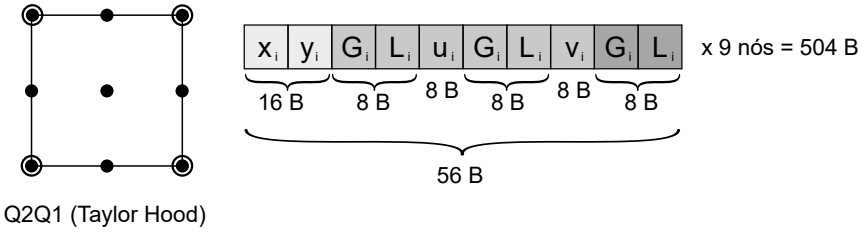
[13] A sincronização das *threads* garante que cada *thread* acesse as informações da memória do tipo *shared* de forma alinhada e sequencial, garantindo assim, uma única transação de memória.

Figura 2.14 – Quantidade de bytes necessários para um elemento triangular do tipo Taylor-Hood P2P1.



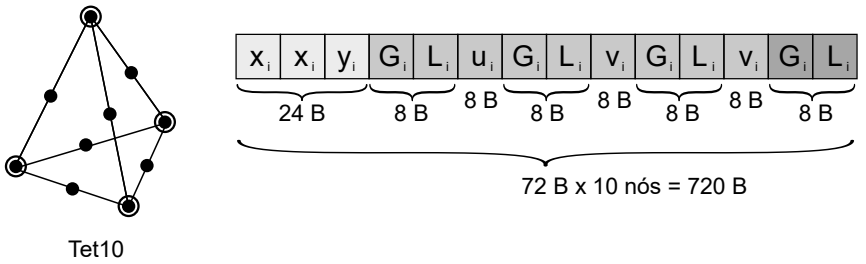
Fonte: O autor

Figura 2.15 – Quantidade de bytes necessários para um elemento quadrilateral do tipo Taylor-Hood Q2Q1.



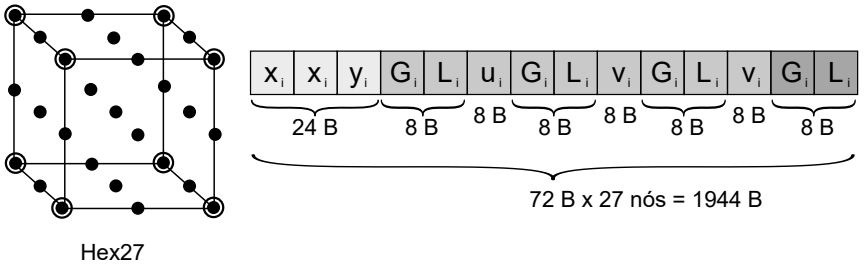
Fonte: O autor

Figura 2.16 – Quantidade de bytes necessários para um elemento do tipo tetraédrico de 10 nós.



Fonte: O autor

Figura 2.17 – Quantidade de bytes necessários para um elemento do tipo hexaédrico de 27 nós.



Fonte: O autor

Como exemplo, o número máximo de elementos finitos do tipo P2P1 que podem ser armazenados na memória do tipo *shared* em um problema de dinâmica dos fluidos ou interação fluido-estrutura utilizando o método das Fronteiras Imersas são: 49 152 B/336 B = 146 elementos. Como cada transação é realizada em grupos de 32, a situação ideal é armazenar 128 elementos[14].

2.2.2.2.4 *Constant*

A memória do tipo *constant* é excelente para armazenar informações que sejam somente leitura para e posteriormente enviar para todas as threads da GPU. A sua limitação é de 64 KB.

2.2.2.2.5 *Global*

A grande limitação da memória do tipo *global* está no seu *bandwidth*. É muito importante que se tenha em mente esta limitação para que se desenvolva aplicações GPGPU que utilizem a memória global somente quando for estritamente necessário, evitando assim a troca de informações entre a CPU e a GPU.

Algumas regras da memória do tipo *global*:

1. Envie as informações a mantenha na memória, sem enviar novamente para a CPU, o maior tempo possível;
2. Execute todos os *kernels*;
3. Tente reutilizar as informações armazenadas evitando as limitações de *bandwidth*.

Certamente não há como evitar o acesso a esta memória, mas é essencial que se tenha em mente as suas limitações.

Em Farber (2011) o autor apresenta diversas formas interessantes de evitar a limitação de *bandwidth* e outras técnicas para aumentar o desempenho das aplicações GPGPU.

[14] A mesma lógica pode ser aplicada aos outros elementos.

3 INTERAÇÃO FLUIDO-ESTRUTURA

Este capítulo apresenta as equações de Navier-Stokes e a teoria da interação fluido-estrutura utilizando o método das fronteiras imersas. A teoria completa pode ser encontrada em Gomes (2013).

3.1 Equações de Navier Stokes

Seja Ω_t em $\mathbb{R}^{n_{sd}}$ o domínio do espaço com contorno Γ_t no instante de tempo $t \in (0, T)$, onde T é o instante de tempo final do domínio. O subscrito t indica a dependência do domínio com o tempo. As equações de Navier-Stokes para escoamentos incompressíveis podem ser escritas em Ω e $\forall t \in (0, T)$ como

$$\rho\left(\frac{\partial \boldsymbol{u}}{\partial t} + \boldsymbol{u} \cdot \nabla \boldsymbol{u}\right) = \nabla \cdot \boldsymbol{T} + \rho \boldsymbol{b}, \tag{3.1}$$

$$\nabla \cdot \boldsymbol{u} = 0, \tag{3.2}$$

onde ρ, $\boldsymbol{u}$ e $\boldsymbol{b}$ são a densidade, velocidade e vetor de forças externas respectivamente. Para fluidos Newtonianos os tensores de tensão $\boldsymbol{T}$ e deformação $\epsilon(\boldsymbol{u})$ são definidos como

$$\boldsymbol{T} = -p\boldsymbol{I} + 2\mu\boldsymbol{\epsilon}(\boldsymbol{u}) \tag{3.3}$$

$$\boldsymbol{\epsilon}(\boldsymbol{u}) = \frac{1}{2}\left[\nabla \boldsymbol{u} + (\nabla \boldsymbol{u})^{\mathsf{T}}\right] \tag{3.4}$$

sendo a variável p a pressão, $\boldsymbol{I}$ o tensor identidade e μ a viscosidade dinâmica do fluido.

O divergente do tensor das tensões é dado por

$$\nabla \cdot \boldsymbol{T} = -\nabla p + \mu \nabla^2 \boldsymbol{u} + \mu \nabla(\nabla \cdot \boldsymbol{u}) \tag{3.5}$$

Como o escoamento é incompressível, o último termo é zero. Portanto

$$\frac{\partial \boldsymbol{u}}{\partial t} + \boldsymbol{u} \cdot \nabla \boldsymbol{u} = -\nabla p + \nu \nabla^2 \boldsymbol{u} + \boldsymbol{b}. \tag{3.6}$$

Note que todos os termos são divididos por ρ na Eq. 3.6, originando a viscosidade cinemática do fluido ν e a pressão cinemática p.

3.2 Condições de contorno

As condições de contorno essenciais de Dirichlet e naturais de Newmann da Eq. 3.1 são representadas como

$$\boldsymbol{u} = \overline{\boldsymbol{u}} \quad \text{em} \quad \Gamma_u, \tag{3.7}$$

$$(2\nu \nabla^s \boldsymbol{u} - p\boldsymbol{I})\,\boldsymbol{n} = \bar{\boldsymbol{t}} \quad \text{em} \quad \Gamma_t, \tag{3.8}$$

$$\boldsymbol{u} = \boldsymbol{u}_0 \quad \text{em} \quad \Omega \ \text{ e } \ t = 0. \tag{3.9}$$

3.3 Forma fraca

Definindo-se funções teste arbitrárias $\boldsymbol{w} \in \mathcal{H}_0^1(\Omega)$ e $q \in \mathcal{L}_2(\Omega)$ para a velocidade e pressão respectivamente, pode-se escrever o sistema de equações diferenciais parciais descritas pela Eq. (3.6) na forma integral equivalente como

$$(\boldsymbol{w}, \dot{\boldsymbol{u}})_\Omega + (\boldsymbol{w}, \boldsymbol{u} \cdot \nabla \boldsymbol{u})_\Omega = -(\boldsymbol{w}, \nabla p)_\Omega + (\boldsymbol{w}, \nabla \cdot (2\nu \nabla^s \boldsymbol{u}))_\Omega + (q, \nabla \cdot \mathbf{u})_\Omega + (\boldsymbol{w}, \boldsymbol{b})_\Omega \quad (3.10)$$

$\forall (\boldsymbol{w}, q)$. Integrando por partes os termos viscoso e pressão, a forma fraca da equação de Navier-Stokes para escoamentos incompressíveis pode ser definido como

$$(\boldsymbol{w}, \dot{\boldsymbol{u}})_\Omega + c(\boldsymbol{u}; \boldsymbol{w}, \boldsymbol{u})_\Omega = (p, \nabla \cdot \boldsymbol{w})_\Omega - a(\boldsymbol{w}, \boldsymbol{u})_\Omega - (q, \nabla \cdot \boldsymbol{u})_\Omega + (\boldsymbol{w}, \bar{\boldsymbol{t}})_{\Gamma_t} + (\boldsymbol{w}, \boldsymbol{b})_\Omega \quad (3.11)$$

$\forall (\boldsymbol{w}, q)$. Note que o termo do contorno em Γ desaparece em Γ_u devido a função teste. Os termos convectivos e viscosos foram escritos em uma notação compacta definidos pelas formas bilinear e trilinear:

$$a(\boldsymbol{w}, \boldsymbol{u})_\Omega = \int_\Omega 2\nu \nabla^s \boldsymbol{w} : \nabla^s \boldsymbol{u} \, \mathrm{d}\Omega \quad (3.12)$$

$$c(\boldsymbol{u}; \boldsymbol{w}, \boldsymbol{u})_\Omega = \int_\Omega \boldsymbol{w} \cdot (\boldsymbol{u} \cdot \nabla \boldsymbol{u}) \, \mathrm{d}\Omega \quad (3.13)$$

3.4 Integração no tempo

O método de integração no tempo adotado para solucionar as equações de Navier-Stokes não estacionárias deste trabalho foi o método de Newmark, onde a derivada da velocidade em relação ao instante de tempo $n+1$ pode ser escrito como

$$\dot{\boldsymbol{u}}^{n+1} = \frac{1}{\gamma} \frac{(\boldsymbol{u}^{n+1} - \boldsymbol{u}^n)}{\Delta t} - \frac{(1-\gamma)}{\gamma} \dot{\boldsymbol{u}}^n, \quad (3.14)$$

onde $\gamma = 1/2$ foi adotado como parâmetro de Newmark de forma a obter precisão de segunda ordem no tempo. Reescrevendo a forma fraca da equação de Navier-Stokes (Eq. 3.11),

$$\Big\{ (\boldsymbol{w}, \boldsymbol{u})_\Omega + \gamma \Delta t \Big[c(\boldsymbol{u}; \boldsymbol{w}, \boldsymbol{u})_\Omega + a(\boldsymbol{w}, \boldsymbol{u})_\Omega - (\nabla \cdot \boldsymbol{w}, p)_\Omega + (q, \nabla \cdot \boldsymbol{u})_\Omega - (\boldsymbol{w}, \bar{\boldsymbol{t}})_{\Gamma_t} - \\ - (\boldsymbol{w}, \boldsymbol{b})_\Omega \Big] \Big\}^{n+1} = (\boldsymbol{w}, \boldsymbol{u}^n)_\Omega + (1-\gamma) \Delta t (\boldsymbol{w}, \dot{\boldsymbol{u}}^n)_\Omega, \forall (\boldsymbol{w}, q). \quad (3.15)$$

3.5 Discretização em Elementos Finitos

Este trabalho adotou uma formulação mista para os elementos do fluido onde duas incógnitas (velocidade e pressão) são descritas como variáveis primitivas e os multiplicadores de Lagrange, incógnitas adicionais que precisam ser discretizadas ao longo da interface

entre o fluido e a estrutura. As funções de aproximação para as velocidades e pressão, definidas localmente no interior do domínio Ω_e, podem ser definidas como

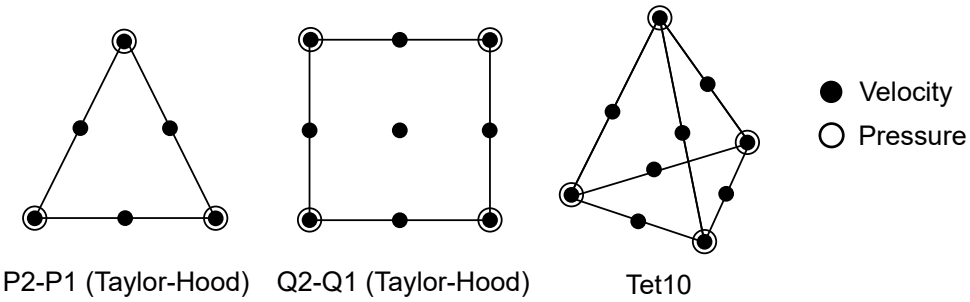
$$\boldsymbol{u}^h(\boldsymbol{x})\Big|_{\boldsymbol{x}\in\Omega_e} = \mathbf{N}_u\mathbf{u}_e \quad \text{and} \quad p^h(\boldsymbol{x})\Big|_{\boldsymbol{x}\in\Omega_e} = \mathbf{N}_p\mathbf{p}_e \tag{3.16}$$

respectivamente. $\mathbf{N}_u$ e $\mathbf{N}_p$ são as funções de forma da velocidade e pressão respectivamente, e $\mathbf{u}_e$ e $\mathbf{p}_e$ seus valores nodais. As funções teste são definidas de forma similar da seguinte maneira:

$$\boldsymbol{w}^h(\boldsymbol{x})\Big|_{\boldsymbol{x}\in\Omega_e} = \mathbf{N}_u\mathbf{w}_e \quad \text{and} \quad q^h(\boldsymbol{x})\Big|_{\boldsymbol{x}\in\Omega_e} = \mathbf{N}_p\mathbf{q}_e \tag{3.17}$$

Neste trabalho, de forma a não violar a condição de compatibilidade LBB[1] para problemas mistos, foram utilizados os elementos de Taylor-Hood P2P1 e Q2Q1 (interpolação quadrática na velocidade e linear na pressão) para problemas bidimensionais e o elemento tetraédrico de 10 nós (sendo 10 nós para a velocidade e 4 para a pressão) para problemas tridimensionais, como ilustrado na Fig. 3.1.

Figura 3.1 – Elementos finitos utilizados na discretização do domínio do fluido.



Fonte: O autor

Após algumas manipulações algébricas das equações apresentadas nas seções anteriores e utilizando a discretização em Elementos Finitos apresentada anteriormente, definem-se

$$\begin{cases} \dfrac{\mathbf{M}}{\gamma\Delta t}\mathbf{u}^{n+1} + \left[\mathbf{C}\left(\mathbf{u}^{n+1}\right) + \mathbf{K}\right]\mathbf{u}^{n+1} + \mathbf{G}\mathbf{p}^{n+1} & = \mathbf{f}^{n+1} + \dfrac{\mathbf{M}}{\gamma\Delta t}\mathbf{u}^n + \dfrac{1-\gamma}{\gamma}\mathbf{M}\dot{\mathbf{u}}^n \\ \mathbf{G}^{\mathsf{T}}\mathbf{u}^{n+1} & = \mathbf{0} \end{cases} \tag{3.18}$$

[1]A condição de compatibilidade LBB - Ladyzhenskaya-Babuška-Brezzi, ou condição inf-sup, necessita que um par de velocidade e pressão sejam escolhidos de forma a satisfazer e garantir a unicidade e existência dos espaços de interpolação para a velocidade e pressão. Uma extensa discussão sobre a condição LBB pode ser encontrada em Brezzi e Fortin (1991).

onde as matrizes locais são definidas como

$$\mathbf{K}_e = \int_{\Omega_e} \mathbf{B}_u^{\mathsf{T}} \nu \mathbf{B}_u \mathrm{d}\Omega_e \qquad \text{Matriz viscosa}$$

$$\mathbf{G}_e = -\int_{\Omega_e} (\nabla \cdot \mathbf{N}_u)^{\mathsf{T}} \mathbf{N}_p \mathrm{d}\Omega_e \qquad \text{Operador gradiente}$$

$$\mathbf{G}_e^{\mathsf{T}} \qquad \text{Operador divergente}$$

$$\mathbf{M}_e = \int_{\Omega_e} \mathbf{N}_u^{\mathsf{T}} \mathbf{N}_u \mathrm{d}\Omega_e \qquad \text{Matriz de massa}$$

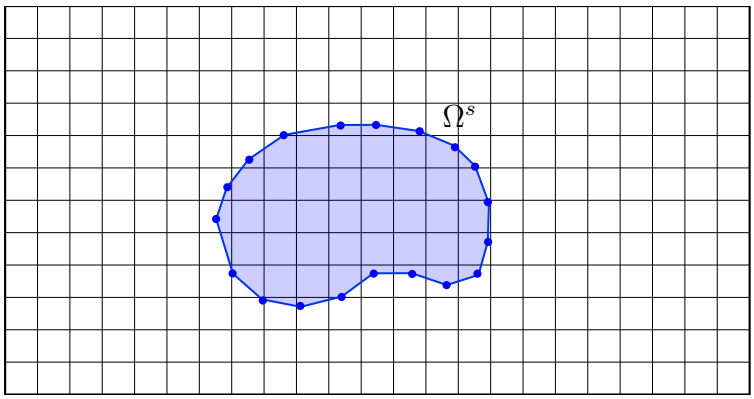
$$\mathbf{C}_e = \int_{\Omega_e} \mathbf{N}_u^{\mathsf{T}} \left[(\mathbf{N}_{u,i} \mathbf{u}_e) \otimes \mathbf{e}_i \right] \mathbf{N}_u \mathrm{d}\Omega_e \qquad \text{Matriz convectiva}$$

$$\mathbf{f}_e = \int_{\Gamma_{te}} \mathbf{N}_u^{\mathsf{T}} \bar{\mathbf{t}} \mathrm{d}\Gamma_{te} + \int_{\Omega_e} \mathbf{N}_u^{\mathsf{T}} \mathbf{b} \mathrm{d}\Omega_e \qquad \text{Condições de contorno e forças de campo}$$

3.6 Interface

A estratégia adotada neste trabalho foi utilizar fronteiras imersas para as simulações de interação fluido-estrutura para calcular as variáveis do escoamento do fluido em uma malha de elementos finitos Euleriana fixa. A malha do fluido é definida sobre o domínio do fluido e estende-se totalmente sobre o interior do domínio da estrutura, ou sobre uma parte deste domínio, como mostrado na Fig. 3.2.

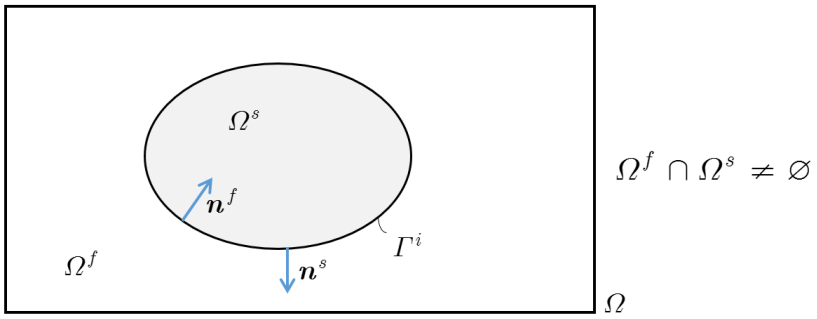
Figura 3.2 – Malhas de fluidos típica para o método das fronteiras imersas.



Fonte: O autor

Por esta razão, a superfície molhada da estrutura, geralmente, não é conforme com os nós da malha do fluido, desta forma as velocidades da interface precisam ser definidas de forma fraca. Define-se o domínio que contém o fluido como Ω^f e a estrutura como Ω^s (Fig. 3.3). Γ^i é a interface entre os domínios do fluido e da estrutura, $\boldsymbol{n}^f$ e $\boldsymbol{n}^s$ seus respectivos vetores normais e unitários à superfície externa da estrutura.

Figura 3.3 – Domínios do fluido e da estrutura.



Fonte: (GOMES, 2013)

De forma a garantir a compatibilidade da velocidade do fluido na interface, precisa-se, para uma interface sem escorregamento (*no-slip condition*)

$$\boldsymbol{u}^f = \dot{\boldsymbol{d}}^s = \overline{\boldsymbol{u}}^i \qquad \forall \boldsymbol{x} \in \Gamma^i, \tag{3.19}$$

$$\boldsymbol{T}^f \boldsymbol{n}^f = -\boldsymbol{T}^s \boldsymbol{n}^s \qquad \forall \boldsymbol{x} \in \Gamma^i \tag{3.20}$$

onde a Eq. 3.19 é a condição cinemática da interface, $\dot{\boldsymbol{d}}$ é a velocidade da estrutura e Eq. 3.20 representa a condição dinâmica. Os sobrescritos nestas equações diferenciam as variáveis dos diferentes domínios (fluido e estrutura).

A imposição da Eq. 3.19 no problema do fluido pode ser realizada com multiplicadores de Lagrange. Portanto, é possível definir um funcional por

$$\Pi = \left(\boldsymbol{\lambda}, \boldsymbol{u}^f - \dot{\boldsymbol{d}}^s\right)_{\Gamma^i} \tag{3.21}$$

onde λ representa o multiplicador de Lagrange da condição Eq. 3.19 e constitui-se de uma variável adicional do problema. O multiplicador de Lagrange pode ser identificado como uma força ao longo de Γ^i. A forma variacional desta função, considerando o movimento da estrutura conhecido *a priori* no momento da solução do problema do fluido, é

$$\delta\Pi = \left(\delta\boldsymbol{\lambda}, \boldsymbol{u}^f - \dot{\boldsymbol{d}}^s\right)_{\Gamma^i} + \left(\boldsymbol{\lambda}, \delta\boldsymbol{u}^f\right)_{\Gamma^i}. \tag{3.22}$$

Estes termos podem ser adicionados na forma fraca apresentada em 3.15. Desta forma

$$\begin{aligned} \Big\{ (\boldsymbol{w}, \boldsymbol{u})_{\Omega^f} + \gamma\Delta t \Big[c\,(\boldsymbol{u}; \boldsymbol{w}, \boldsymbol{u})_{\Omega^f} + a\,(\boldsymbol{w}, \boldsymbol{u})_{\Omega^f} - (\nabla \cdot \boldsymbol{w}, p)_{\Omega^f} + (q, \nabla \cdot \boldsymbol{u})_{\Omega^f} - \\ - (\boldsymbol{w}, \overline{\boldsymbol{t}})_{\Gamma_t} - \left(\delta\boldsymbol{\lambda}, \boldsymbol{u} - \overline{\boldsymbol{u}}^i\right)_{\Gamma^i} - (\boldsymbol{w}, \boldsymbol{\lambda})_{\Gamma^i} - (\boldsymbol{w}, \boldsymbol{b})_{\Omega^f} \Big] \Big\}^{n+1} = \\ = (\boldsymbol{w}, \boldsymbol{u}^n)_{\Omega^f} + (1-\gamma)\,\Delta t\,(\boldsymbol{w}, \dot{\boldsymbol{u}}^n)_{\Omega^f}\,,\ \forall\,(\delta\boldsymbol{\lambda}, \boldsymbol{w}, q) \end{aligned} \tag{3.23}$$

.

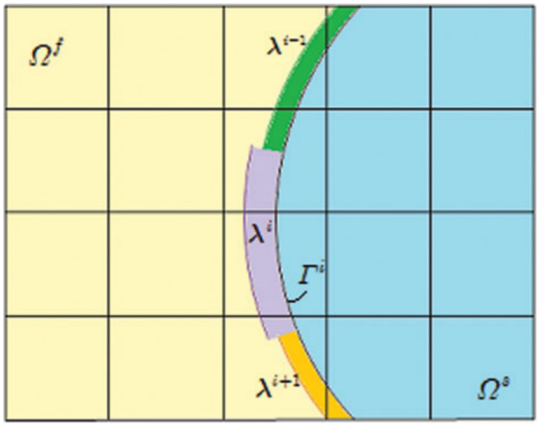
3.7 Discretização da interface

A discretização da interface pode ser realizada de diferentes formas. A técnica mais intuitiva e adotada é definir os nós da intersecção entre a interface e os elementos de

fluido. Estes nós são então utilizados para definir os valores nodais dos multiplicadores de Lagrange, e uma função de aproximação é utilizada para interpolar os valores ao longo da interface. Nas simulações realizadas em Gomes (2013), a adoção de elementos finitos de baixa ordem mostraram alta sensibilidade a instabilidades numéricas nos resultados dos multiplicadores de Lagrange.

De forma a resolver este problema, foi proposto em Gomes (2013) uma técnica alternativa de discretizar a interface de forma independente da malha do fluido. Adicionalmente, funções de forma descontínuas e constantes foram adotadas para simplificar a implementação. O esquema geral da discretização da interface é apresentada na Fig. 3.4.

Figura 3.4 – Discretização da interface entre o fluido e a estrutura.



Fonte: (GOMES, 2013)

As funções teste e de aproximação dos multiplicadores de Lagrange podem ser definidos como:

$$\boldsymbol{\lambda}^h(\boldsymbol{x})\Big|_{\boldsymbol{x}\in\Gamma_e^i} = \mathbf{N}_\lambda \boldsymbol{\lambda}_e \quad \text{and} \quad \delta\boldsymbol{\lambda}^h(\boldsymbol{x})\Big|_{\boldsymbol{x}\in\Gamma_e^i} = \mathbf{N}_\lambda \delta\boldsymbol{\lambda}_e \tag{3.24}$$

$\mathbf{N}_\lambda$ é a função de forma dos multiplicadores de Lagrange.

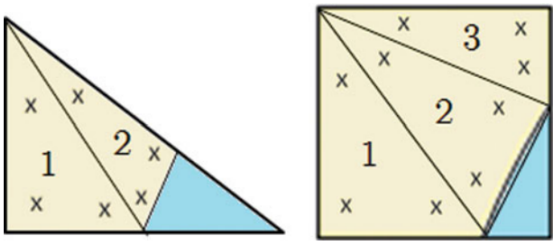
3.8 Células de integração

De forma a calcular as integrais dos elementos de fluido "cortados" pela estrutura, em Gomes (2013) o elemento que pertence tanto ao domínio da estrutura como do fluido é subdividido em células triangulares de integração, utilizando uma técnica chamada *Tessellation*.

Dentro do elemento de fluido a interface é aproximada por uma linha reta. Esta aproximação é muito comum nos Métodos dos Elementos Finitos onde o erro converge para zero com a diminuição do tamanho do elemento.

[1] *Tessellation* é uma técnica de recobrimento de superfícies que utiliza um padrão de formatos geométricos (polígonos regulares) de forma a não possuir espaços nem sobreposição entre eles.

Figura 3.5 – Integração nos elementos "cortados". Os pontos de Gauss são representados pelo símbolo "x".



Fonte: (GOMES, 2013)

Neste trabalho adotou-se, além da técnica de *Tessellation*, uma técnica de integração onde não é necessário subdividir o elemento "cortado" em células de integração. A integração é realizada no domínio do polígono resultante do "corte". Esta técnica facilita a integração em problemas bidimensionais, mas principalmente em problemas tridimensionais (SUDHAKAR et al., 2014), e é apresentada na próxima seção.

3.9 Integração no polígono

Nesta seção são apresentados os principais tópicos da teoria da integração no polígono aplicada no presente trabalho. A teoria completa encontra-se em Sudhakar et al. (2014).

Considere a integração da função polinomial $\mathcal{F}$ no domínio $\mathcal{R} \subset \mathbb{R}^n$ da seguinte forma

$$I_{\mathcal{R}} = \int_{\mathcal{R}} \mathcal{F} \mathrm{d}\mathcal{R} \tag{3.25}$$

Seja $\mathcal{R}$ a região em $\mathbb{R}^n$ delimitado pela superfície fechada $\mathcal{S}$ (no caso do presente trabalho Ω_s como apresentado na Fig. 3.3). Seja $\hat{\mathbf{n}}$ o vetor normal a superfície externa de $\mathcal{R}$ em $\mathcal{S}$, então o teorema do divergente define que para qualquer vetor $\mathbf{F}$ definido em $\mathcal{R}$, como

$$\int_{\mathcal{R}} \nabla \cdot \mathbf{F} \mathrm{d}\mathcal{R} = \int_{\mathcal{S}} \mathbf{F} \cdot \hat{\mathbf{n}} \mathrm{d}\mathcal{S} \tag{3.26}$$

Pode-se definir o vetor $\mathbf{F}$ como sendo[2]

$$\mathbf{F} = \mathcal{G}(\mathbf{x})\,\hat{\imath} + 0\hat{\jmath} + 0\hat{k} \tag{3.27}$$

onde $\hat{\imath}$, $\hat{\jmath}$ e $\hat{k}$ são os vetores normais nas direções x, y e z, respectivamente.

Portanto

$$\mathcal{G}(\mathbf{x}) = \int_{\kappa}^{\mathbf{x}} \mathcal{F} \mathrm{d}x \tag{3.28}$$

[2]O objetivo é encontrar $\mathbf{F}$ que satisfaça a equação $\nabla \cdot \mathbf{F} = \mathcal{F}$, sendo $\mathcal{F}$ uma função polinomial escalar. Existe mais de uma solução para esta equação e pode ser escolhida qualquer um dos vetores $\mathbf{F}$ sem perda de generalidade. No exemplo e em Sudhakar et al. (2014) escolheu-se trabalhar com valores não nulos da componente x, mas poderia ter sido escolhido tanto y quanto z e os resultados seriam os mesmos.

onde κ é um ponto de referência sobre a linha de integração.

Substituindo as Eq. 3.26 e 3.27 em 3.28, tem-se

$$\int_{\mathcal{R}} \mathcal{F} \mathrm{d}\mathcal{R} = \int_{\mathcal{S}} \mathcal{G}\left(\mathbf{x}\right) n_x \mathrm{d}\mathcal{S} \tag{3.29}$$

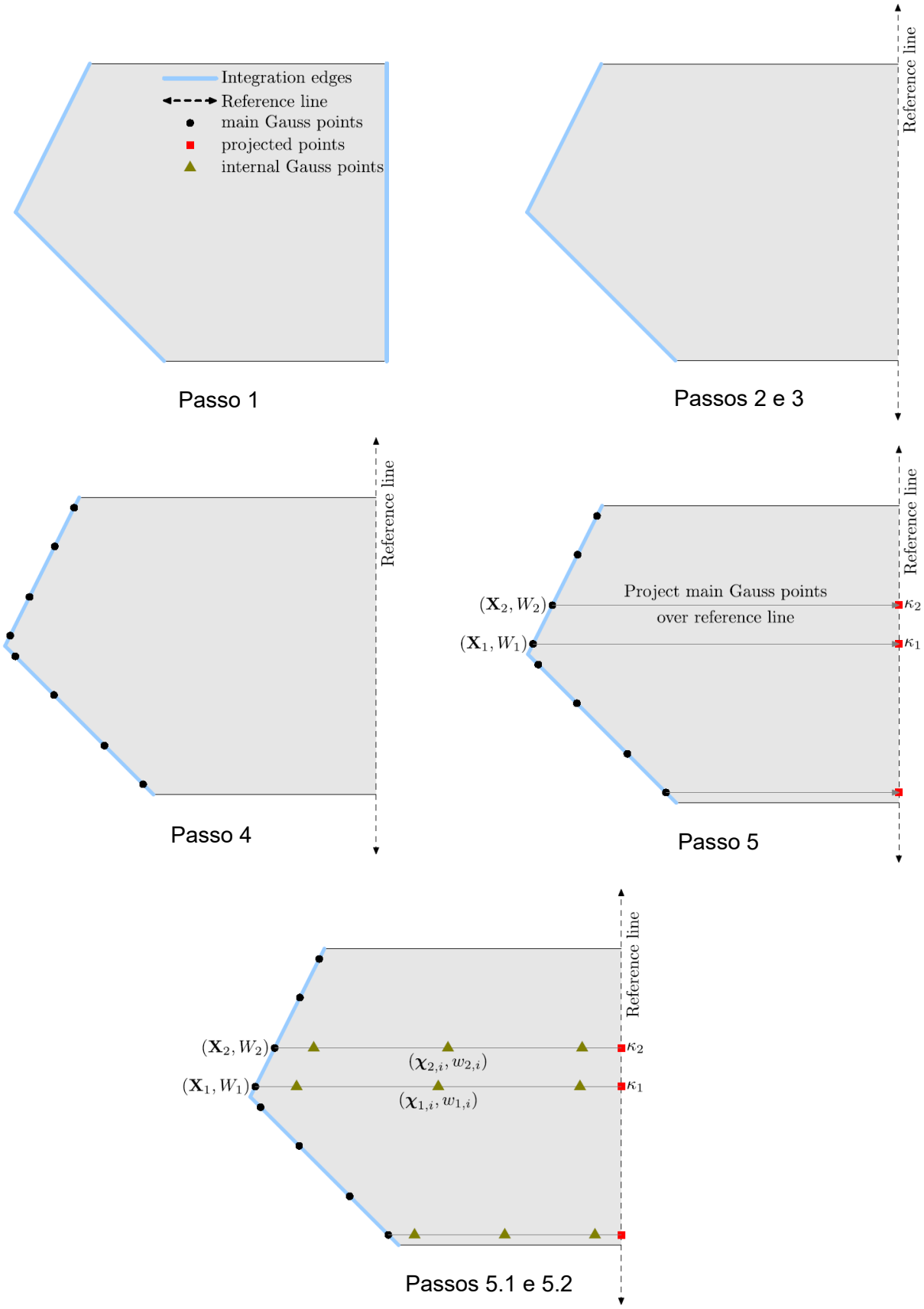
onde n_x é a componente do vetor $\hat{\mathbf{n}}$ na direção x.

A integração no polígono "cortado" é, portanto, realizado da seguinte forma (a Fig. 3.6 mostra os passos necessários para integrar um polígono)

Tabela 3.1 – Algoritmo de integração no polígono

Algoritmo Integração em {polígono} [poliedro]	
1:	Identificar e armazenar os {lados} [faces] do polígono "cortado";
2:	Definir a {linha} [plano] de referência;
3:	Excluir os {lados} [faces] que estejam sobre a {linha} [plano] de referência;
4:	Distribuir os pontos principais de Gauss sobre cada {lado} [face] do polígono que possua a componente da normal diferente de zero;
5:	Para cada ponto principal de Gauss sobre {lado} [face] projetar o vetor de coordenadas $\mathbf{X}_i$ na {linha} [plano] de referência
5.1:	Distribuir os pontos internos de Gauss entre $\mathbf{X}_i$ e κ_i
5.2:	Para cada ponto de Gauss interno $\boldsymbol{\chi}_i$ calcular $\mathcal{G}\left(\mathbf{X}_i\right) = \mathcal{G}\left(\mathbf{X}_i\right) + \lvert\mathbf{J}\rvert\mathcal{F}\left(\boldsymbol{\chi}_{i,j}\right) \mathrm{w}_{i,j}$, sendo $\lvert\mathbf{J}\rvert$ o Jacobiano da linha projetada;
6:	Calcular $I_{\mathcal{R}} = I_{\mathcal{R}} + \lvert\mathbf{J}\rvert\mathcal{G}\left(\mathbf{X}_i\right) \mathrm{n}_x\left(\mathbf{X}_i\right) \mathrm{W}_i$, sendo $\lvert\mathbf{J}\rvert$ o Jacobiano do lado do polígono com componente normal diferente de zero.

Figura 3.6 – Passos necessários para integrar um polígono.

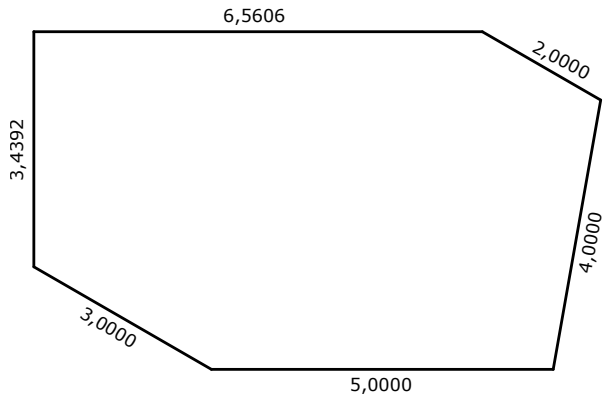


Fonte: (SUDHAKAR et al., 2014)

A seguir é apresentado um exemplo para melhor entendimento do método desenvolvido por Sudhakar et al. (2014).

Considere o polígono apresentado na Fig. 3.7 como resultado do "corte" entre a superfície molhada e a malha de elementos finitos. Utilizando-se da teoria apresentada anteriormente será calculada a área deste polígono.

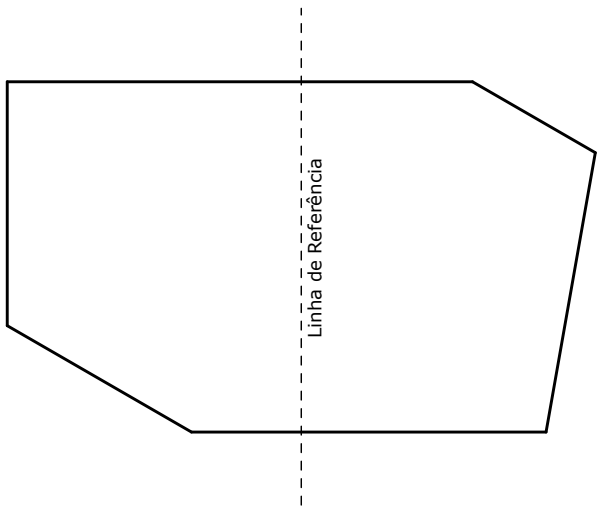
Figura 3.7 – Polígono resultado do "corte" entre a superfície molhada e a malha de elementos finitos. Unidades: adimensional



Fonte: O autor

Primeiramente define-se a linha de referência. Esta linha de referência pode ser qualquer um dos lados do polígono, mas também pode ser uma linha que passe verticalmente no centro geométrico da seção. Esta linha é apresentada na Fig. 3.8.

Figura 3.8 – Linha de referência.

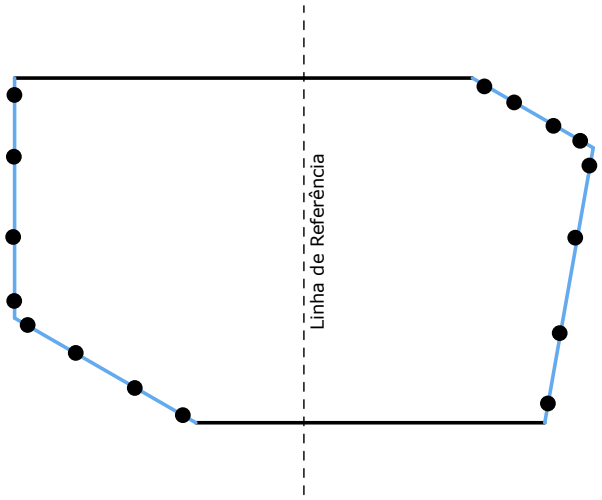


Fonte: O autor

Exclui-se os lados do polígono que estejam sob a linha de referência definida no passo anterior. Como não há nenhum lado sob esta linha mantém-se o polígono na sua forma original. Distribuem-se os pontos de Gauss sobre cada lado do polígono onde sua

componente normal seja diferente de zero (Fig. 3.9). Na teoria apresentada em Sudhakar et al. (2014) utilizou-se 4 pontos de Gauss[3].

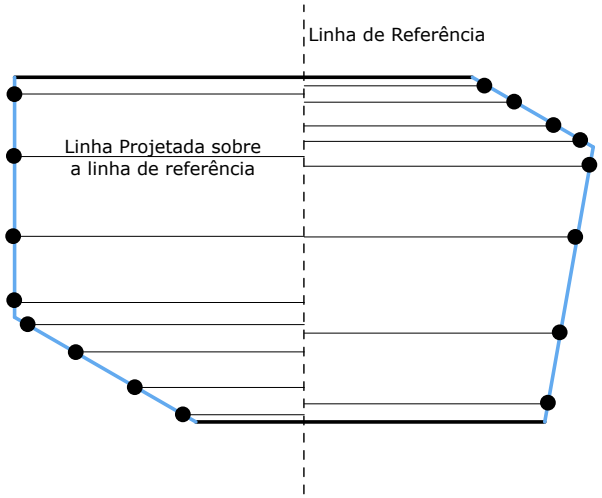
Figura 3.9 – Distribuição dos pontos de Gauss principais sobre os lados do polígono com componente normal diferente de zero.



Fonte: O autor

Para cada ponto de Gauss principal projeta-se o vetor de coordenadas $\mathbf{X}_i$ na linha de referência como apresentado na Fig. 3.10.

Figura 3.10 – Linhas projetadas sobre a linha de referência partindo de cada ponto de Gauss principal.



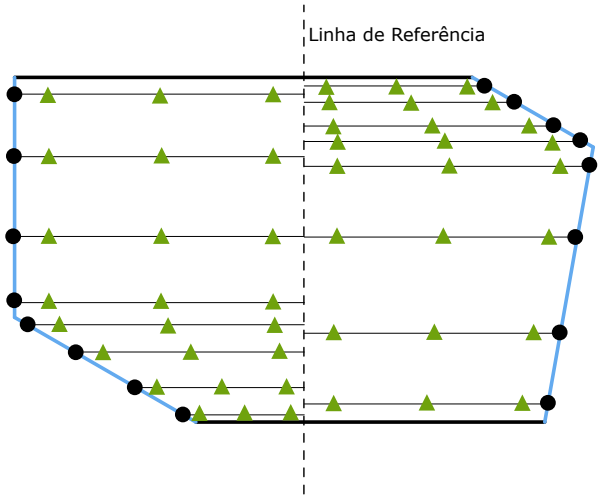
Fonte: O autor

Esta nova linha projetada será a base para posicionamento dos pontos de Gauss internos. Distribuem-se nesta linha os pontos de Gauss internos, neste caso 3. A Fig. 3.11

[3]Os pontos de Gauss internos são utilizados para integrar $\mathcal{F}$ e os pontos de Gauss principais são utilizados para integrar $\mathcal{G}(\mathbf{x})$. Como $\mathcal{G}(\mathbf{x})$ é calculado integrando $\mathcal{F}$, a ordem do polinômio de $\mathcal{G}(\mathbf{x})$ é acrescido de um. Portanto a ordem dos pontos de Gauss principais devem ser suficientes para integrar $\mathcal{G}(\mathbf{x})$. Para integrar um polinômio de ordem 5, foram utilizados 3 pontos de Gauss para os pontos internos e 4 para os pontos principais neste exemplo e em Sudhakar et al. (2014).

mostra os pontos de Gauss principais e internos posicionados sobre os lados do polígono com componente normal diferente de zero e sobre as linhas projetadas, respectivamente.

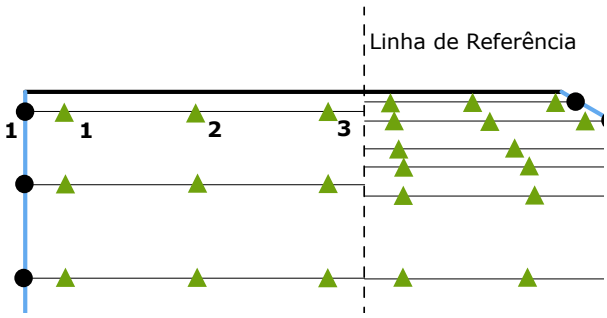
Figura 3.11 – Distribuição dos pontos de Gauss internos sobre a linha projetada.



Fonte: O autor

Para cada ponto de Gauss interno, calcula-se $\mathcal{G}\left(\mathbf{X}_i\right) = \mathcal{G}\left(\mathbf{X}_i\right) + |\mathbf{J}|\mathcal{F}\left(\boldsymbol{\chi}_{i,j}\right) \mathrm{w}_{i,j}$. A Fig. 3.12 mostra a numeração do ponto de Gauss principal 1 e os pontos de Gauss internos sobre a linha projetada. A Tab. 3.2 apresenta os resultados para este ponto de Gauss principal.

Figura 3.12 – Cálculo de $\mathcal{G}\left(\mathbf{X}_i\right)$ para o ponto de Gauss principal 1.



Fonte: O autor

Tabela 3.2 – Cálculo do $\mathcal{G}\left(\mathbf{X}_i\right)$ para o ponto de Gauss principal 1.

Ponto de Gauss Principal (i)	Ponto de Gauss Interno (j)	$\lvert\mathbf{J}\rvert$	$\mathrm{w}_{i,j}$	$\lvert\mathbf{J}\rvert\mathcal{F}\left(\boldsymbol{\chi}_{i,j}\right) \mathrm{w}_{i,j}$
1	1	2.0732	0.5556	1.1518
	2	2.0732	0.8889	1.8428
	3	2.0732	0.5556	1.1518
			$\mathcal{G}\left(\mathbf{X}_1\right)$	**4.1463**

Calcula-se finalmente $I_{\mathcal{R}} = I_{\mathcal{R}} + |\mathbf{J}|\mathcal{G}\left(\mathbf{X}_i\right)\mathrm{n}_x\left(\mathbf{X}_i\right)\mathrm{W}_i$. A Tab. 3.3 apresenta os resultados para cada ponto e a área do polígono.

Tabela 3.3 – Cálculo de $I_{\mathcal{R}}$ para o polígono.

i	j	$\lvert\mathbf{J}\rvert$	$\mathrm{w}_{i,j}$	$\lvert\mathbf{J}\rvert\mathcal{F}\left(\boldsymbol{\chi}_{i,j}\right)\mathrm{w}_{i,j}$	$\lvert\mathbf{J}\rvert$	$\mathrm{n}_x\left(\mathbf{X}_i\right)$	W_i	$\lvert\mathbf{J}\rvert\mathcal{G}\left(\mathbf{X}_i\right)\mathrm{n}_x\left(\mathbf{X}_i\right)\mathrm{W}_i$
1	1	2.0732	0.5556	1.1518				
	2	2.0732	0.8889	1.8428				
	3	2.0732	0.5556	1.1518				
			$\mathcal{G}\left(\mathbf{X}_1\right)$	**4.1463**	1.7196	0.3479	1.0000	**2.4802**
2	1	2.0732	0.5556	1.1518				
	2	2.0732	0.8889	1.8428				
	3	2.0732	0.5556	1.1518				
			$\mathcal{G}\left(\mathbf{X}_2\right)$	**4.1463**	1.7196	0.6521	1.0000	**4.6498**
3	1	2.0732	0.5556	1.1518				
	2	2.0732	0.8889	1.8428				
	3	2.0732	0.5556	1.1518				
			$\mathcal{G}\left(\mathbf{X}_3\right)$	**4.1463**	1.7196	0.6521	1.0000	**4.6498**
4	1	2.0732	0.5556	1.1518				
	2	2.0732	0.8889	1.8428				
	3	2.0732	0.5556	1.1518				
			$\mathcal{G}\left(\mathbf{X}_4\right)$	**4.1463**	1.7196	0.3479	1.0000	**2.4802**
5	1	1.9830	0.5556	1.1017				
	2	1.9830	0.8889	1.7627				
	3	1.9830	0.5556	1.1017				
			$\mathcal{G}\left(\mathbf{X}_5\right)$	**3.9660**	1.5000	0.3479	0.5000	**1.0347**
6	1	1.6445	0.5556	0.9136				
	2	1.6445	0.8889	1.4617				
	3	1.6445	0.5556	0.9136				
			$\mathcal{G}\left(\mathbf{X}_6\right)$	**3.2889**	1.5000	0.6521	0.5000	**1.6086**
7	1	1.2028	0.5556	0.6682				
	2	1.2028	0.8889	1.0692				
	3	1.2028	0.5556	0.6682				
			$\mathcal{G}\left(\mathbf{X}_7\right)$	**2.4056**	1.5000	0.6521	0.5000	**1.1766**
8	1	0.8643	0.5556	0.4802				
	2	0.8643	0.8889	0.7683				
	3	0.8643	0.5556	0.4802				
			$\mathcal{G}\left(\mathbf{X}_8\right)$	**1.7286**	1.5000	0.3479	0.5000	**0.4510**
9	1	2.0491	0.5556	1.1384				
	2	2.0491	0.8889	1.8214				
	3	2.0491	0.5556	1.1384				
			$\mathcal{G}\left(\mathbf{X}_9\right)$	**4.0981**	2.0000	0.3479	0.9848	**2.8078**
10	1	1.9586	0.5556	1.0881				
	2	1.9586	0.8889	1.7409				
	3	1.9586	0.5556	1.0881				
			$\mathcal{G}\left(\mathbf{X}_{10}\right)$	**3.9171**	2.0000	0.6521	0.9848	**5.0314**

Continua

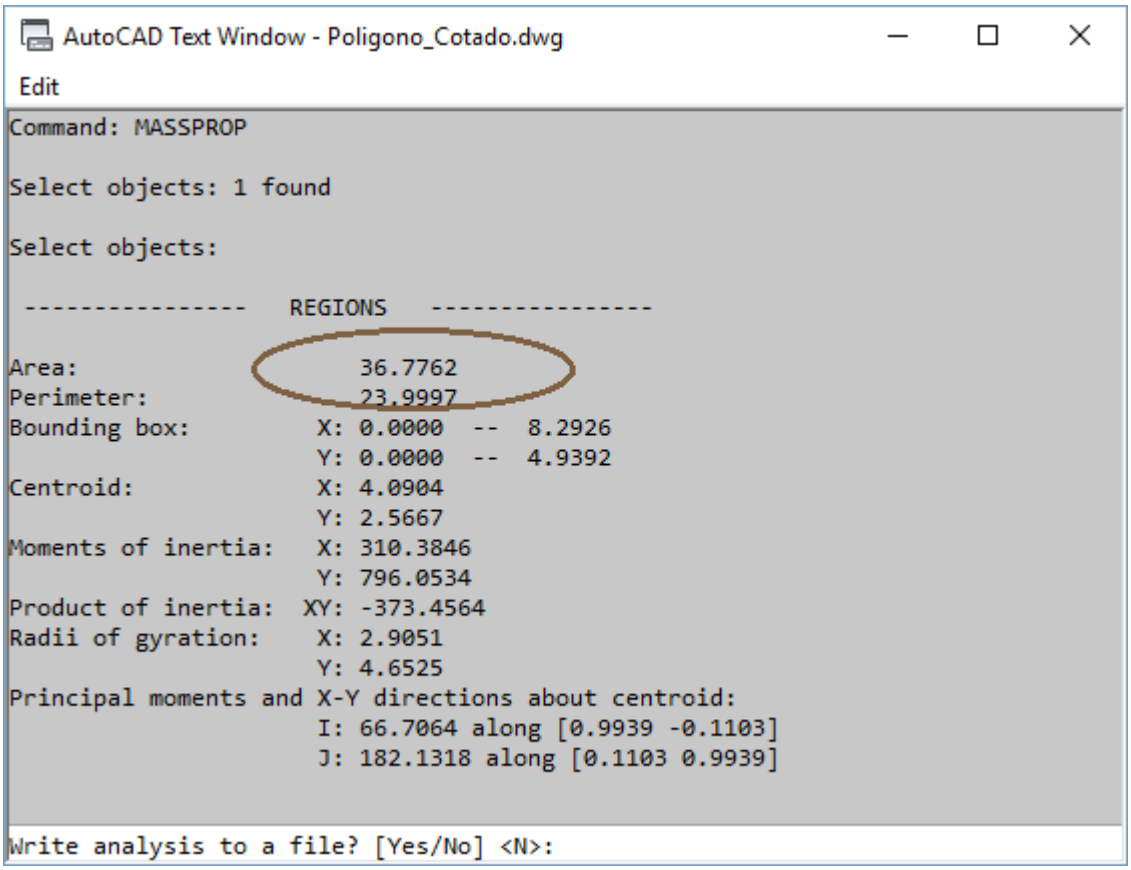
Tabela 3.3 – Cálculo de $I_{\mathcal{R}}$ para o polígono.

i	j	$\lvert\mathbf{J}\rvert$	$\mathrm{w}_{i,j}$	$\lvert\mathbf{J}\rvert\mathcal{F}\left(\boldsymbol{\chi}_{i,j}\right)\mathrm{w}_{i,j}$	$\lvert\mathbf{J}\rvert$	$\mathrm{n}_x\left(\mathbf{X}_i\right)$	W_i	$\lvert\mathbf{J}\rvert\mathcal{G}\left(\mathbf{X}_i\right)\mathrm{n}_x\left(\mathbf{X}_i\right)\mathrm{W}_i$
11	1	1.8405	0.5556	1.0225				
	2	1.8405	0.8889	1.6360				
	3	1.8405	0.5556	1.0225				
			$\mathcal{G}\left(\mathbf{X}_{11}\right)$	**3.6810**	2.0000	0.6521	0.9848	**4.7281**
12	1	1.7500	0.5556	0.9722				
	2	1.7500	0.8889	1.5556				
	3	1.7500	0.5556	0.9722				
			$\mathcal{G}\left(\mathbf{X}_{12}\right)$	**3.5000**	2.0000	0.3479	0.9848	**2.3980**
13	1	1.2673	0.5556	0.7040				
	2	1.2673	0.8889	1.1264				
	3	1.2673	0.5556	0.7040				
			$\mathcal{G}\left(\mathbf{X}_{13}\right)$	**2.5345**	1.0000	0.3479	0.5000	**0.4408**
14	1	1.4930	0.5556	0.8294				
	2	1.4930	0.8889	1.3271				
	3	1.4930	0.5556	0.8294				
			$\mathcal{G}\left(\mathbf{X}_{14}\right)$	**2.9859**	1.0000	0.6521	0.5000	**0.9736**
15	1	1.7874	0.5556	0.9930				
	2	1.7874	0.8889	1.5888				
	3	1.7874	0.5556	0.9930				
			$\mathcal{G}\left(\mathbf{X}_{15}\right)$	**3.5747**	1.0000	0.6521	0.5000	**1.1656**
16	1	2.0131	0.5556	1.1184				
	2	2.0131	0.8889	1.7894				
	3	2.0131	0.5556	1.1184				
			$\mathcal{G}\left(\mathbf{X}_{16}\right)$	**4.0261**	1.0000	0.3479	0.5000	**0.7002**
							$I_{\mathcal{R}}$	**36.7764**

Conclusão

Como base de verificação do algoritmo desenvolvido, utilizou-se a ferramenta *MASSPROP* do software comercial Autodesk AutoCAD 2016® para comparar a área com a integração calculada para o polígono. A Fig. 3.13 mostra a janela com o resultado.

Figura 3.13 – Resultado do software comercial Autodesk AutoCAD 2016®.



Fonte: O autor

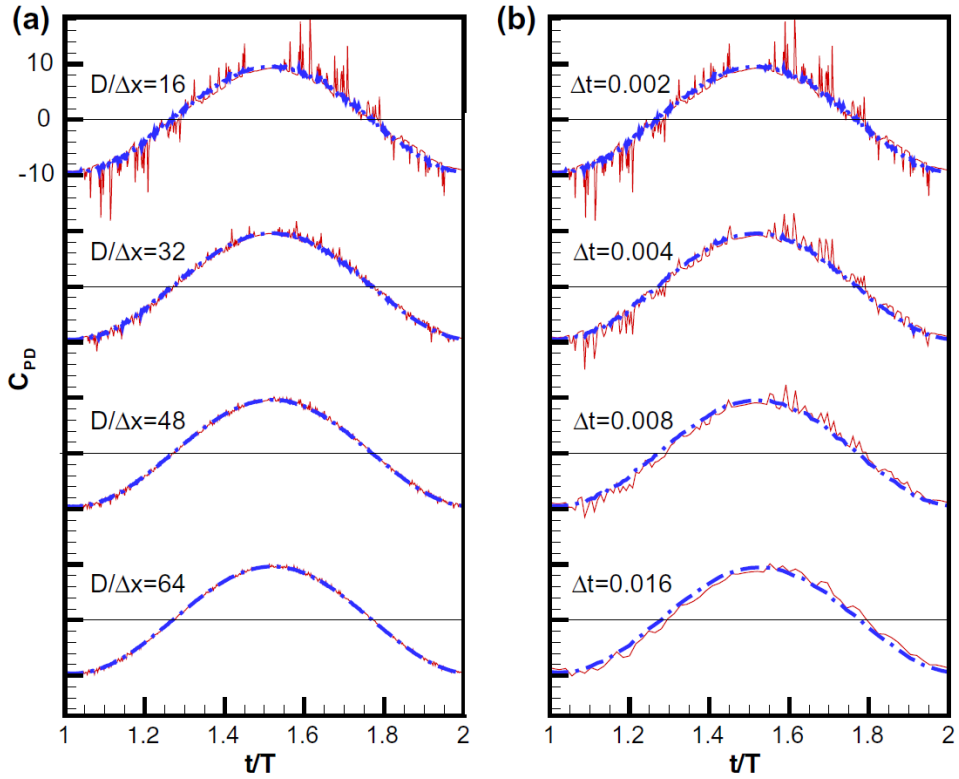
3.10 Conservação de massa nos métodos de Fronteiras Imersas

Os métodos de Fronteiras Imersas são muito versáteis para a simulação de escoamentos em torno de estruturas que se movimentam e deformam com formas geometricamente complexas. Uma propriedade indesejável dos métodos de fronteiras imersas que tem sido reportados por pesquisadores é a oscilação temporal dos campos de pressão. Estas oscilações são observadas virtualmente para todos os tipos de fronteiras imersas. A maior fonte de erro destas oscilações é devido ao fato que alguns nós da malha ou são "gerados" com o tempo (nós do domínio do fluido que estavam no interior da estrutura, mas com o movimento e deformação desta estrutura, eles "aparecem") ou "desaparecem" com o tempo (nós do domínio do fluido que estavam fora da estrutura, mas com o movimento e deformação desta estrutura, eles passam a estar no interior da estrutura) (SEO; MITTAL, 2011).

As teorias para garantir a conservação de massa baseiam-se principalmente nas leis de conservação geométricas (LESOINNE; FARHAT, 1996). Em Seo e Mittal (2011) é discutido com profundidade este tema e é apresentada uma solução para diminuir estas oscilações utilizando o método *cut-cell* aplicado ao método das diferenças finitas e volumes finitos, mas expansível ao Método dos Elementos Finitos.

A Fig. 3.14 mostra o coeficiente de arrasto para um cilindro circular oscilante no tempo, em (a) diferentes resoluções de malha para o mesmo intervalo de tempo $\Delta_t = 0.002$ e em (b) para a mesma resolução de malha e diferentes intervalos de tempo. A linha sólida: método da diferença finita original e a linha traço-ponto: resultado do trabalho apresentado no artigo.

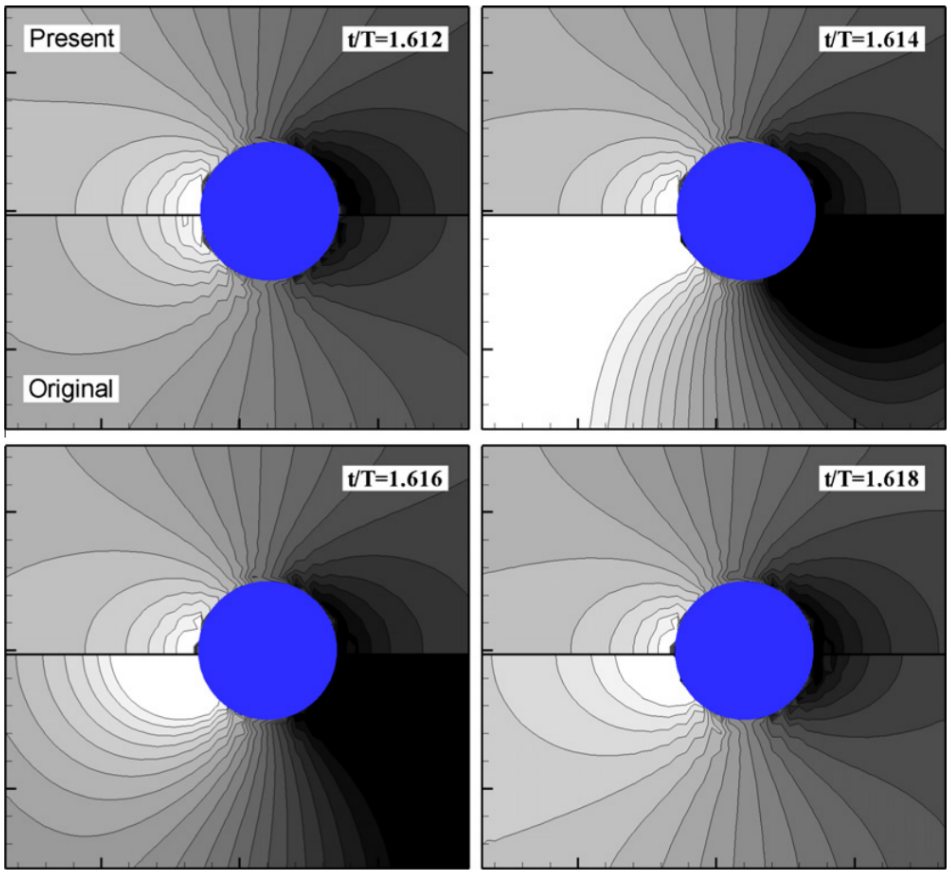
Figura 3.14 – Coeficiente de arrasto para um cilindro circular oscilante no tempo.



Fonte: (SEO; MITTAL, 2011)

A Fig. 3.15 apresenta os resultados do método para redução das oscilações da pressão com o tempo.

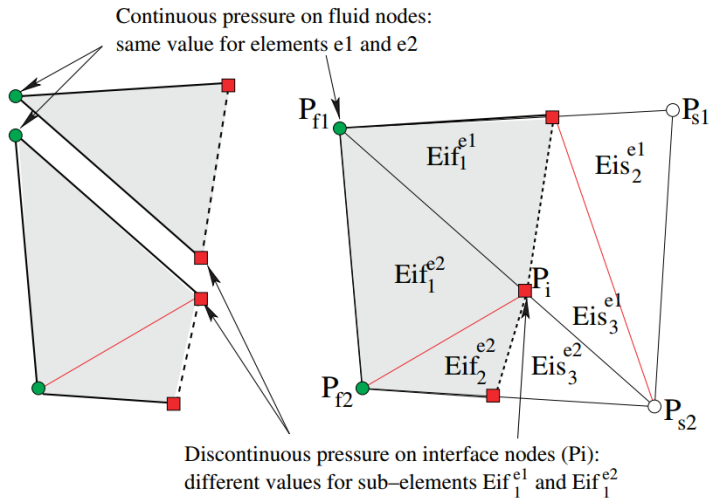
Figura 3.15 – Oscilações na pressão com o tempo e a solução apresentada no artigo.



Fonte: (SEO; MITTAL, 2011)

Outra forma discutida em Ilinca e Hetu (2010) utiliza a condensação estática de forma a eliminar os graus de liberdade da pressão definidos nos nós da intersecção entre a interface a malha de fluidos. A Fig. 3.16 mostra os nós gerados na interface quando da intersecção entre a estrutura e a malha de fluido e que são eliminados utilizando as condições de contorno de Dirichlet para a velocidade e a condensação estática para a pressão.

Figura 3.16 – Nós gerados na interface quando da intersecção entre a estrutura e a malha de fluido eliminados utilizando as condições de contorno de Dirichlet para a velocidade e a condensação estática para a pressão.



Fonte: (ILINCA; HETU, 2010)

A conservação de massa nos métodos de fronteiras imersas é amplamente discutida pela comunidade científica e diversas abordagens são utilizadas. Neste trabalho não foi implementada a conservação de massa sendo uma melhoria a ser analisada em trabalhos futuros.

4 RISERS

Uma das simulações numéricas realizadas neste trabalho é a interação fluido-estrutura entre um *riser* (com contatos entre o fundo do oceano e a própria estrutura), e a malha de elementos de fluido simulando a corrente do mar. Este capítulo demonstrará os principais tópicos da teoria para melhor entendimento dos exemplos deste trabalho. A teoria completa para o modelo de vigas 3D cinematicamente exatas pode ser encontrada em Gay Neto et al. (2013).

4.1 Modelo de vigas 3D cinematicamente exatas

Os *riser*s podem ser considerados como vigas longas, como por exemplo os cabos umbilicais, tubos flexíveis ou rígidos e mangueiras, imersos no oceano, suspensos desde o fundo até as unidades flutuantes, como pode ser visto na Fig. 4.1 (Gay Neto et al., 2013). Estes tubos rígidos ou flexíveis ou cabos umbilicais submersos possuem vínculos fixos nas duas extremidades (fundo do oceano e unidade flutuante). Eles estão sujeitos a carregamentos como por exemplo o peso próprio da estrutura, efeitos inerciais, arrastamento pela corrente do mar, como também excitações sofridas pelas ondas.

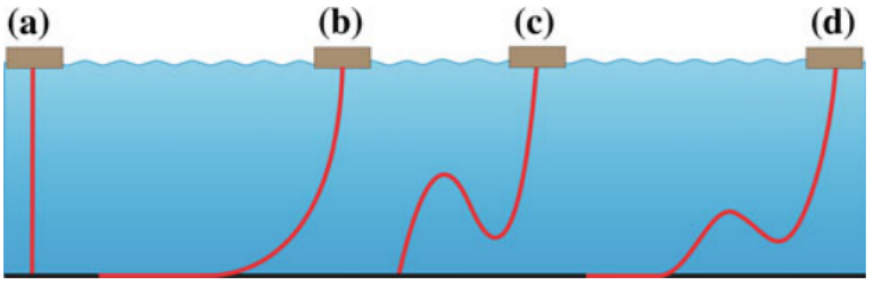


Figura 4.1 – Diferentes concepções de configurações dos *riser*s (a) vertical, (b) catenária (c) *Steep-wave* (d) *Lazy-wave* (Gay Neto et al., 2013)

4.1.1 Premissas do modelo

Um modelo de vigas de Timoshenko é assumido para representar a estrutura 3D do *riser*. Portanto é possível considerar tensões, compressões e carregamentos cisalhantes, como também momentos fletores e de torção.

4.1.2 Parametrização de rotações

Os parâmetros de Rodrigues foram escolhidos para escrever o tensor de rotação ($\boldsymbol{Q}$) na teoria encontrada em Gay Neto et al. (2013). Entretanto, a interpretação física

deste tensor não é trivial sendo interessante apresentar os parâmetros de Euler para a rotação e compará-los. O ângulo de rotação de Euler θ é utilizado para construir o vetor rotação $\boldsymbol{\theta} = \theta \boldsymbol{e}$, onde $\boldsymbol{e}$ é o vetor unitário e representa o eixo de rotação.

É possível obter a seguinte expressão para o tensor rotação utilizando os parâmetros de Rodrigues da seguinte forma

$$\boldsymbol{Q} = \boldsymbol{I} + \frac{4}{4+\alpha^2}\left(\boldsymbol{A} + \frac{1}{2}\boldsymbol{A}^2\right) \tag{4.1}$$

onde:

- $\boldsymbol{A} = \text{skew}(\boldsymbol{\alpha})$;
- $\boldsymbol{\alpha}$: Vetor de rotação de Rodrigues, definido por $\boldsymbol{\alpha} = \alpha \boldsymbol{e}$ com $\alpha = 2\,\tan(\theta/2)$.

A velocidade angular utilizando os parâmetros de Rodrigues é:

$$\boldsymbol{\omega} = \boldsymbol{\Xi}\dot{\boldsymbol{\alpha}} \tag{4.2}$$

onde

$$\boldsymbol{\Xi} = \frac{4}{4+\alpha^2}\left(\boldsymbol{I} + \frac{1}{2}\boldsymbol{A}\right) \tag{4.3}$$

A aceleração angular pode ser definida como

$$\dot{\boldsymbol{\omega}} = \dot{\boldsymbol{\Xi}}\dot{\boldsymbol{\alpha}} + \boldsymbol{\Xi}\ddot{\boldsymbol{\alpha}} \tag{4.4}$$

onde

$$\dot{\boldsymbol{\Xi}} = -\frac{1}{2}\frac{4}{4+\alpha^2}\left[(\boldsymbol{\alpha}\cdot\dot{\boldsymbol{\alpha}})\,\boldsymbol{\Xi} - \dot{\boldsymbol{A}}\right] \tag{4.5}$$

4.1.3 Cinemática da viga

Para permitir deslocamentos e rotações ilimitadas, abrangendo uma ordem maior de problemas, como por exemplo a formação de *loops* nos *risers*, a formulação cinematicamente exata mostrou ser a forma mais eficiente e natural para atingir este objetivo. Dependendo da magnitude das rotações é necessário utilizar uma descrição lagrangiana atualizada devido às singularidades presentes no tensor rotação associado a alguns valores dos parâmetros de Rodrigues (ou Euler). A Fig. 4.2 mostra esta descrição.

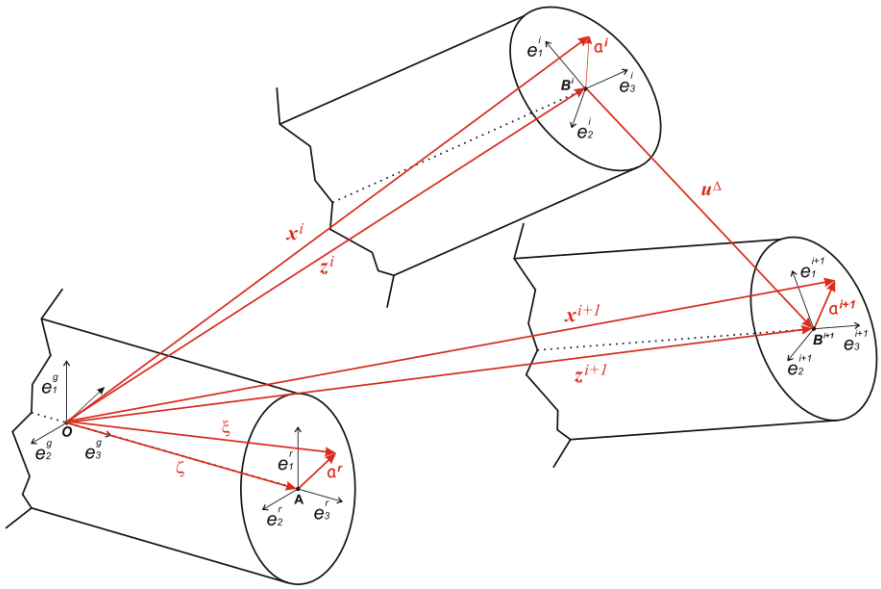


Figura 4.2 – Formulação de vigas utilizando uma descrição lagrangiana atualizada (Gay Neto et al., 2013).

onde

- $\boldsymbol{\zeta}$ é o vetor posição do ponto $\boldsymbol{A}$, definido pela intersecção entre o eixo da viga e a seção transversal que está sendo analisada (em sua configuração de referência);
- $\boldsymbol{\xi}$ é o vetor posição de um ponto genérico da seção transversal considerada da viga em sua configuração de referência;
- $\boldsymbol{a}^r$ é a diferença entre os vetores $\boldsymbol{\xi}$ e $\boldsymbol{\zeta}$. Após o movimento da estrutura se transforma em $\boldsymbol{a}^i$ na configuração "i" e $\boldsymbol{a}^{i+1}$ na configuração "$i+1$";
- $\boldsymbol{z}^i$ e $\boldsymbol{z}^{i+1}$ são os vetores posição dos pontos $\boldsymbol{B}^i$ e $\boldsymbol{B}^{i+1}$, definidos pela interseção entre o eixo da viga e a seção transversal que está sendo analisada, respectivamente nas configurações "i" e "$i+1$";
- $\boldsymbol{x}^i$ e $\boldsymbol{x}^{i+1}$ são os vetores posição de um ponto genérico da seção transversal considerada da viga, respectivamente nas configurações "i" e "$i+1$";
- $\boldsymbol{u}^\Delta$ é o vetor deslocamento que vai do ponto $\boldsymbol{B}^i$ até o ponto $\boldsymbol{B}^{i+1}$, origem dos referenciais locais "i" e "$i+1$", respectivamente;

A relação entre os vetores $\boldsymbol{a}^r$ e $\boldsymbol{a}^i$ pode ser definida utilizando-se do tensor rotação $\boldsymbol{Q}^i$ como $\boldsymbol{a}^i = \boldsymbol{Q}^i\boldsymbol{a}^r$, e a relação entre os vetores $\boldsymbol{a}^i$ e $\boldsymbol{a}^{i+1}$ pode ser definida utilizando-se do tensor $\boldsymbol{Q}^\Delta$ como $\boldsymbol{a}^{i+1} = \boldsymbol{Q}^\Delta\boldsymbol{a}^i$. Portanto é possível escrever uma relação que compõem sucessivas rotações, da seguinte forma

$$\boldsymbol{Q}^{i+1} = \boldsymbol{Q}^\Delta\boldsymbol{Q}^i \tag{4.6}$$

4.1.4 Deformações

As deformações podem ser definidas pelo vetor deformação translacional e pelo vetor rotação específico retro-rotacionados como:

$$\boldsymbol{\eta}^{i+i^r} = \boldsymbol{Q}^{i+1^\mathsf{T}} \boldsymbol{z}^{i+1'} - \boldsymbol{e}_3^r \tag{4.7}$$

$$\boldsymbol{\kappa}^{i+1^r} = \boldsymbol{Q}^{i^\mathsf{T}} \boldsymbol{\Xi}^{\Delta^\mathsf{T}} \boldsymbol{\alpha}^{\Delta'} + \boldsymbol{\kappa}^{i^r} \tag{4.8}$$

O vetor de deformação retro-rotacionado generalizado é definido por

$$\boldsymbol{\epsilon}^{i+1^r} = \begin{bmatrix} \boldsymbol{\eta}^{i+1^r} \\ \boldsymbol{\kappa}^{i+1^r} \end{bmatrix} \tag{4.9}$$

O desenvolvimento matemático destas equações pode ser encontrado em Gay Neto et al. (2013).

4.1.5 Tensões

A descrição das tensões em um modelo de vigas é feito utilizando-se do primeiro tensor de Piola-Kirchhoff de forma a escrever tudo na configuração de referência.

Os vetores de carregamento interno são dados por

$$\boldsymbol{n}^i = V_1^i \boldsymbol{e}_1^i + V_2^i \boldsymbol{e}_2^i + N_3^i \boldsymbol{e}_3^i \tag{4.10}$$

$$\boldsymbol{m}^i = M_1^i \boldsymbol{e}_1^i + M_2^i \boldsymbol{e}_2^i + T_3^i \boldsymbol{e}_3^i \tag{4.11}$$

onde os vetores $\boldsymbol{n}^i$ e $\boldsymbol{m}^i$ representam as forças resultantes internas atuando em uma dada seção transversal no instante "i" e o momento gerado pelo sistema de forças internas em uma dada seção transversal no mesmo instante, respectivamente.

Portanto o vetor de tensões generalizados é definido por

$$\boldsymbol{\sigma}^{i^r} = \begin{bmatrix} \boldsymbol{Q}^{i^\mathsf{T}} \boldsymbol{n}^i \\ \boldsymbol{Q}^{i^\mathsf{T}} \boldsymbol{m}^i \end{bmatrix} \tag{4.12}$$

Desejando-se retro-rotacionar, os vetores tornam-se

$$\boldsymbol{n}^{i^r} = \boldsymbol{Q}^{i^\mathsf{T}} \boldsymbol{n}^i = V_1^i \boldsymbol{e}_1^r + V_2^i \boldsymbol{e}_2^r + N_3^i \boldsymbol{e}_3^r \tag{4.13}$$

$$\boldsymbol{m}^{i^r} = \boldsymbol{Q}^{i^\mathsf{T}} \boldsymbol{m}^i = M_1^i \boldsymbol{e}_1^r + M_2^i \boldsymbol{e}_2^r + T_3^i \boldsymbol{e}_3^r \tag{4.14}$$

$$\tag{4.15}$$

4.1.6 Contato

A cinemática de contato foi desenvolvida utilizando uma descrição *master-slave*. O fundo do oceano é considerado como a superfície *master*. Os nós dos elementos finitos do *riser* utilizados na discretização do modelo de vigas são considerados como *slave points*.

A teoria completa desenvolvida para a descrição do modelo de contato pode ser encontrada em Gay Neto et al. (2013).

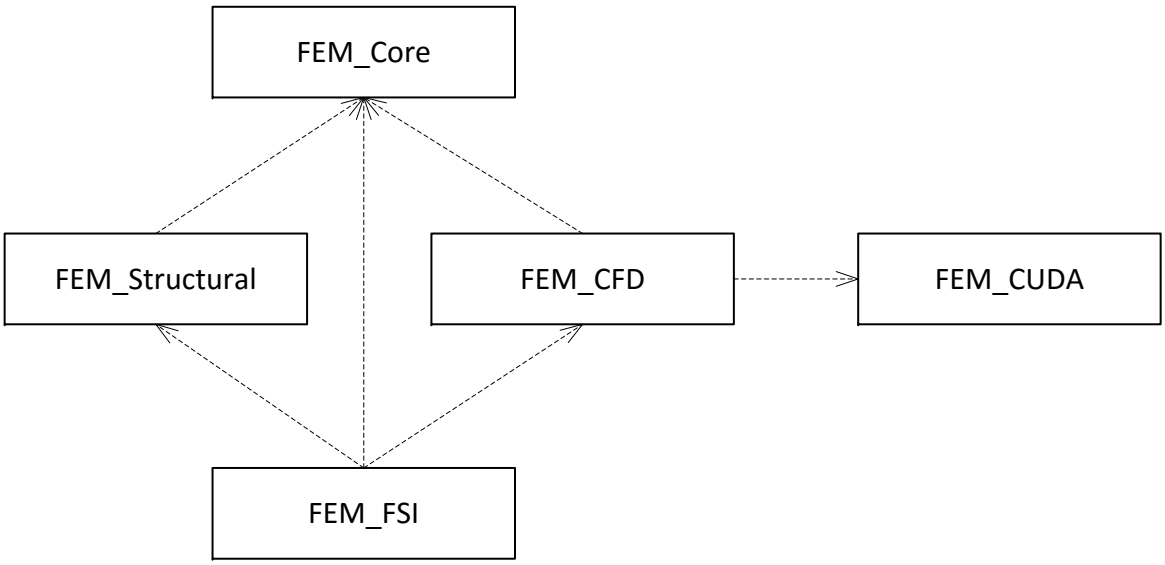
5 IMPLEMENTAÇÃO COMPUTACIONAL

Este capítulo apresenta a implementação computacional do *software* desenvolvido neste trabalho, junto com uma apresentação da arquitetura, as classes UML (*Unified Modeling Language*) e os respectivos diagramas de sequência. Em Fowler (2004) pode-se encontrar uma referência completa sobre UML.

5.1 Arquitetura

A arquitetura do *software* foi dividida em cinco pacotes como mostra a Fig. 5.1. O pacote *FEM_CUDA* foi criado para utilizar a plataforma de computação paralela CUDA quando o computador ou servidor que estiver executando o *software* possua uma placa de vídeo com GPU da NVIDIA. Neste trabalho somente a resolução dos problemas de dinâmica dos fluidos e interação fluido-estrutura utilizou a linguagem CUDA, mas pode ser facilmente estendido para a resolução de problemas estruturais.

Figura 5.1 – Arquitetura dividida em pacotes ou módulos.



Fonte: O autor

Desta forma pode-se separar o código-fonte em módulos reutilizáveis facilitando a extensão para novas funcionalidades, como por exemplo, dinâmica estrutural (um pacote ou módulo como extensão de *FEM_Structural*) ou turbulência (um pacote ou módulo como extensão de *FEM_CFD*), entre outros.

5.1.1 Diagrama de classes UML

Esta seção apresenta os diagramas de classes UML utilizados na arquitetura separados pelos diferentes pacotes (*FEM_Core*, *FEM_Structural*, *FEM_CFD* e *FEM_FSI*).

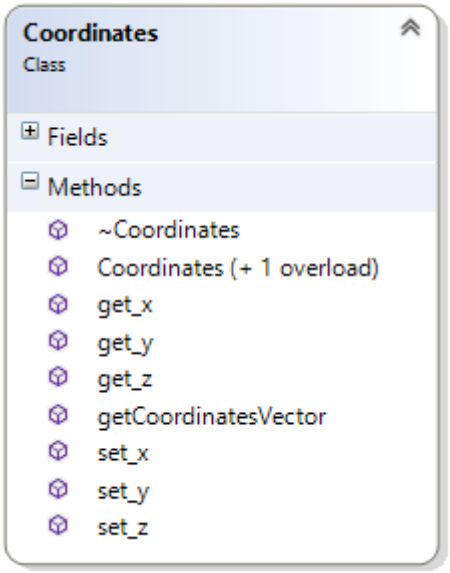
5.1.1.1 Pacote *FEM_Core*

Este pacote armazena as principais classes da arquitetura. Estas classes são chamadas de *cross classes*, ou seja, classes que são utilizadas por toda a arquitetura e normalmente são definidas como classes abstratas[1] ou possuem métodos virtuais[2].

As seguintes classes pertencem ao pacote *FEM_Core*:

- *Coordinates*: Esta classe é responsável por armazenar as coordenadas dos nós da malha de elementos finitos (Fig. 5.2).

Figura 5.2 – Classe *Coordinates*.



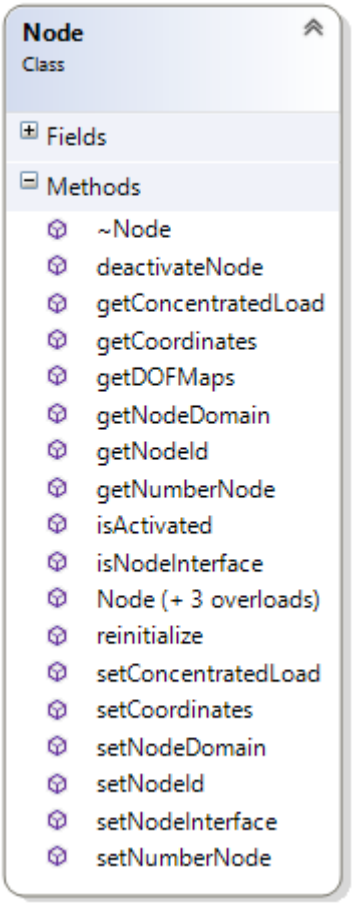
Fonte: O autor

- *Node*: Esta classe é responsável por armazenar os nós da malha de elementos finitos (Fig. 5.3). Os graus de liberdade da estrutura, tanto local quanto global, também são armazenados nesta classe.

[1] As classes abstratas são classes que não podem ser instanciadas, somente herdadas, sendo necessário que as classes filhas sobrescrevam os métodos da classe pai.

[2] Os métodos virtuais são métodos que podem ser sobrescritos pelas classes filhas.

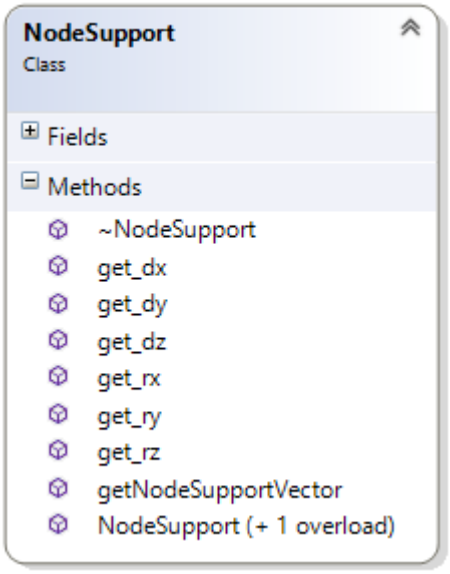
Figura 5.3 – Classe *Node*.



Fonte: O autor

- *NodeSupport*: Esta classe é responsável por definir as restrições aplicadas ao nó da malha de elementos finitos (Fig. 5.3), equivalente às condições de contorno de *Dirichlet*.

Figura 5.4 – Classe *NodeSupport*.

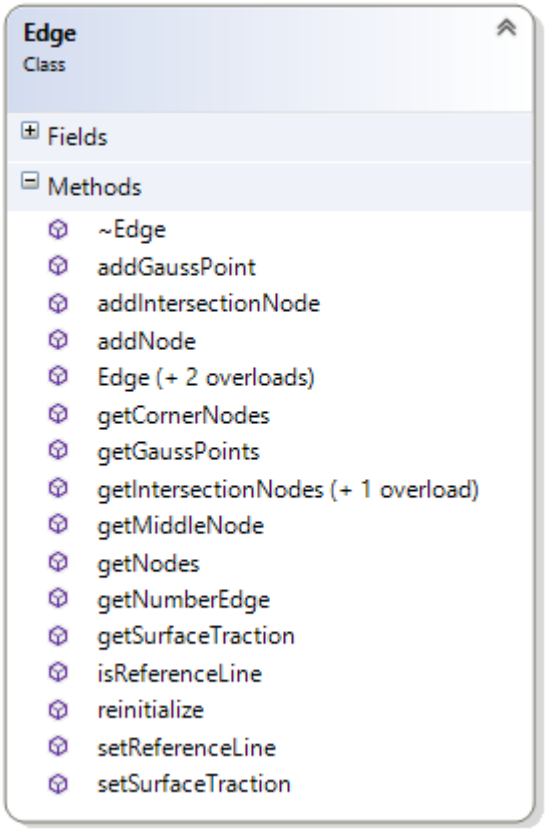


Fonte: O autor

- *Edge*: Esta classe é responsável por armazenar os lados[3] do elemento finito (Fig. 5.5). Esta classe também é responsável por armazenar os nós de intersecção entre a superfície molhada e a malha de fluido, e os pontos de Gauss para a integração no polígono como apresentado em capítulos anteriores.

[3]Para as barras, a classe Edge armazena o próprio elemento finito. No caso de elementos bidimensionais, a classe Edge armazena os lados do elemento. Para elementos tridimensionais, a classe Edge armazena os lados das faces do elemento.

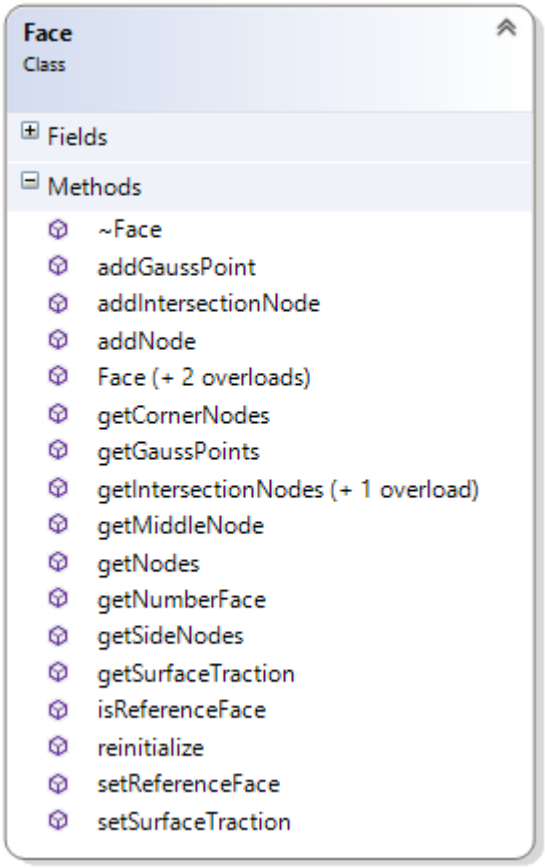
Figura 5.5 – Classe *Edge*.



Fonte: O autor

- *Face*: Esta classe é responsável por armazenar as faces do elemento finito (Fig. 5.6). Da mesma forma que a classe *Edge*, esta classe também é responsável por armazenar os nós de intersecção entre a superfície molhada e a malha de fluido, e os pontos de Gauss para a integração no polígono como apresentado em capítulos anteriores.

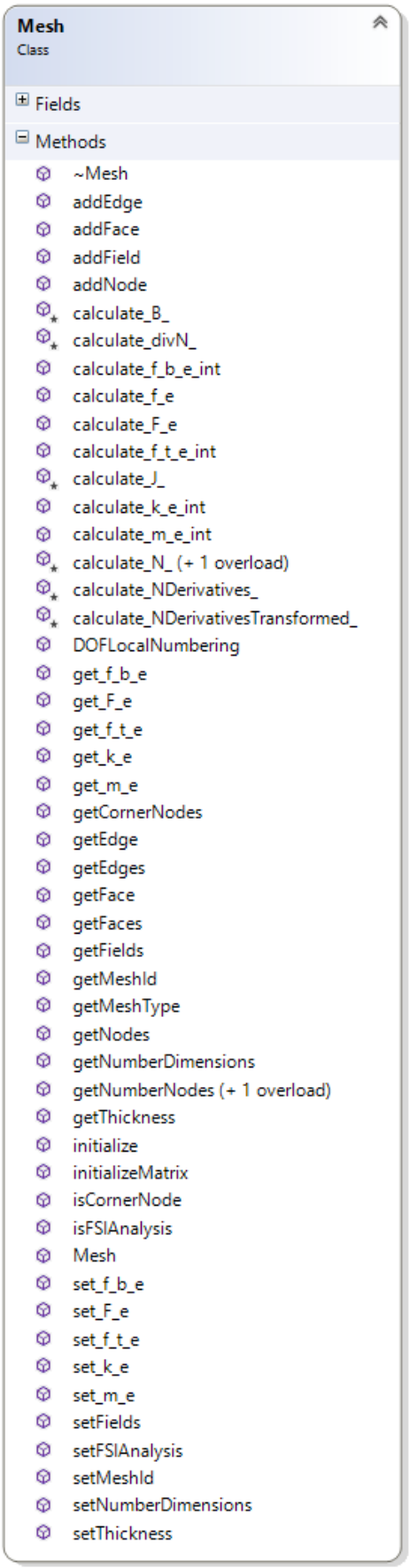
Figura 5.6 – Classe *Face*.



Fonte: O autor

- *Mesh*: Esta classe é responsável por armazenar o elemento finito (Fig. 5.7). É a classe mais importante do pacote *FEM_Core* pois realiza os principais cálculos pelo método dos elementos finitos, como as funções de forma, suas derivadas, o Jacobiano da transformação quando utiliza-se elementos finitos isoparamétricos, como também é responsável pela contribuição local do elemento na matriz global da estrutura.

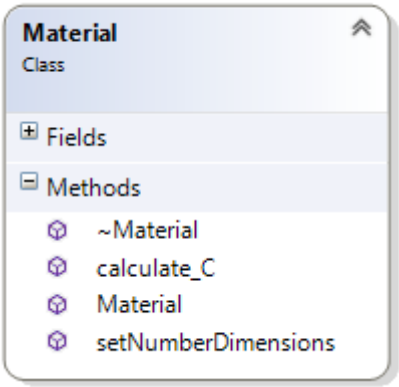
Figura 5.7 – Classe *Mesh*.



Fonte: O autor

- *Material*: Esta classe é responsável por armazenar o material do elemento finito (Fig. 5.8). As classes dos pacotes *FEM_Strutural* e *FEM_CFD* herdam esta classe para especializá-las para os materiais estruturais e de fluido.

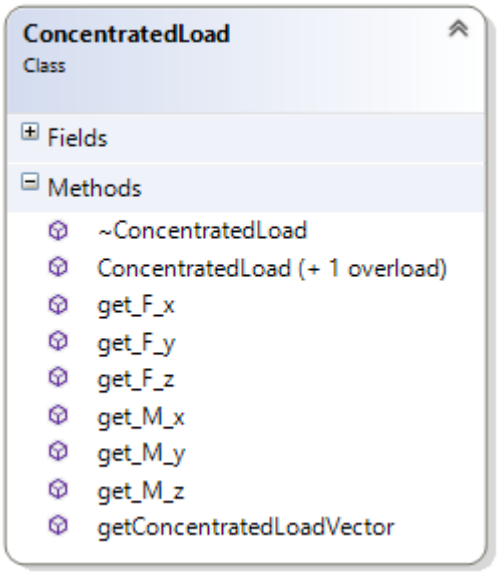
Figura 5.8 – Classe Material.



Fonte: O autor

- *ConcentratedLoad*: Esta classe é responsável por armazenar a força concentrada aplicada ao nó da malha de elementos finitos (Fig. 5.9).

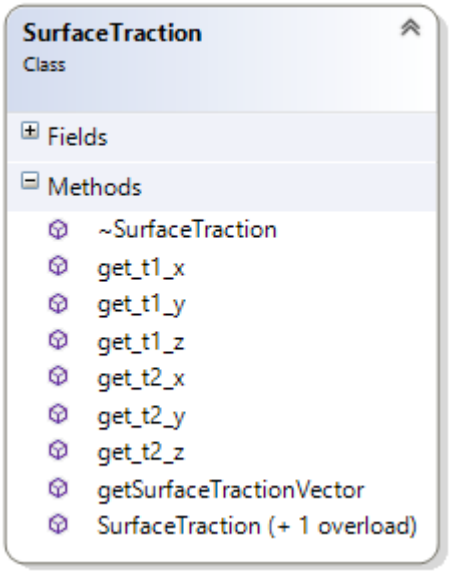
Figura 5.9 – Classe *ConcentratedLoad*.



Fonte: O autor

- *SurfaceTraction*: Esta classe é responsável por armazenar a força distribuída aplicada a um lado ou a uma face de um elemento finito (Fig. 5.10).

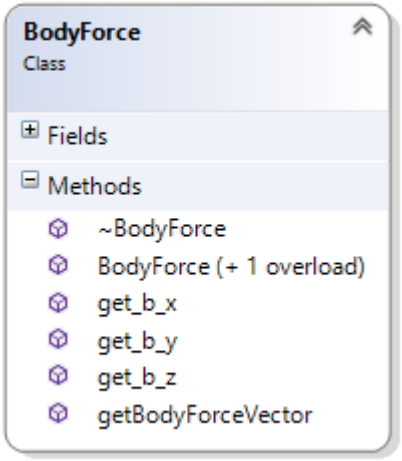
Figura 5.10 – Classe *SurfaceTraction*.



Fonte: O autor

- *BodyForce*: Esta classe é responsável por armazenar a força de campo aplicada a um elemento finito (Fig. 5.11).

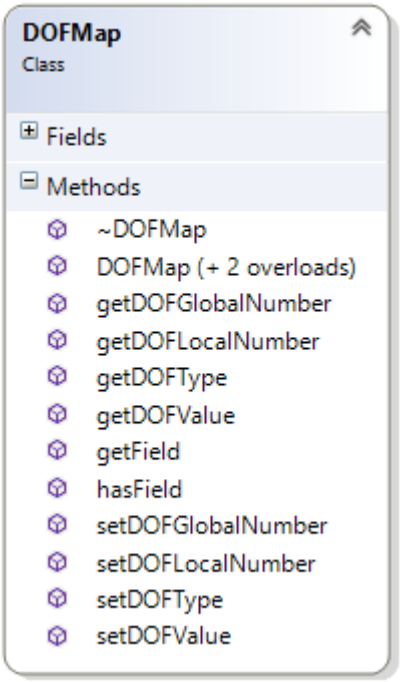
Figura 5.11 – Classe *BodyForce*.



Fonte: O autor

- *DOFMap*: Esta classe é responsável por mapear os graus de liberdade locais para os graus de liberdade globais da estrutura (Fig. 5.12).

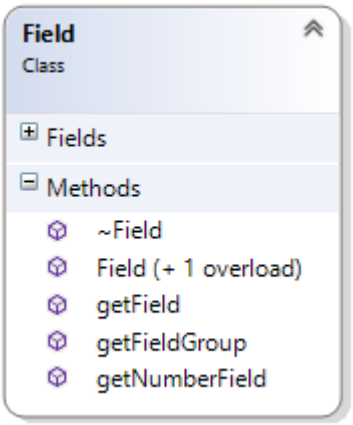
Figura 5.12 – Classe *DOFMap*.



Fonte: O autor

- *Field*: Esta classe é responsável por armazenar os diversos campos[4] que estão atuando em um elemento finito (Fig. 5.13). No caso do CFD, como utiliza-se elementos finitos mistos é necessário diferenciar o mesmo nó de um elemento como ele sendo um nó que possui informações de velocidade como também de pressão (Cap. 3).

Figura 5.13 – Classe *Field*.

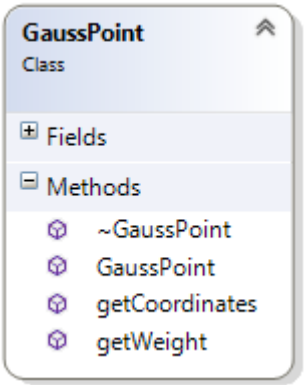


Fonte: O autor

- *GaussPoint*: Esta classe é responsável por armazenar os pontos de Gauss utilizados na integração no polígono como apresentado em capítulos anteriores (Fig. 5.14).

[4]Estes campos podem ser tanto escalares como vetoriais.

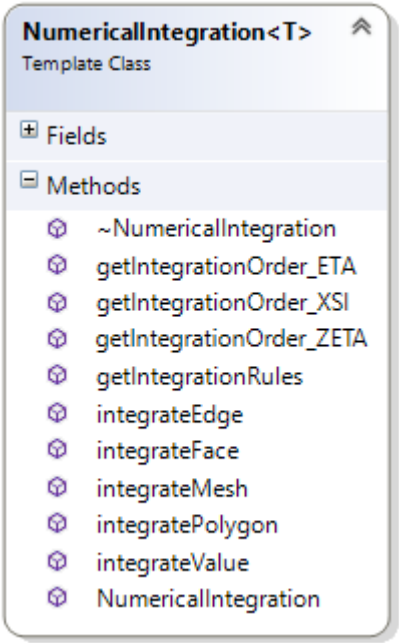
Figura 5.14 – Classe *GaussPoint*.



Fonte: O autor

- *NumericalIntegration*: Esta classe é responsável por realizar a integração numérica utilizada pelo método dos elementos finitos (Fig. 5.15). Pode ser herdada e especializada para realizar não somente integração utilizando as quadraturas de Gauss como também outras quadraturas ou mesmo outros métodos de integração.

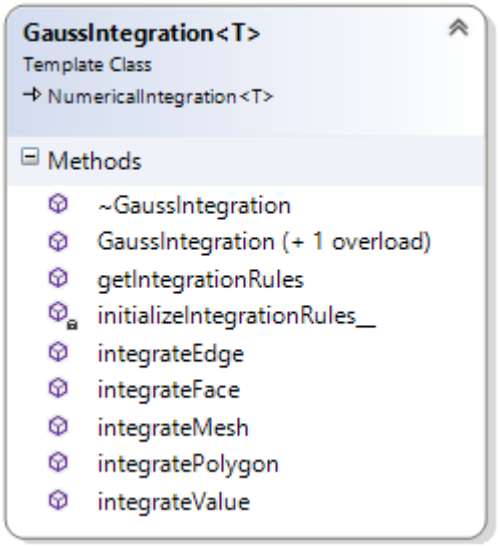
Figura 5.15 – Classe *NumericalIntegration*.



Fonte: O autor

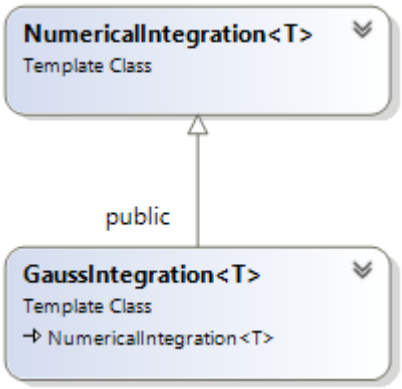
- *GaussIntegration*: Esta classe é responsável por realizar a integração numérica utilizando a quadratura de Gauss (Fig. 5.16 e 5.17). Esta classe herda a classe pai *NumericalIntegration*.

Figura 5.16 – Classe *GaussIntegration*.



Fonte: O autor

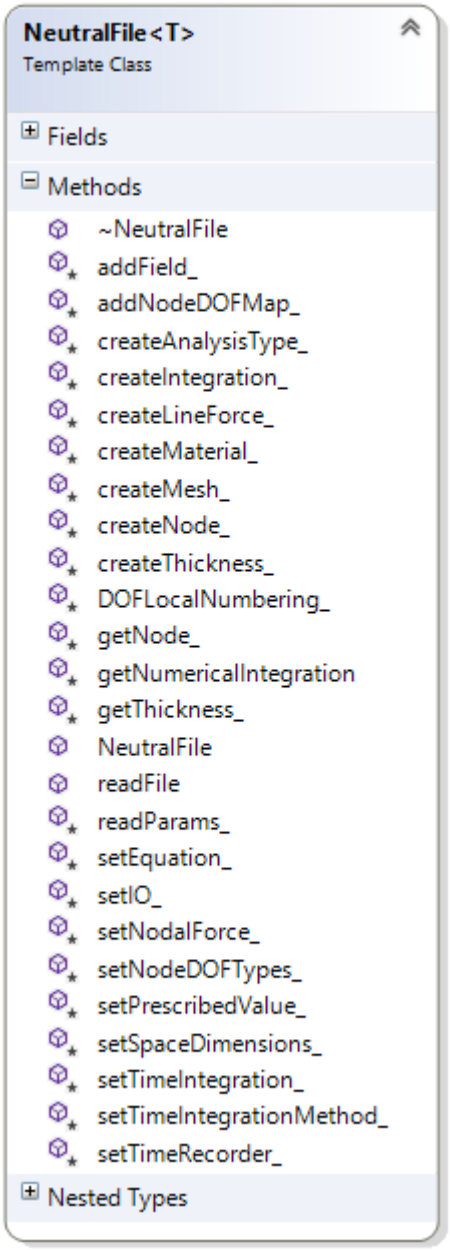
Figura 5.17 – Hierarquia da classe *GaussIntegration*.



Fonte: O autor

- *NeutralFile*: Esta classe é responsável por realizar a leitura do arquivo neutro onde são definidas as configurações de execução (coordenadas dos nós, elementos finitos, tipo de execução, entre outras) (Fig. 5.18). No Apêndice B é apresentado um arquivo neutro como exemplo e uma explicação de cada seção deste arquivo.

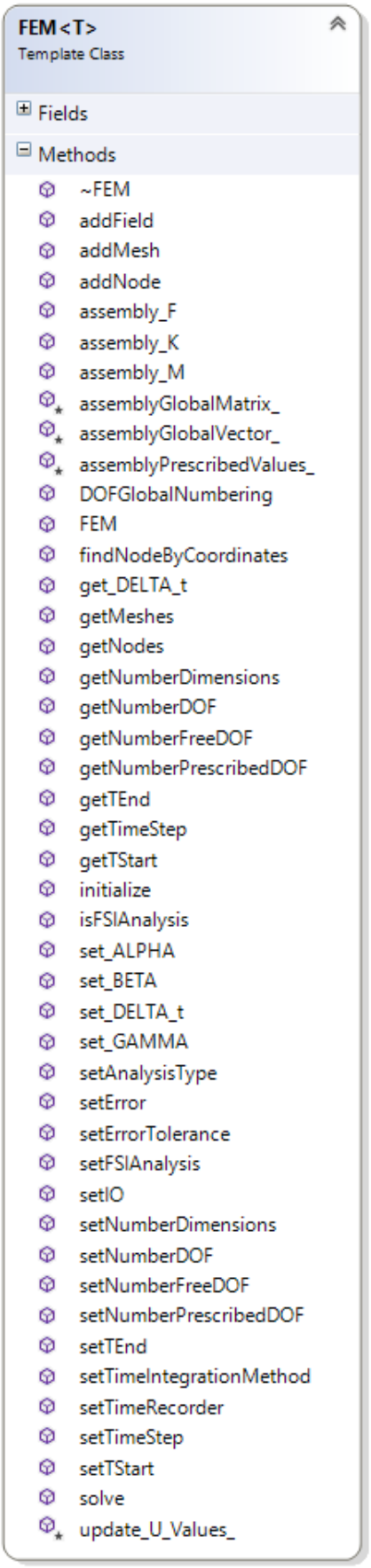
Figura 5.18 – Classe *NeutralFile*.



Fonte: O autor

- *FEM*: Esta classe é responsável por realizar o cálculo pelo método dos elementos finitos tanto linear como não-linear (Fig. 5.19).

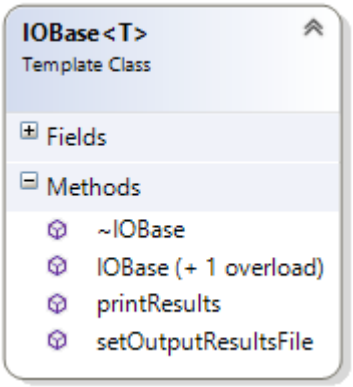
Figura 5.19 – Classe *FEM*.



Fonte: O autor

- *IOBase*: Esta classe é responsável por salvar os resultados para pós-processamento em um formato adequado (Fig. 5.20). Neste trabalho dois formatos foram utilizados, um formato pré-definido pelo GiD (MELENDO et al., 2015) a ser utilizado pelo seu pós-processador, de mesmo nome, e um outro formato pré-definido pelo VTK (SCHROEDER et al., 2006) para ser visualizado no pós-processador ParaView (AHRENS et al., 2005).

Figura 5.20 – Classe *IOBase*.



Fonte: O autor

5.1.1.2 Pacote *FEM_Structural*

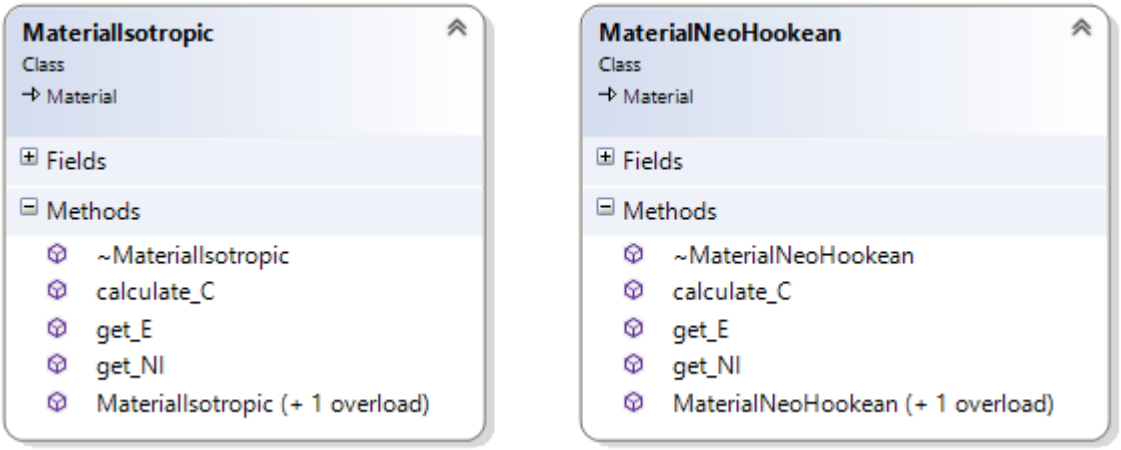
Este pacote possui as classes responsáveis pela análise pelo método dos elementos finitos de uma estrutura qualquer. Elas herdam algumas classes do pacote ou módulo *FEM_Core* e sobrescrevem seus métodos especializando-os.

No caso da interação fluido-estrutura pelo método das Fronteiras Imersas, o pacote *FEM_Structural* é responsável por armazenar informações como a superfície molhada e o polígono de intersecção, resultado da intersecção entre a estrutura e a malha de fluido, entre outras.

As seguintes classes pertencem ao pacote *FEM_Structural*:

- *MaterialIsotropic* e *MaterialNeoHookean*: Estas classes são responsáveis por armazenar os materiais isotrópicos e neo-hookeanos de uma estrutura (Fig. 5.21). Elas herdam a classe pai *Material* do pacote *FEM_Core*.

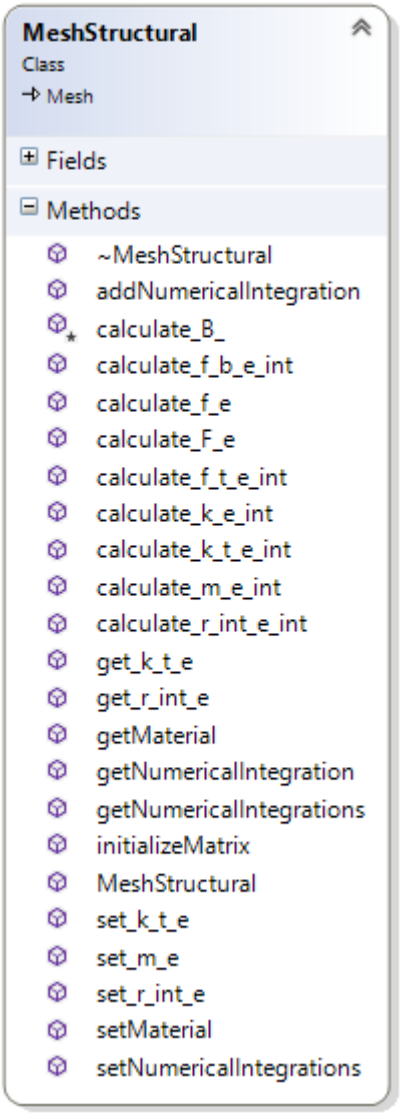
Figura 5.21 – Classes *MaterialIsotropic* e *MaterialNeoHookean*.



Fonte: O autor

- *MeshStructural*: Esta classe é responsável por armazenar o elemento finito de uma estrutura (Fig. 5.22). Esta classe herda a classe pai *Mesh* do pacote *FEM_Core*.

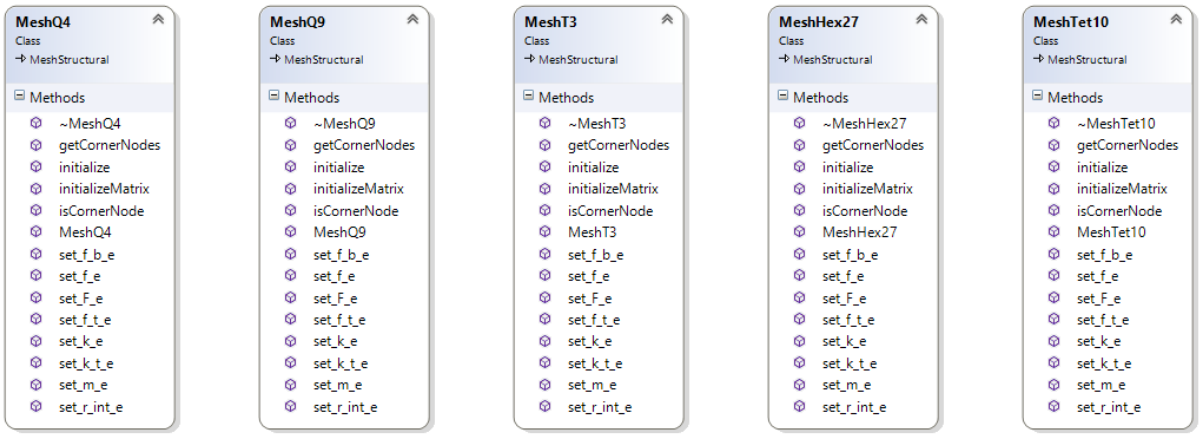
Figura 5.22 – Classe MeshStructural.



Fonte: O autor

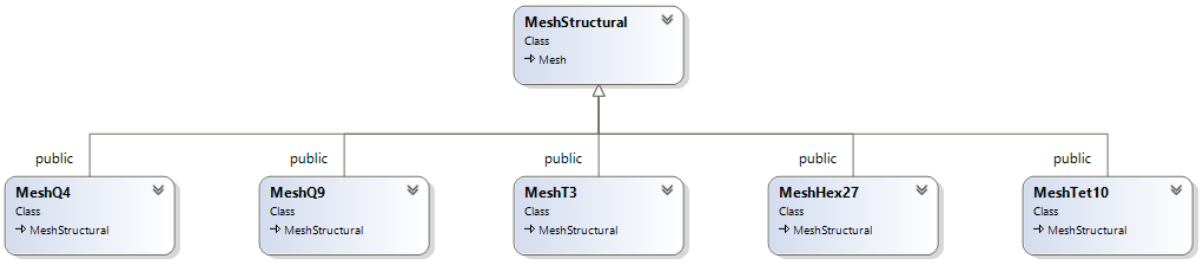
- *MeshT3*, *MeshQ4*, *MeshQ9*, *MeshTet10* e *MeshHex27*: Estas classes são responsáveis por armazenar elementos finitos de uma estrutura dos tipos, triangular de 3 nós, quadrangular de 4 e de 9 nós, tetraédrico de 10 nós e hexaédrico de 27 nós (Fig. 5.23 e 5.24). Elas herdam a classe pai *MeshStructural*.

Figura 5.23 – Classes *MeshT3*, *MeshQ4*, *MeshQ9*, *MeshTet10* e *MeshHex27*.



Fonte: O autor

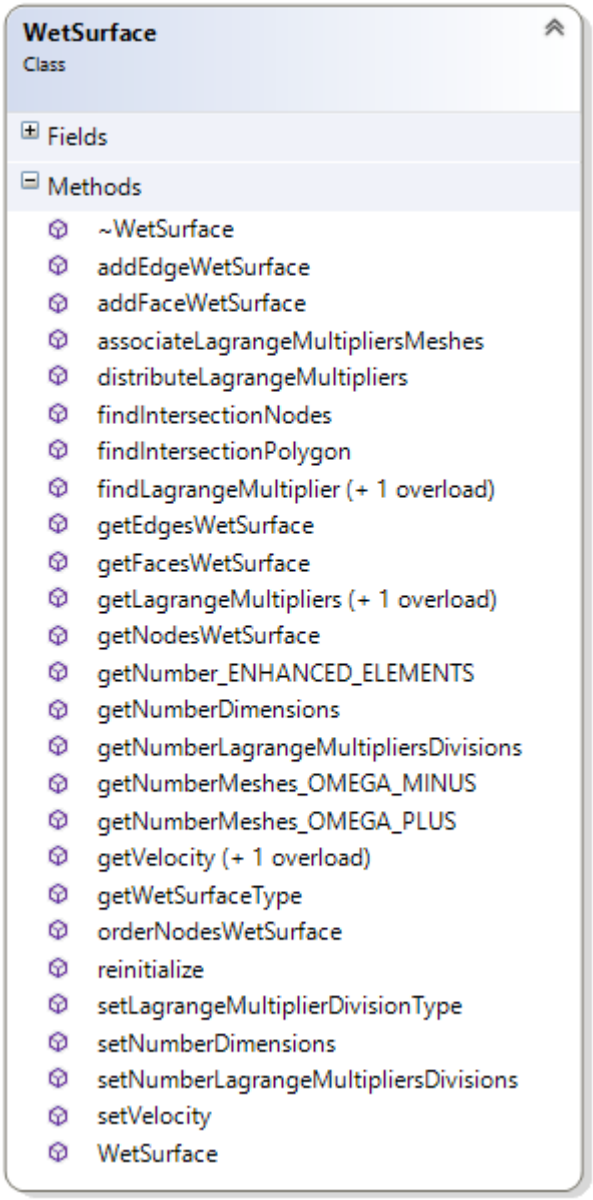
Figura 5.24 – Hierarquia de classes dos diferentes elementos finitos de uma estrutura.



Fonte: O autor

- *WetSurface*: Esta classe é responsável por armazenar a superfície molhada da estrutura, como resultado da intersecção da malha da estrutura com a malha do fluido (Fig. 5.25). Nesta classe armazena-se os multiplicadores de Lagrange (método da Fronteira Imersa) como discutido em capítulos anteriores.

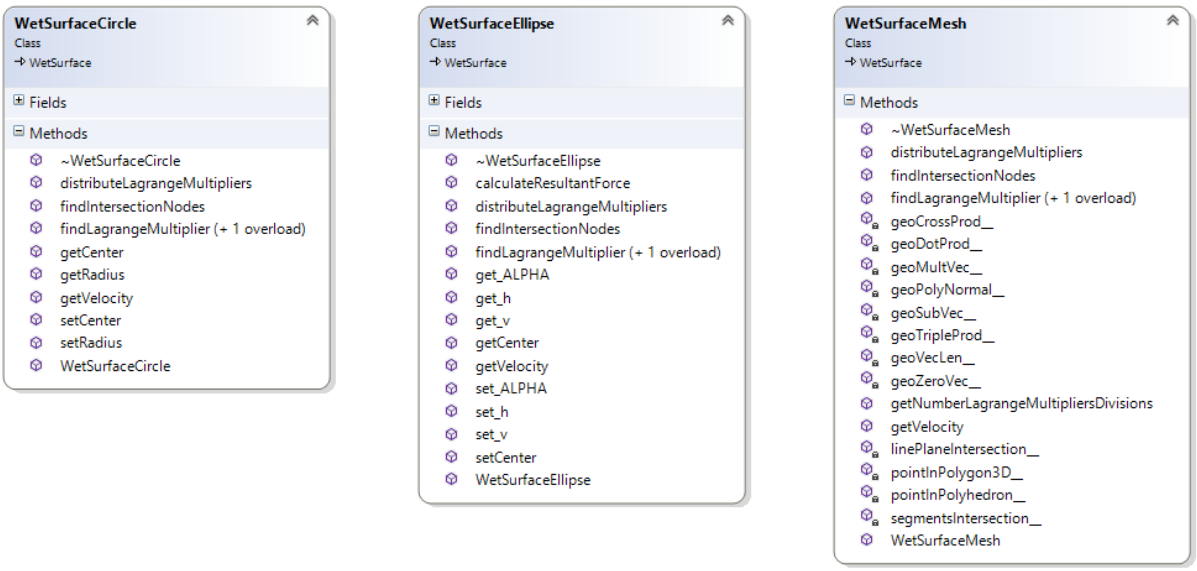
Figura 5.25 – Classe *WetSurface*.



Fonte: O autor

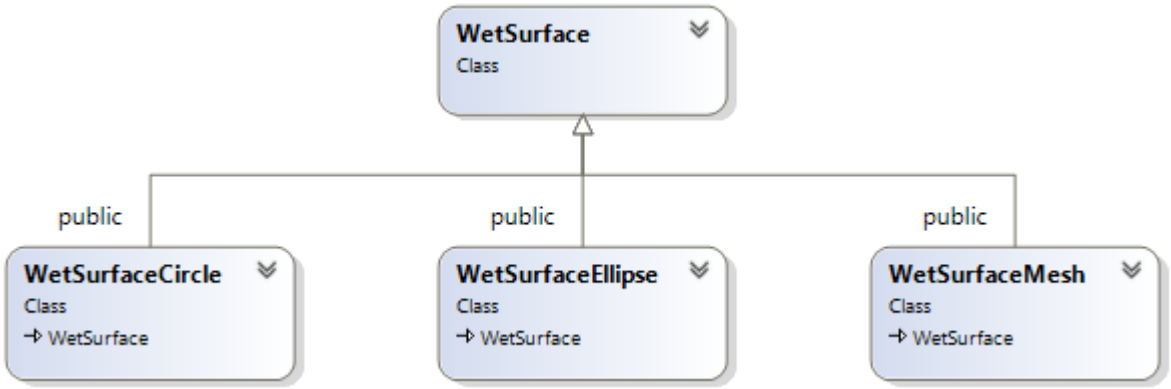
- *WetSurfaceCircle*, *WetSurfaceEllipse*, *WetSurfaceMesh*: Estas classes são responsáveis por armazenar a superfície molhada para os seguintes tipos: círculo e elipse, ou uma malha de elementos finitos qualquer (Fig. 5.26 e 5.27). Elas herdam a classe pai *WetSurface*.

Figura 5.26 – Classes *WetSurfaceCircle*, *WetSurfaceEllipse* e *WetSurfaceMesh*.



Fonte: O autor

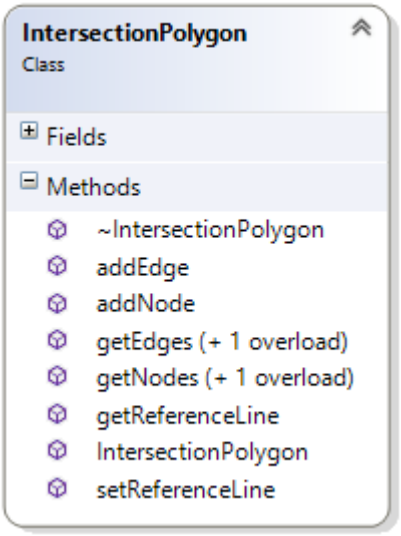
Figura 5.27 – Hierarquia de classes da superfície molhada.



Fonte: O autor

- *IntersectionPolygon*: Esta classe é responsável por armazenar o polígono de intersecção a ser utilizado pela integração no polígono como discutido em capítulos anteriores (Fig. 5.28).

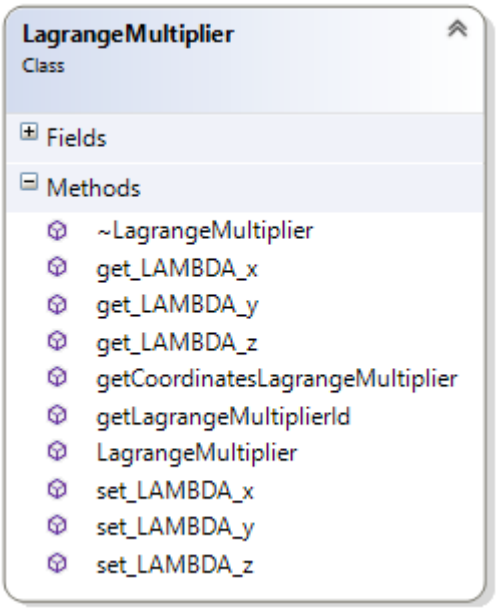
Figura 5.28 – Classe *IntersectionPolygon*.



Fonte: O autor

- *LagrangeMultiplier*: Esta classe é responsável por armazenar o multiplicador de Lagrange a ser utilizado pelo método da Fronteira Imersa como discutido em capítulos anteriores (Fig. 5.29).

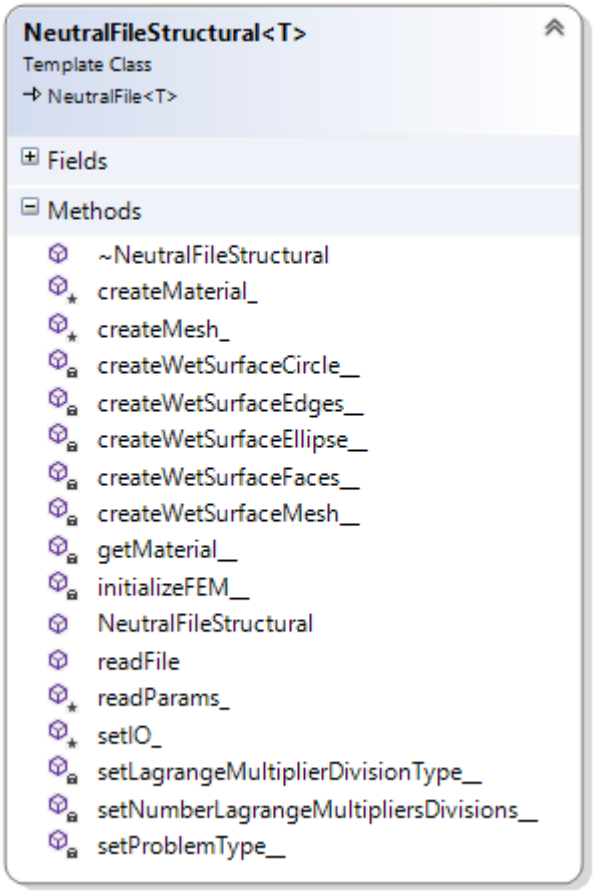
Figura 5.29 – Classe *LagrangeMultiplier*.



Fonte: O autor

- *NeutralFileStructural*: Esta classe é responsável por realizar a leitura do arquivo neutro onde são definidas as configurações de execução (coordenadas dos nós, elementos finitos, tipo de execução, entre outras) (Fig. 5.30). No Apêndice B é apresentado um arquivo neutro como exemplo e uma explicação de cada seção deste arquivo. Esta classe herda e especializa a classe pai *NeutralFile* do pacote *FEM_Core*.

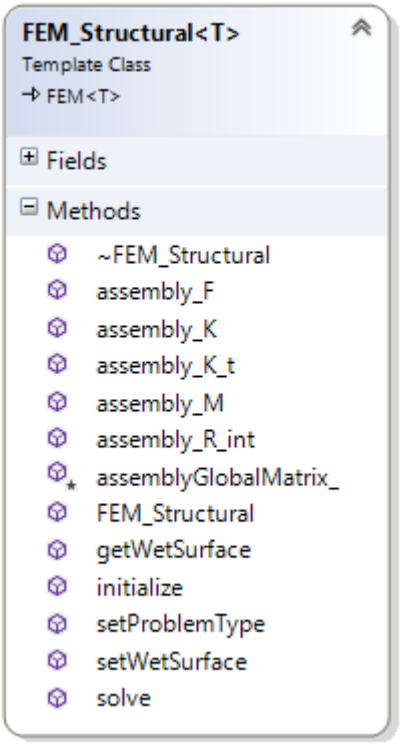
Figura 5.30 – Classe *NeutralFileStructural*.



Fonte: O autor

- *FEM_Structural*: Esta classe é responsável por realizar o cálculo pelo método dos elementos finitos de uma estrutura qualquer tanto linear como não-linear (Fig. 5.31). Esta classe herda e especializa a classe pai *FEM* do pacote *FEM_Core*.

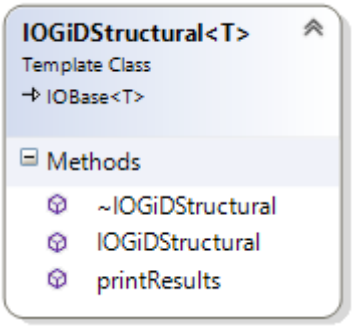
Figura 5.31 – Classe *FEM_Structural*.



Fonte: O autor

- *IOGiDStructural*: Esta classe é responsável por salvar os resultados para pós-processamento em um formato adequado (Fig. 5.32). Neste trabalho dois formatos foram utilizados, um formato pré-definido pelo GiD (MELENDO et al., 2015) a ser utilizado pelo seu pós-processador, de mesmo nome, e um outro formato pré-definido pelo VTK (SCHROEDER et al., 2006) para ser visualizado no pós-processador ParaView (AHRENS et al., 2005). Esta classe herda e especializa a classe pai *IOBase* do pacote *FEM_Core*.

Figura 5.32 – Classe *IOGiDStructural*.



Fonte: O autor

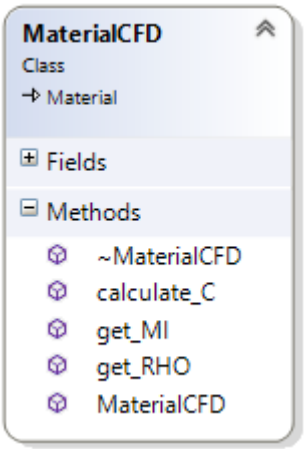
5.1.1.3 Pacote *FEM_CFD*

Este pacote possui as classes responsáveis pela análise pelo método dos elementos finitos de uma malha de fluido. Elas herdam algumas classes do pacote ou módulo *FEM_Core* e sobrescrevem seus métodos especializando-os.

As seguintes classes pertencem ao pacote *FEM_CFD*:

- *MaterialCFD*: Esta classe é responsável por armazenar o material de um fluido (Fig. 5.33). Ela herda da classe pai *Material* do pacote *FEM_Core*.

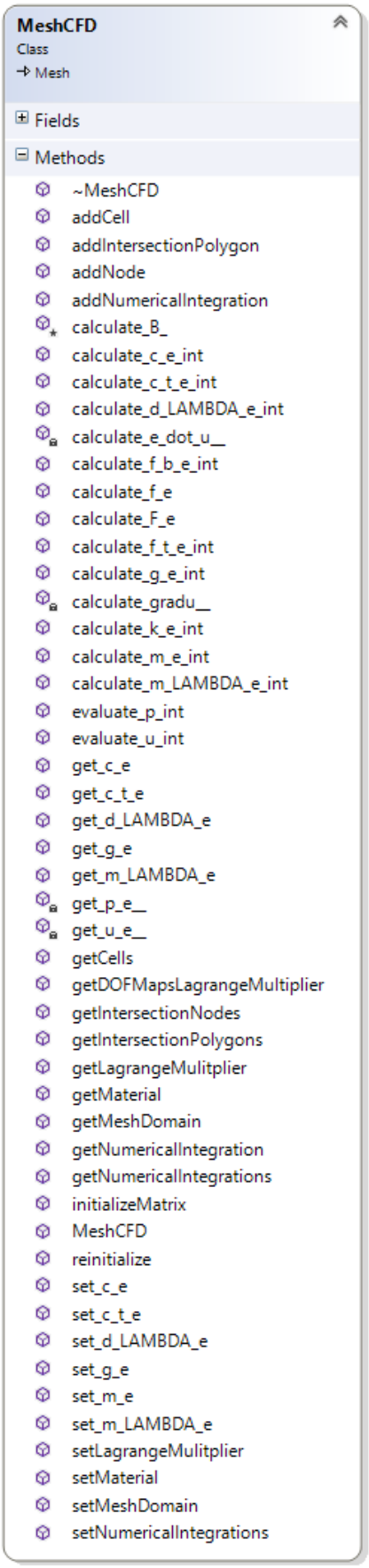
Figura 5.33 – Classe *MaterialCFD*.



Fonte: O autor

- *MeshCFD*: Esta classe é responsável por armazenar o elemento finito de fluido (Fig. 5.34). Ela herda a classe pai *Mesh* do pacote *FEM_Core*.

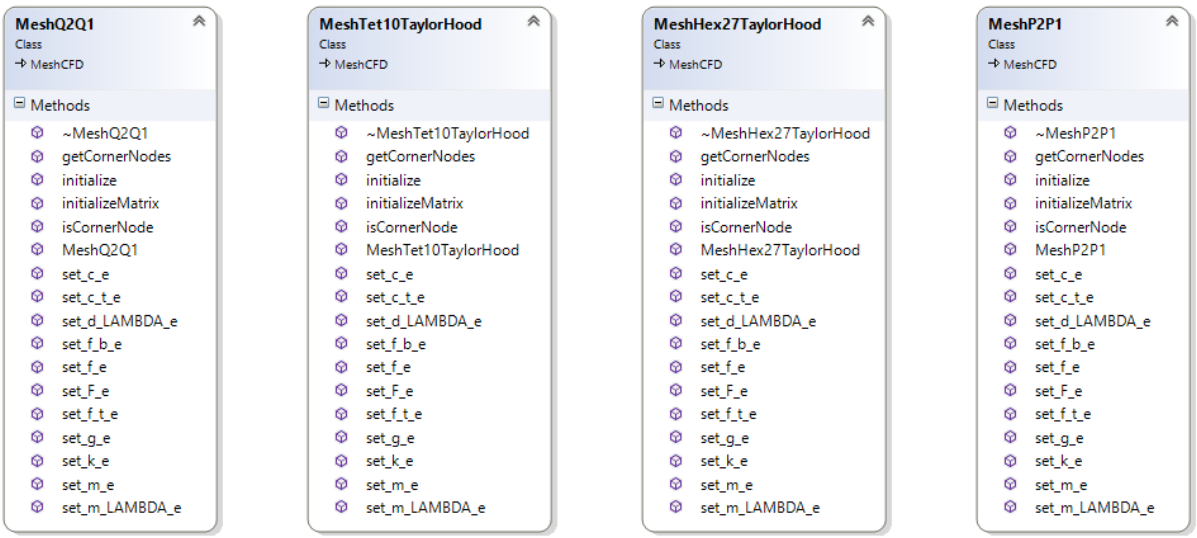
Figura 5.34 – Classe *MeshCFD*.



Fonte: O autor

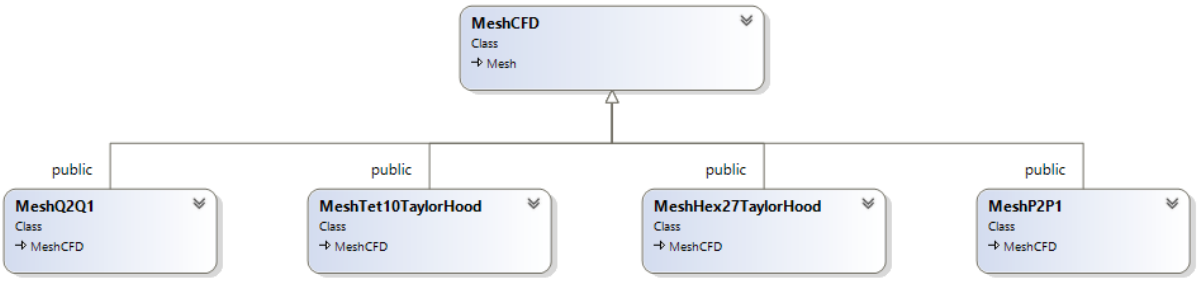
- *MeshP2P1*, *MeshQ2Q1*, *MeshTet10TaylorHood* e *MeshHex27TaylorHood*: Estas classes são responsáveis por armazenar elementos finitos de fluido dos tipos, triangular híbrido de 6 nós (6 nós para velocidade e 3 nós para pressão), quadrangular híbrido 9 nós (9 nós para velocidade e 4 nós para pressão), tetraédrico de 10 nós (10 nós para velocidade e 4 nós para pressão) e hexaédrico de 27 nós (27 nós para velocidade e 8 nós para pressão) (Fig. 5.35 e 5.36). Elas herdam a classe pai *MeshCFD*.

Figura 5.35 – Classes *MeshP2P1*, *MeshQ2Q1*, *MeshTet10TaylorHood* e *MeshHex27TaylorHood*.



Fonte: O autor

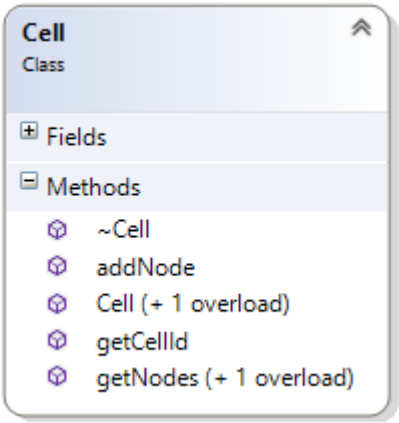
Figura 5.36 – Hierarquia de classes dos elementos finitos de fluido.



Fonte: O autor

- *Cell*: Esta classe é responsável por armazenar a célula de integração quando utilza-se a integração pelo método *Tessellation* (Fig. 5.37).

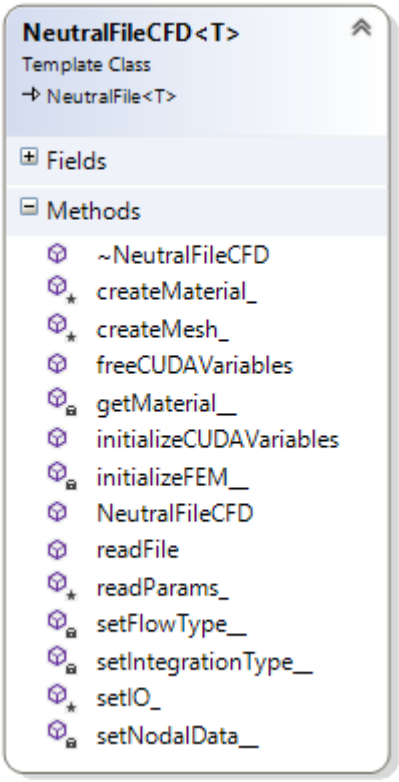
Figura 5.37 – Classe *Cell*.



Fonte: O autor

- *NeutralFileCFD*: Esta classe é responsável por realizar a leitura do arquivo neutro onde são definidas as configurações de execução (coordenadas dos nós, elementos finitos, tipo de execução, entre outras) (Fig. 5.38). No Apêndice B é apresentado um arquivo neutro como exemplo e uma explicação de cada seção deste arquivo. Esta classe herda e especializa a classe pai *NeutralFile* do pacote *FEM_Core*.

Figura 5.38 – Classe *NeutralFileCFD*.

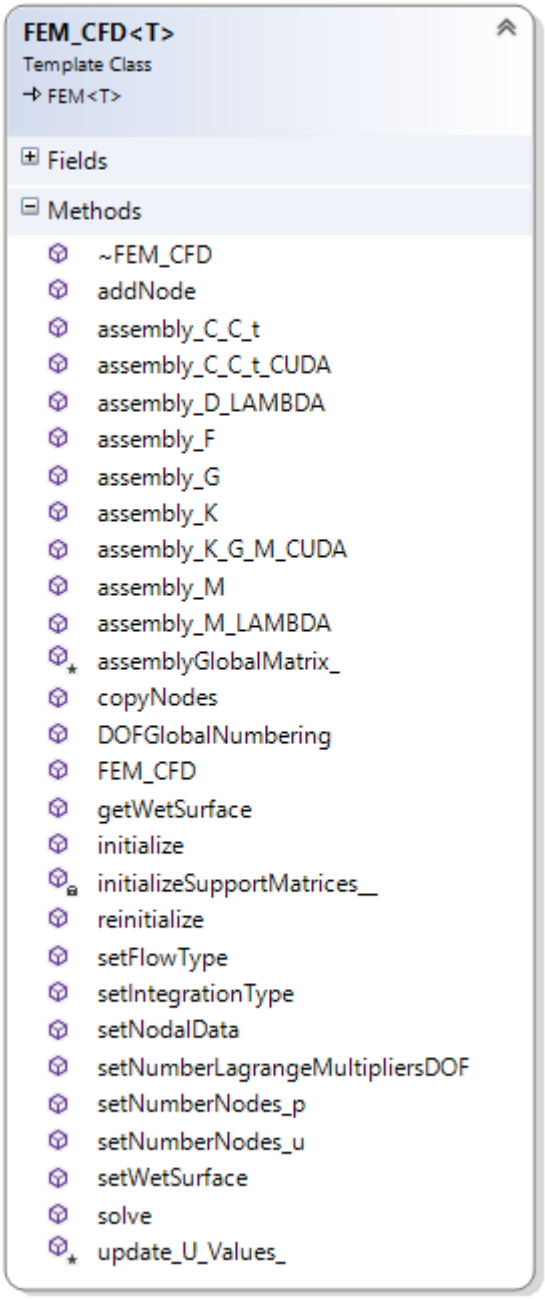


Fonte: O autor

- *FEM_CFD*: Esta classe é responsável por realizar o cálculo pelo método dos elementos

finitos de um fluido (Fig. 5.39). Esta classe herda e especializa a classe pai *FEM* do pacote *FEM_Core*.

Figura 5.39 – Classe *FEM_CFD*.

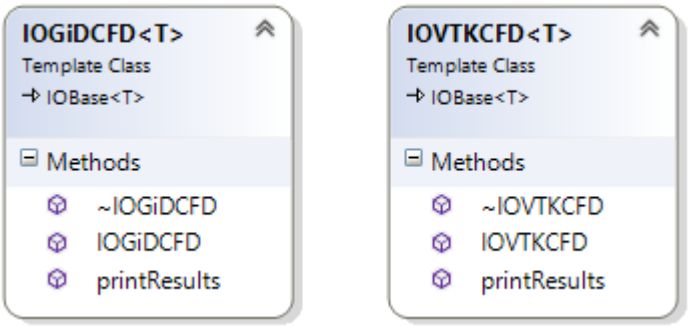


Fonte: O autor

- *IOGiDCFD* e *IOVTKCFD*: Estas classes são responsáveis por salvar os resultados para pós-processamento em um formato adequado (Fig. 5.40). Neste trabalho dois formatos foram utilizados, um formato pré-definido pelo GiD (MELENDO et al., 2015) a ser utilizado pelo seu pós-processador, de mesmo nome, e um outro formato pré-definido pelo VTK (SCHROEDER et al., 2006) para ser visualizado no pós-

processador ParaView (AHRENS et al., 2005). Esta classe herda e especializa a classe pai *IOBase* do pacote *FEM_Core*.

Figura 5.40 – Classes *IOGiDCFD* e *IOVTKCFD*.



Fonte: O autor

5.1.1.4 Pacote *FEM_CUDA*

Este pacote possui as classes responsáveis pelo *assembly* das matrizes locais na resoluçãop de problemas de dinâmica dos fluidos e interação fluido-estrutura. Os seguintes arquivos .cuh ao pacote *FEM_CUDA*[5]:

- *CUDA_Macros.cuh*: Este arquivo possui funções para verificação de erros associados a utilização da GPU como a transformação da linha e coluna das matrizes para acesso serial e contíguo a memória da placa de vídeo.

```
...
#define IDX2C(i, j, ld) (((j) * (ld)) + (i))
...
```

```
...
static void checkCudaErrors(cudaError_t err,
                            const char *file,
                            int line ) {

    if (err != cudaSuccess) {
        printf("%s in %s at line %d\n", cudaGetErrorString(err),
            file, line);
    }
}
...
```

- *CUDA_MatrixOperation.cuh*: Este arquivo possui funções auxiliares para manipulação das matrizes, como soma, subtração, determinante, entre outras.

```
...
//----------------------------- sum ----------------------------------------
__device__
void sum(   int                 M,
            int                 N,
            const Precision*    A,
```

[5]No Apêndice B

```
            const Precision*    B,
            Precision*          C)
{
    for (int m = 0; m < M; ++m)
    {
        for (int n = 0; n < N; ++n)
        {
            C[IDX2C(m, n, M)] = A[IDX2C(m, n, M)] + B[IDX2C(m, n, M)];
        }
    }
}

//------------------------------ subtract ------------------------------------
__device__
void subtract(  int                 M,
                int                 N,
                const Precision*    A,
                const Precision*    B,
                Precision*          C)
{
    for (int m = 0; m < M; ++m)
    {
        for (int n = 0; n < N; ++n)
        {
            C[IDX2C(m, n, M)] = A[IDX2C(m, n, M)] - B[IDX2C(m, n, M)];
        }
    }
}

//------------------------------ determinant ---------------------------------
__device__
Precision determinant(  int         M,
                        int         N,
                        Precision*  A)
{
    Precision determinant = 0.;

    if (M == 2 && N == 2)
    {
        determinant =   A[IDX2C(0, 0, M)] * A[IDX2C(1, 1, M)] -
                        A[IDX2C(1, 0, M)] * A[IDX2C(0, 1, M)];
    }
    else if (M == 3 && N == 3)
    {
        determinant =   A[IDX2C(0, 0, M)] * A[IDX2C(1, 1, M)] * A[IDX2C(2, 2, M)] +
                        A[IDX2C(0, 1, M)] * A[IDX2C(1, 2, M)] * A[IDX2C(2, 0, M)] +
                        A[IDX2C(0, 2, M)] * A[IDX2C(2, 1, M)] * A[IDX2C(1, 0, M)] -
                        (   A[IDX2C(2, 0, M)] * A[IDX2C(1, 1, M)] * A[IDX2C(0, 2, M)] +
                            A[IDX2C(2, 1, M)] * A[IDX2C(1, 2, M)] * A[IDX2C(0, 0, M)] +
                            A[IDX2C(2, 2, M)] * A[IDX2C(0, 1, M)] * A[IDX2C(1, 0, M)]);
    }

    return determinant;
}
...
```

- *CUDA_NumericalIntegration.cuh*: Este arquivo possui os pontos de Gauss necessários para a integração numérica no método dos elementos finitos.

```
...
//------------------------------ initializeIntegrationRules -------------------
__device__
void getIntegrationRules(   IntegrationElementType  integrationElementType,
                            size_t                  iIntegrationOrder,
                            Precision*              rules)
{
    if (integrationElementType == TRIANGLE)
    {
        if (iIntegrationOrder == 1)
        {
            rules[IDX2C(0, 0, 12)] = 1.000000000000000;
            rules[IDX2C(0, 1, 12)] = 0.333333333333333;
```

```
            rules[IDX2C(0, 2, 12)] = 0.333333333333333;
            rules[IDX2C(0, 3, 12)] = 0.333333333333333;
        }
        else if (iIntegrationOrder == 3)
        {
            rules[IDX2C(0, 0, 12)] = 0.333333333333333;
            rules[IDX2C(0, 1, 12)] = 0.666666666666667;
            rules[IDX2C(0, 2, 12)] = 0.166666666666667;
            rules[IDX2C(0, 3, 12)] = 0.000000000000000;

            rules[IDX2C(1, 0, 12)] = 0.333333333333333;
            rules[IDX2C(1, 1, 12)] = 0.166666666666667;
            rules[IDX2C(1, 2, 12)] = 0.666666666666667;
            rules[IDX2C(1, 3, 12)] = 0.000000000000000;

            rules[IDX2C(2, 0, 12)] = 0.333333333333333;
            rules[IDX2C(2, 1, 12)] = 0.166666666666667;
            rules[IDX2C(2, 2, 12)] = 0.166666666666667;
            rules[IDX2C(2, 3, 12)] = 0.000000000000000;
        }
...
```

- *CUDA_MeshCFD.cuh*: Este arquivo é o principal do pacote *FEM_CUDA*. Ele contém o cálculo das funções de forma e suas derivadas, o *assembly* das matrizes locais, a integração numérica dos elementos finitos, entre outras.

```
...
//------------------------------ do_calculate_k_g_m_e -------------------------
__device__
void do_calculate_k_g_m_e(  MeshCFDType meshCFDType,
                            MatrixType  matrixType,
                            size_t      iNumberDimensions,
                            size_t      iNumberNodes_u,
                            size_t      iNumberNodes_p,
                            Precision*  nodesCoordinates,
                            Precision   RHO,
                            Precision   MI,
                            Precision   d_XSI,
                            Precision   d_ETA,
                            Precision   d_ZETA,
                            Precision*  Nu,
                            Precision*  Np,
                            Precision*  Nu_DOF,
                            Precision*  Nu_DOF_T,
                            Precision*  Nu_DOF_T_Nu_DOF,
                            Precision*  NuDerivatives,
                            Precision*  div_Nu,
                            Precision*  div_Nu_T,
                            Precision*  div_Nu_T_Np,
                            Precision*  Ju,
                            Precision*  JuInverse,
                            Precision*  NuDerivativesTransformed,
                            Precision*  Bu,
                            Precision*  Bu_T,
                            Precision*  C,
                            Precision*  Bu_T_C,
                            Precision*  Bu_T_C_Bu,
                            Precision   JuDeterminant,
                            Precision*  k_e_int,
                            Precision*  g_e_int,
                            Precision*  m_e_int)
{
    calculate_Nu(meshCFDType, d_XSI, d_ETA, d_ZETA, Nu);
    calculate_Np(meshCFDType, d_XSI, d_ETA, d_ZETA, Np);
    calculate_NuDerivatives(meshCFDType, d_XSI, d_ETA, d_ZETA, NuDerivatives);
    calculate_J(iNumberDimensions, iNumberNodes_u, nodesCoordinates, NuDerivatives, Ju);
    JuDeterminant = determinant(iNumberDimensions, iNumberDimensions, Ju);
    inverse(iNumberDimensions, JuDeterminant, Ju, JuInverse);
    calculate_NDerivativesTransformed(  iNumberDimensions,  iNumberNodes_u,
                                        Ju,                 JuInverse,
                                        NuDerivatives,      NuDerivativesTransformed);
```

```
46
47      if (matrixType == MT_k_e)
48      {
49          do_calculate_k_e(   iNumberDimensions,          iNumberNodes_u,
50                              RHO,                        MI,
51                              NuDerivativesTransformed,   Bu,
52                              Bu_T,                       C,
53                              Bu_T_C,                     Bu_T_C_Bu,
54                              JuDeterminant,              k_e_int);
55      }
56      else if (matrixType == MT_g_e)
57      {
58          do_calculate_g_e(   iNumberDimensions,          iNumberNodes_u,
59                              iNumberNodes_p,             Np,
60                              NuDerivatives,              Ju,
61                              NuDerivativesTransformed,   div_Nu,
62                              div_Nu_T,                   div_Nu_T_Np,
63                              JuDeterminant,              g_e_int);
64      }
65      else if (matrixType == MT_m_e)
66      {
67          do_calculate_m_e(   iNumberDimensions,  iNumberNodes_u,
68                              RHO,                Nu,
69                              Nu_DOF,             Nu_DOF_T,
70                              Nu_DOF_T_Nu_DOF,    JuDeterminant,
71                              m_e_int);
72      }
73  }
74  ...
```

- *CUDA_Mesh.cuh*: Este arquivo contém o cálculo do Jacobiano da transformação quando se utiliza elementos finitos isoparamétricos, entre outras funções. Ele serve de base para outros elementos finitos, como por exemplo estruturais, facilitando assim a extensibilidade do *software*.

```
1   ...
2   //----------------------------- calculate_J ---------------------------------
3   __device__
4   void  calculate_J(  size_t      iNumberDimensions,
5                       size_t      iNumberNodes,
6                       Precision*  nodesCoordinates,
7                       Precision*  NDerivatives,
8                       Precision*  J)
9   {
10      multiply(   iNumberDimensions,  iNumberDimensions,
11                  iNumberNodes,       NDerivatives,
12                  nodesCoordinates,   J);
13  }
14  ...
```

- *CUDA_Main.cu*: Este arquivo é o principal do pacote *FEM_CUDA* no que se refere a paralelização utilizando CUDA. É neste arquivo que são definidos o tamanho do bloco de *threads* que serão executados em paralelo, as estruturas de dados nodais, o armazenamento das informações que são enviadas da CPU para a memória global da GPU e posteriormente para a memória compartilhada para maior desempenho ao acesso dos dados nodais pelas *threads*, a contribuição da matriz local para a matriz global, entre outras.

```
...
//!< Tamanho do bloco de threads a serem executados em paralelo
//!< Está relacionado com o tamanho da variável shared utilizada
#define BLOCK_SIZE 128
...
```

```
...
//! Estrutura de dados do vetor de dados nodais
struct NodalData
{
    Precision       x;              //!< Coordenada x do nó do elemento finito
    Precision       y;              //!< Coordenada y do nó do elemento finito

    //!< GL Global da velocidade na direção x do nó do elemento finito
    unsigned int    DOFGlobal_u;
    //!< GL Local da velocidade na direção y do nó do elemento finito
    unsigned int    DOFLocal_u;
    //!< Velocidade na direção x do nó do elemento finito (C e C_t)
    Precision       u;

    //!< GL Global da velocidade na direção y do nó do elemento finito
    unsigned int    DOFGlobal_v;
    //!< GL Local da velocidade na direção y do nó do elemento finito
    unsigned int    DOFLocal_v;
    //!< Velocidade na direção y do nó do elemento finito (C e C_t)
    Precision       v;

    //!< GL Global da pressão do nó do elemento finito
    unsigned int    DOFGlobal_p;
    //!< GL Local da pressão do nó do elemento finito
    unsigned int    DOFLocal_p;
};
...
```

```
...
//----------------------------- do_assembly_K_G_M ----------------------------
//! Realiza o assembly das matrizes K, G e M
/*!
 * \param meshCFDType Tipo de elemento finito: P2P1 ou Q2Q1
 * \param inumberDimensions Número de dimensões do problema analisado
 * \param iNumberNodes_u Número de nós para velocidade
 * \param iNumberNodes_p Número de nós para pressão
 * \param N Número de elementos finitos do problema
 * \param RHO Massa específica do fluido
 * \param MI Coeficiente de viscosidade dinâmica do fluido
 */
__global__
void do_assembly_K_G_M( MeshCFDType meshCFDType,
                        size_t      iNumberDimensions,
                        size_t      iNumberNodes_u,
                        size_t      iNumberNodes_p,
                        size_t      N,
                        Precision   RHO,
                        Precision   MI)
{
    ////////////////////////////////////////////////////////////////////////
    //     Variável do tipo shared a ser compartilhada com as threads do     //
    //     bloco                                                             //
    ////////////////////////////////////////////////////////////////////////
    extern __shared__ NodalData s_nodalData[];

    ////////////////////////////////////////////////////////////////////////
    //     Armazena os valores dos dados nodais global na variável           //
    //     compartilhada                                                     //
    ////////////////////////////////////////////////////////////////////////
    size_t iContNodalData = 0;

    for (   size_t  i       = blockDim.x * blockIdx.x * iNumberNodes_u,
    iLen    = blockDim.x * blockIdx.x * iNumberNodes_u + blockDim.x * iNumberNodes_u;
    i < iLen;
    ++i)
    {
        if (i < N * iNumberNodes_u)
        {
```

```
41            s_nodalData[iContNodalData].x           = d_nodalData[i].x;
42            s_nodalData[iContNodalData].y           = d_nodalData[i].y;
43
44            s_nodalData[iContNodalData].DOFGlobal_u = d_nodalData[i].DOFGlobal_u;
45            s_nodalData[iContNodalData].DOFLocal_u  = d_nodalData[i].DOFLocal_u;
46            s_nodalData[iContNodalData].u           = d_nodalData[i].u;
47
48            s_nodalData[iContNodalData].DOFGlobal_v = d_nodalData[i].DOFGlobal_v;
49            s_nodalData[iContNodalData].DOFLocal_v  = d_nodalData[i].DOFLocal_v;
50            s_nodalData[iContNodalData].v           = d_nodalData[i].v;
51
52            s_nodalData[iContNodalData].DOFGlobal_p = d_nodalData[i].DOFGlobal_p;
53            s_nodalData[iContNodalData].DOFLocal_p  = d_nodalData[i].DOFLocal_p;
54        }
55
56        ++iContNodalData;
57    }
58
59    ////////////////////////////////////////////////////////////////////////
60    //     Sincroniza as threads do bloco para início do assembly          //
61    ////////////////////////////////////////////////////////////////////////
62    __syncthreads();
63 ...
```

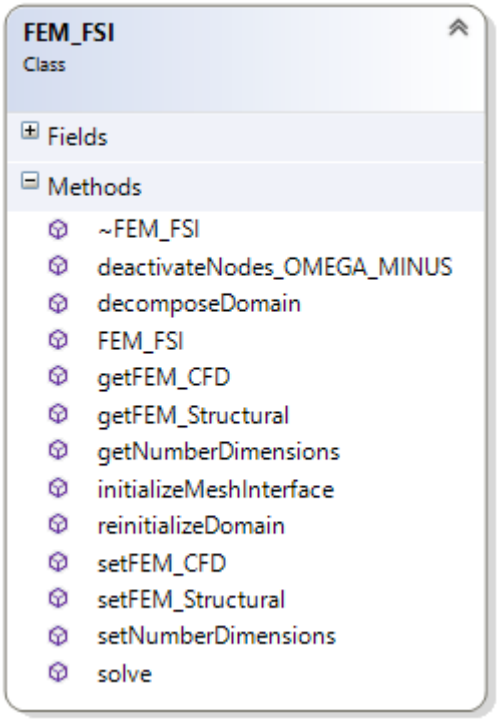
5.1.1.5 Pacote *FEM_FSI*

Este pacote possui as classes responsáveis pela análise pelo método dos elementos finitos de uma malha de fluido utilizando o método das Fronteiras Imersas como apresentado em capítulos anteriores. Elas utilizam algumas classes dos pacotes ou módulos *FEM_Core*, *FEM_Structural* e *FEM_CFD*.

As seguintes classes pertencem ao pacote *FEM_FSI*:

- *FEM_FSI*: Esta classe é responsável por realizar o cálculo pelo método dos elementos finitos de um fluido utilizando o método das Fronteiras Imersas como apresentado em capítulos anteriores (Fig. 5.41).

Figura 5.41 – Classe FEM_FSI.



Fonte: O autor

5.1.2 Diagramas de sequência

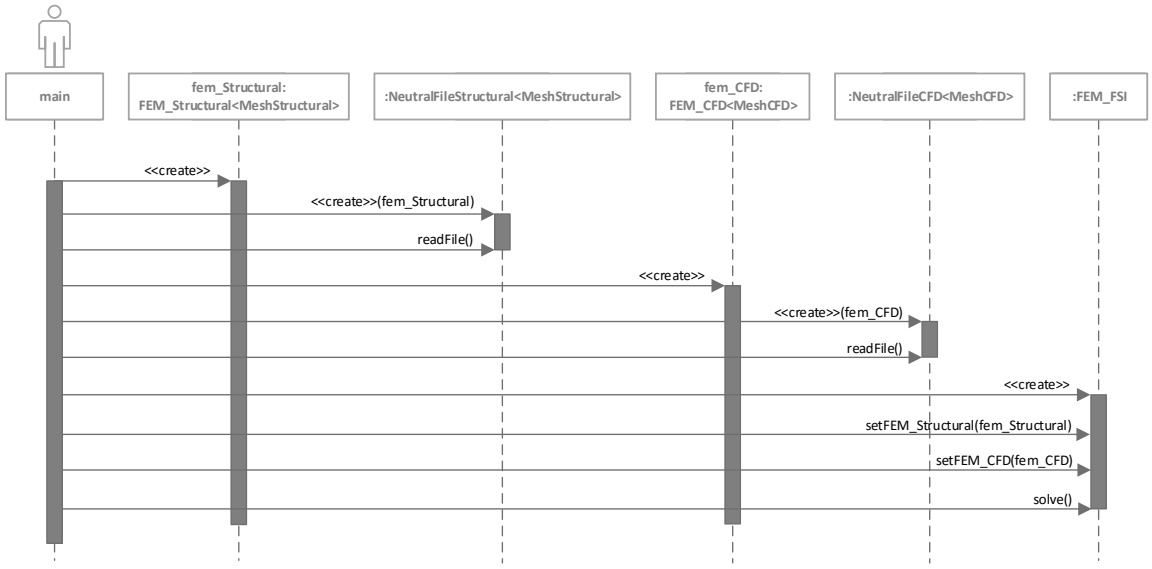
Nesta seção são apresentados os diagramas de sequência do *software* desenvolvido para que o leitor possa compreender a utilização das classes apresentadas nas seções anteriores e de que forma estas classes interagem entre si para realizar o cálculo pelo método das Fronteiras Imersas.

O diagrama da Fig. 5.42 é executado da seguinte forma:

Tabela 5.1 – Algoritmo da sequência principal.

Algoritmo Sequência principal	
1:	Cria-se uma instância da classe *FEM_Structural*;
2:	Cria-se uma instância da classe *NeutralFileStructural*;
3:	Lê-se o arquivo de configurações da estrutura;
4:	Cria-se uma instância da classe *FEM_CFD*;
5:	Cria-se uma instância da classe *NeutralFileCFD*;
6:	Lê-se o arquivo de configurações do fluido;
7:	Cria-se uma instância da classe *FEM_FSI*;
8:	Envia as instâncias das classes criadas nos passos anteriores para a instância da classe *FEM_FSI* de forma que os métodos dessas classes estejam disponíveis e acessíveis;
9:	Resolve-se o problema pelo método das Fronteiras Imersas.

Figura 5.42 – Diagrama de sequência principal.



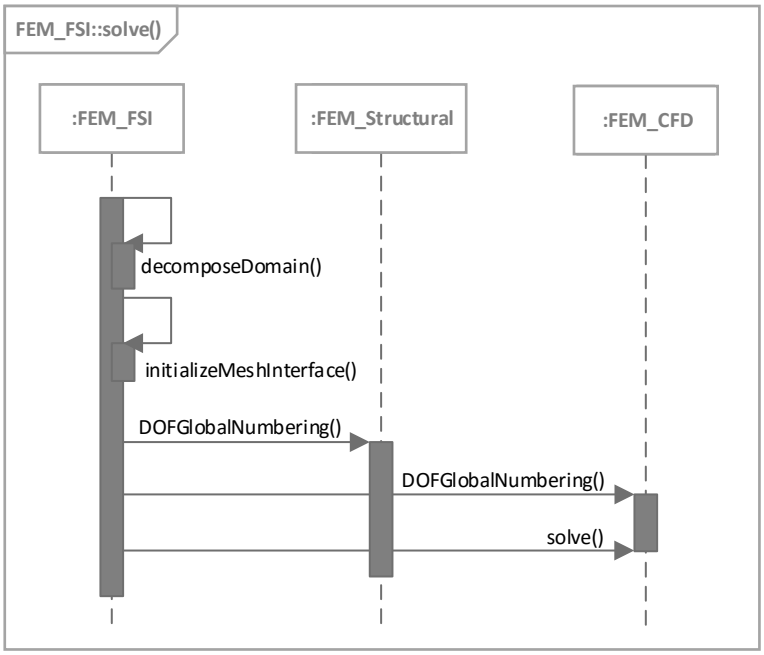
Fonte: O autor

O diagrama da Fig. 5.43 é executado da seguinte forma:

Tabela 5.2 – Algoritmo do método *solve* da classe *FEM_FSI*.

Algoritmo Método *solve* da classe *FEM_FSI*	
1:	Decompõe-se o domínio em Ω^- e Ω^+ como apresentado em capítulos anteriores;
2:	Inicializa-se a interface como resultado da intersecção entre a malha de elementos finitos da estrutura e a malha de fluido;
3:	Realiza-se a numeração dos graus de liberdade globais da estrutura;
4:	Realiza-se a numeração dos graus de liberdade globais do fluido;
5:	Resolve-se o problema pelo método dos elementos finitos.

Figura 5.43 – Diagrama de sequência do método *sove* da classe *FEM_FSI*.



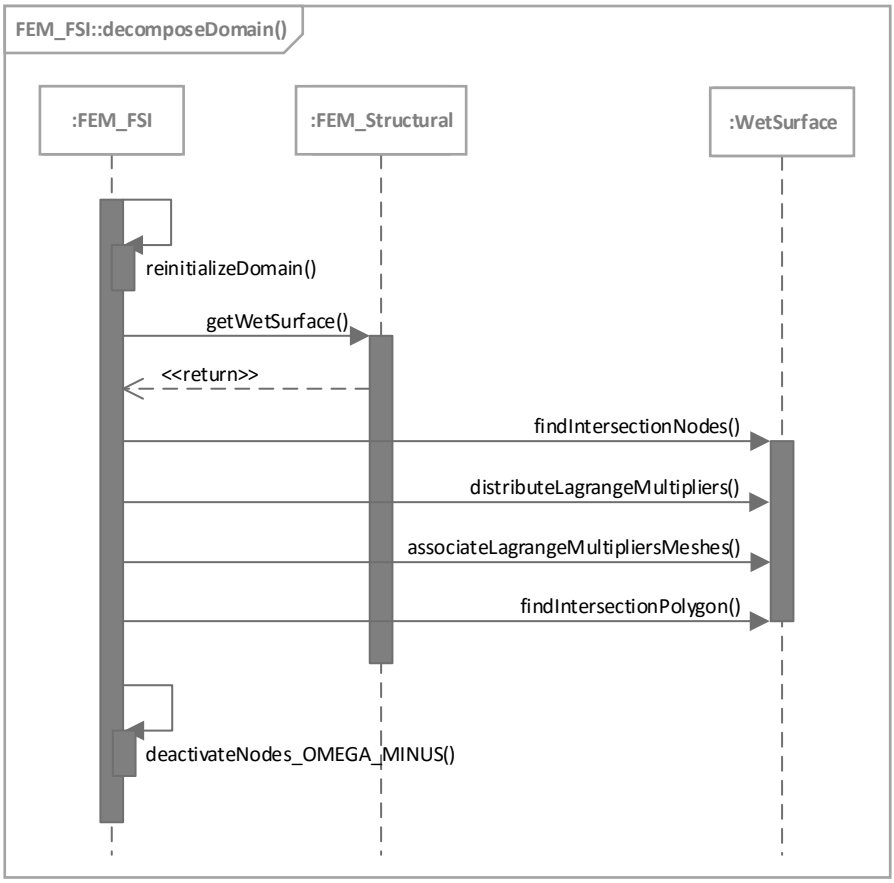
Fonte: O autor

O diagrama da Fig. 5.44 é executado da seguinte forma:

Tabela 5.3 – Algoritmo do método *decomposeDomain* da classe *FEM_FSI*.

Algoritmo Método *decomposeDomain* da classe *FEM_FSI*	
1:	Reinicializa-se o domínio tornando toda a malha de fluido como Ω^+ para que posteriormente possa ser dividido em Ω^- e Ω^+;
2:	Solicita-se à instância da classe *FEM_Structural* a superfície molhada, definida no arquivo de configuração da estrutura;
3:	Encontra-se os nós de intersecção entre a malha da estrutura e a malha de fluido;
4:	Distribui-se os multiplicadores de Lagrange como apresentado em capítulos anteriores;
5:	Associa-se os multiplicadores de Lagrange do passo anterior a cada elemento finito interseccionado;
6:	Encontra-se o polígono de intersecção para utilização na integração no polígono como apresentado em capítulos anteriores;
7:	Desativa-se os nós que esteja sob a malha da estrutura. Desta forma estes nós não contribuirão para a matriz global.

Figura 5.44 – Diagrama de sequência do método *decomposeDomain* da classe *FEM_FSI*.



Fonte: O autor

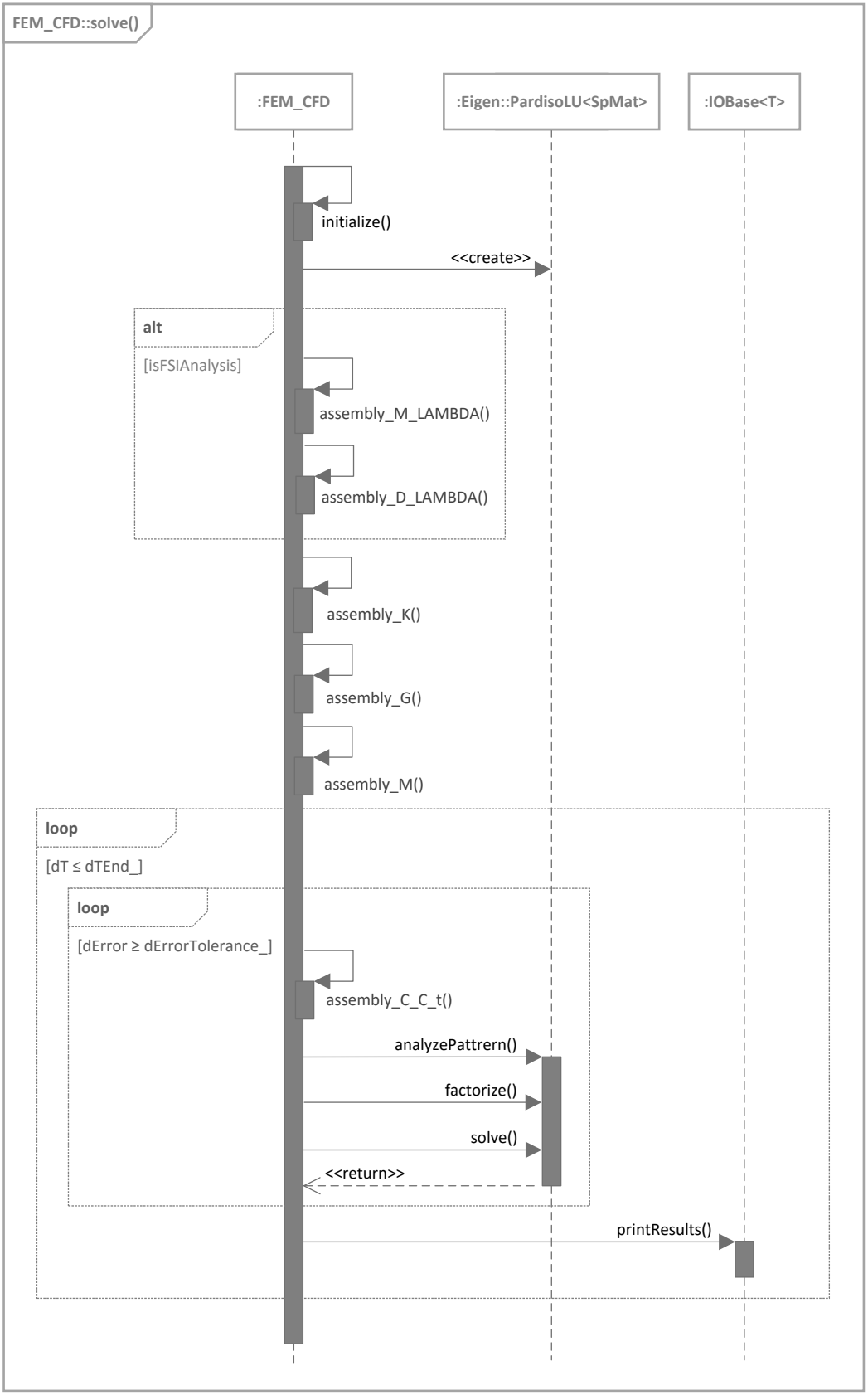
O diagrama da Fig. 5.44 é executado da seguinte forma:

Tabela 5.4 – Algoritmo do método *solve* da classe *FEM_CFD*.

Algoritmo Método *solve* da classe *FEM_CFD*	
1:	Inicializa-se as matrizes $\mathbf{K}$, $\mathbf{G}$, $\mathbf{M}$, $\mathbf{C}$, $\mathbf{C_t}$, $\mathbf{M_\lambda}$ e $\mathbf{D_\lambda}$ como apresentado em capítulos anteriores;
2:	Cria-se uma instância da classe Pardiso para solução pelo método direto da matriz principal;
3:	Como a classe *FEM_CFD* é utilizada tanto para resolução de problemas de dinâmica dos fluidos como para interação fluido-estrutura utilizando o método das Fronteiras Imersas, é necessário utilizar uma *flag*[6] para decidir em tempo de execução se será um problema de dinâmica de fluidos ou interação fluido-estrutura. Esta tomada de decisão (*flag* $isFSIAnalysis$)pode ser observada no diagrama de sequências da Fig. 5.45;
3.1:	Caso seja um problema de interação fluido-estrutura é necessário realizar o *assembly*[7] das matrizes $\mathbf{M_\lambda}$ e $\mathbf{D_\lambda}$;
4:	Realiza-se o *assembly* das matrizes $\mathbf{K}$, $\mathbf{G}$ e $\mathbf{M}$;
5:	O problema a ser resolvido tanto pode ser um problema estático, quanto um problema dinâmico. Utiliza-se a variável dT como controle;
5.1:	Enquanto a variável dT for menor ou igual a variável $dTEnd\_$, ou seja, se for um problema estático, estas duas variáveis terão o mesmo valor de forma que este *loop* seja executado pelo menos uma vez. Se for um problema dinâmico, este *loop* será executado tantas vezes forem necessárias até que a variável $dTEnd\_$ seja maior que a variável de controle dT. Faça:
5.1.1:	Enquanto o erro analisado[8] for menor ou igual a tolerância definida para o problema, faça:
5.1.1.1:	Realiza-se o *assembly* das matrizes $\mathbf{C}$ e $\mathbf{C_t}$;
5.1.1.2:	Analisa-se, fatora-se e resolve-se a matriz principal, retornando-se o Δv (passo de velocidade e pressão a ser adicionado às matrizes de velocidade e pressão) da iteração.
5.2:	Imprima-se os resultados da iteração.

[6]Uma *flag* é uma variável utilizada em um código-fonte como tomada de decisão em tempo de execução.
[7]O *assembly* é a contribuição local de cada nó para a matriz global.
[8]Este erro é controlado como convergência do problema.

Figura 5.45 – Diagrama de sequência do método *solve* da classe *FEM_CFD*.



Fonte: O autor

6 RESULTADOS E ANÁLISES

Este capítulo apresenta os resultados de simulações numéricas realizadas e suas análises. Para isso, foi desenvolvido um código computacional implementando a teoria apresentada nos capítulos anteriores, bem como o método de *assembly* executado na GPU, descrito no Capítulo 2.

As simulações numéricas foram realizadas em duas fases, uma sequencial e uma paralelizada na GPU. Com isso pôde-se testar e validar o software de forma a abarcar tanto computadores sem o poder da GPU, quanto os computadores que possuam placas de vídeo com processamento paralelo utilizando a tecnologia CUDA.

A configuração utilizada para a análise de desempenho entre a implementação computacional sequencial e paralela é composta por um processador de vídeo NVIDIA GeForce GTX 750 (versão do *driver*: 364.72) instalada no barramento PCI *Express* 3.0 de um computador Intel Core i5 3.20 GHz com 16 GB de memória RAM e arquitetura 64 bits rodando em um sistema operacional Windows 10 *Professional*. A placa de vídeo possui 4 multiprocessadores, 512 *cores* CUDA, 1024 MB de memória DDR5 com um *bandwidth* de 80,16 GB/s. A versão CUDA utilizada é a 5.0.

Para todas as simulações realizadas neste capítulo foi feito primeiramente uma análise de sensibilidade do tempo total para execução do *assembly* das matrizes globais $\mathbf{K}$, $\mathbf{G}$, $\mathbf{M}$, $\mathbf{C}$ e $\mathbf{C_t}$, para resolução do sistema de equações e da simulação para diferentes malhas de elementos finitos. Desta forma é possível observar qual a quantidade de elementos finitos que fornece o melhor custo-benefício na utilização da GPU. Os tempos apresentados para o código paralelo incluem as chamadas aos *kernels* para *assembly* das matrizes globais, mas não incluem qualquer tempo gasto para a transferência de dados CPU-GPU.

Como apresentado no Cap. 2 é necessário encontrar a melhor combinação de blocos e *threads* que geram o menor custo computacional e o maior desempenho. Exaustivos experimentos foram realizados a fim de encontrar esta relação e a melhor configuração encontrada foi:

$$\#\text{Blocos} = \frac{(N + \#Threads - 1)}{\#Threads} \tag{6.1}$$

onde

- #Blocos: Número de blocos que serão executados em paralelo;
- N: Número de elementos finitos do problema;
- $\#Threads$: Número de *threads* que serão executadas por bloco – Este número está relacionado com o número de *bytes* que serão armazenados na memória compartilhada da GPU.

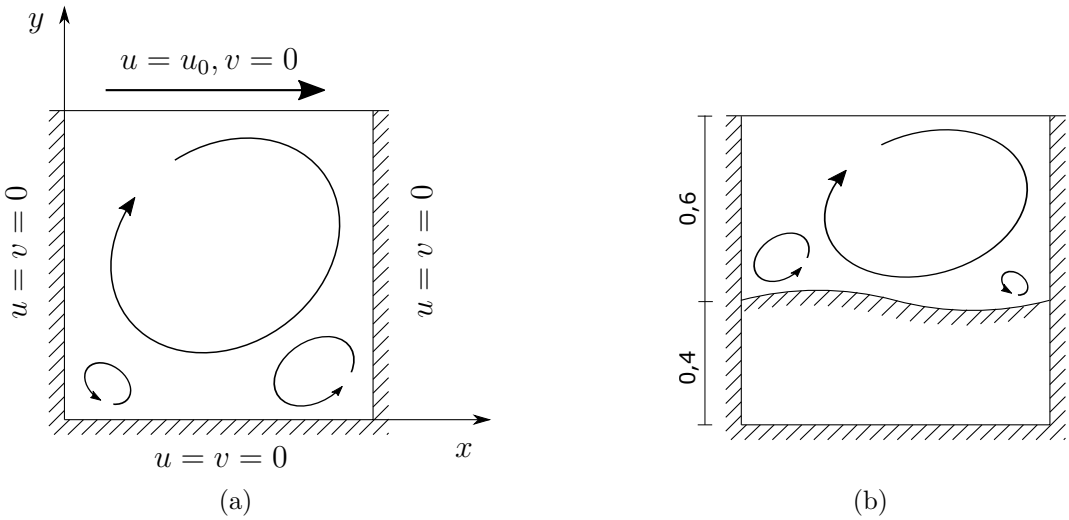
6.1 Escoamento em uma cavidade

Um problema clássico da literatura e um caso comum de validação para códigos CFD é o escoamento em uma cavidade. Este problema modela uma recirculação ideal com aplicações em modelagens ambientais, geofísicas e industriais. A razão para sua popularidade como caso de validação é devido a descrição simples do domínio do problema, mas que apesar de sua simplicidade apresenta comportamentos de natureza complexa mas de fácil determinação, como por exemplo, a contagem de vórtices.

Em um problema bidimensional o domínio é modelado como um quadrado de lados unitários onde uma das paredes (parede superior) possui velocidade imposta constante diferente de zero. Em Erturk (2009) é apresentada uma discussão extensa sobre este problema. Uma boa fonte de resultados para diferentes números de Reynolds pode ser encontrado em Gomes (2013) e Ghia et al. (1982).

Um outro exemplo interessante pode ser visto em Zhang et al. (2012) onde uma das paredes da cavidade (inferior) é hiperelástica. No presente trabalho utilizou-se uma parede rígida, com um formato ondular, similar ao apresentado neste artigo (após sua deformação) de forma a comparar os resultados de uma malha conforme com uma malha não-conforme utilizando o método das Fronteiras Imersas. A Fig. 6.1 ilustra estas geometrias.

Figura 6.1 – Descrição dos modelos (a) bidimensional e (b) tridimensional



Fonte: O autor

O número de Reynolds para os dois problemas é dado por

$$Re = \frac{\rho u}{\mu} \tag{6.2}$$

onde:

- ρ: Densidade do fluido;

- u: Velocidade imposta na parede superior da cavidade correspondente ao próprio número de Reynolds do problema;
- μ: Viscosidade dinâmica do fluido.

A densidade e viscosidade dinâmica do fluido são considerados iguais à unidade.

6.1.1 Cavidade com paredes retas

6.1.1.1 Análise de sensibilidade

Para a análise de sensibilidade desta simulação foram utilizadas malhas com número crescente de elementos finitos do tipo P2P1 e $Re = 400$[1]. As Tab. 6.1, 6.2, 6.3, 6.4, 6.5, 6.6 e 6.7 apresentam o tempo total para o *assembly* das matrizes globais, solução do sistema de equações e simulação para uma malha com 98, 510, 1102, 5678, 10210, 22804 e 40920 elementos tanto para o processamento sequencial quanto para o paralelizado na GPU.

Tabela 6.1 – Tempo total para o *assembly* das matrizes globais, solução do sistema de equações e simulação para uma malha com 98 elementos.

#Elementos = 98				
Sequencial				
#Iteração	*Assembly* **K** e **G** (s)	*Assembly* **C** e $\mathbf{C_t}$ (s)	Solução (s)	Total (s)
–	0.015	–	–	0.015
1	–	0.015	0.100	0.115
2	–	0.015	0.015	0.030
3	–	0.038	0.000	0.038
4	–	0.015	0.017	0.032
5	–	0.031	0.000	0.031
6	–	0.015	0.000	0.015
7	–	0.022	0.000	0.022
8	–	0.015	0.020	0.035
	0.015	0.166	0.152	0.333
Paralelo				
–	0.015	–	–	0.015
1	–	0.021	0.100	0.121
2	–	0.000	0.015	0.015
3	–	0.000	0.015	0.015
4	–	0.015	0.000	0.015
5	–	0.000	0.000	0.000
6	–	0.000	0.000	0.000
7	–	0.006	0.015	0.021
8	–	0.015	0.000	0.015
	0.015	0.057	0.145	0.217

[1] Com este número de Reynolds o problema é resolvido em 8 iterações aumentando o tempo total da simulação, facilitando assim a comparação entre os processamentos sequencial e paralelo.

Tabela 6.2 – Tempo total para o *assembly* das matrizes globais, solução do sistema de equações e simulação para uma malha com 510 elementos.

#Elementos = 510				
Sequencial				
#Iteração	*Assembly* **K** e **G** (s)	*Assembly* **C** e $\mathbf{C_t}$ (s)	Solução (s)	Total (s)
–	0.077	–	–	0.077
1	–	0.100	0.115	0.215
2	–	0.115	0.031	0.146
3	–	0.115	0.020	0.135
4	–	0.115	0.021	0.136
5	–	0.116	0.033	0.149
6	–	0.100	0.037	0.137
7	–	0.115	0.031	0.146
8	–	0.116	0.015	0.131
	0.077	0.892	0.303	1.272
Paralelo				
–	0.029	–	–	0.029
1	–	0.025	0.130	0.155
2	–	0.000	0.015	0.015
3	–	0.015	0.015	0.030
4	–	0.015	0.038	0.053
5	–	0.000	0.037	0.037
6	–	0.012	0.038	0.050
7	–	0.012	0.038	0.050
8	–	0.012	0.034	0.046
	0.029	0.091	0.345	0.465

Tabela 6.3 – Tempo total para o *assembly* das matrizes globais, solução do sistema de equações e simulação para uma malha com 1102 elementos.

#Elementos = 1102				
Sequencial				
#Iteração	*Assembly* **K** e **G** (s)	*Assembly* **C** e $\mathbf{C_t}$ (s)	Solução (s)	Total (s)
–	0.143	–	–	0.143
1	–	0.232	0.138	0.370
2	–	0.247	0.062	0.309
3	–	0.234	0.068	0.302
4	–	0.254	0.062	0.316
5	–	0.238	0.053	0.291
6	–	0.247	0.062	0.309
7	–	0.235	0.053	0.288
8	–	0.247	0.062	0.309
	0.143	1.934	0.56	2.637
Paralelo				
–	0.031	–	–	0.031
1	–	0.031	0.147	0.178
2	–	0.022	0.100	0.122
3	–	0.031	0.080	0.111
4	–	0.024	0.120	0.144
5	–	0.026	0.113	0.139
6	–	0.026	0.075	0.101
7	–	0.031	0.049	0.080
8	–	0.027	0.041	0.068
	0.031	0.218	0.725	0.974

Tabela 6.4 – Tempo total para o *assembly* das matrizes globais, solução do sistema de equações e simulação para uma malha com 5678 elementos.

#Elementos = 5678				
Sequencial				
#Iteração	*Assembly* **K** e **G** (s)	*Assembly* **C** e $\mathbf{C_t}$ (s)	Solução (s)	Total (s)
–	0.821	–	–	0.821
1	–	1.249	0.450	1.699
2	–	1.252	0.370	1.622
3	–	1.267	0.337	1.604
4	–	1.262	0.369	1.631
5	–	1.258	0.331	1.589
6	–	1.262	0.347	1.609
7	–	1.242	0.353	1.595
8	–	1.256	0.351	1.607
	0.821	10.048	2.908	13.777
Paralelo				
–	0.169	–	–	0.169
1	–	0.115	0.398	0.513
2	–	0.115	0.251	0.366
3	–	0.100	0.248	0.348
4	–	0.116	0.245	0.361
5	–	0.116	0.257	0.373
6	–	0.100	0.255	0.355
7	–	0.115	0.239	0.354
8	–	0.116	0.244	0.360
	0.169	0.893	2.137	3.199

Tabela 6.5 – Tempo total para o *assembly* das matrizes globais, solução do sistema de equações e simulação para uma malha com 10210 elementos.

#Elementos = 10210				
Sequencial				
#Iteração	*Assembly* **K** e **G** (s)	*Assembly* **C** e $\mathbf{C_t}$ (s)	Solução (s)	Total (s)
–	1.397	–	–	1.397
1	–	2.238	0.781	3.019
2	–	2.247	0.719	2.966
3	–	2.251	0.717	2.968
4	–	2.260	0.720	2.980
5	–	2.257	0.717	2.974
6	–	2.233	0.731	2.964
7	–	2.244	0.717	2.961
8	–	2.255	0.718	2.973
	1.397	17.985	5.820	25.202
Paralelo				
–	0.253	–	–	0.253
1	–	0.216	0.569	0.785
2	–	0.210	0.451	0.661
3	–	0.207	0.466	0.673
4	–	0.211	0.469	0.680
5	–	0.214	0.468	0.682
6	–	0.213	0.469	0.682
7	–	0.214	0.469	0.683
8	–	0.217	0.457	0.674
	0.253	1.702	3.818	5.773

Tabela 6.6 – Tempo total para o *assembly* das matrizes globais, solução do sistema de equações e simulação para uma malha com 22804 elementos.

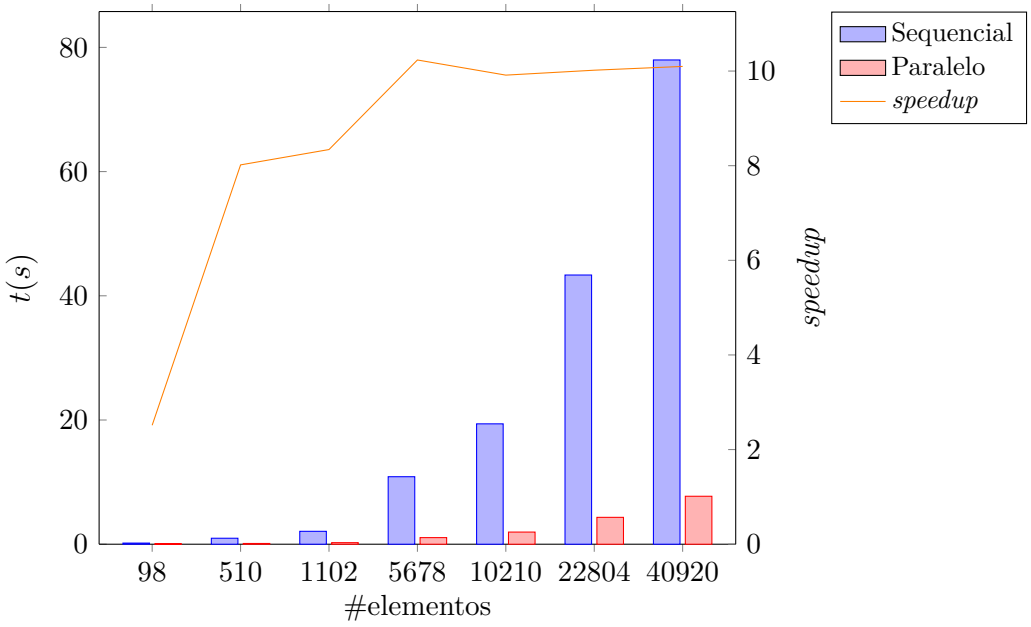
#Elementos = 22084				
Sequencial				
#Iteração	*Assembly* **K** e **G** (s)	*Assembly* **C** e $\mathbf{C_t}$ (s)	Solução (s)	Total (s)
–	3.089	–	–	3.089
1	–	4.998	1.905	6.903
2	–	5.031	1.818	6.849
3	–	5.017	1.905	6.922
4	–	5.036	1.797	6.833
5	–	5.028	1.856	6.884
6	–	5.038	1.823	6.861
7	–	5.058	1.778	6.836
8	–	5.051	1.803	6.854
	3.089	40.257	14.685	58.031
Paralelo				
–	0.57	–	–	0.570
1	–	0.485	1.197	1.682
2	–	0.470	1.132	1.602
3	–	0.471	1.107	1.578
4	–	0.469	1.111	1.580
5	–	0.463	1.110	1.573
6	–	0.463	1.092	1.555
7	–	0.472	1.169	1.641
8	–	0.464	1.101	1.565
	0.570	3.757	9.019	13.346

Tabela 6.7 – Tempo total para o *assembly* das matrizes globais, solução do sistema de equações e simulação para uma malha com 40920 elementos.

#Elementos = 40920				
Sequencial				
#Iteração	*Assembly* **K** e **G** (s)	*Assembly* **C** e $\mathbf{C_t}$ (s)	Solução (s)	Total (s)
–	5.563	–	–	5.563
1	–	9.021	3.566	12.587
2	–	9.064	3.576	12.640
3	–	9.083	3.596	12.679
4	–	9.084	3.685	12.769
5	–	9.041	3.694	12.735
6	–	9.050	3.716	12.766
7	–	9.028	3.721	12.749
8	–	9.040	3.702	12.742
	5.563	72.411	29.256	107.230
Paralelo				
–	1.031	–	–	1.031
1	–	0.844	2.340	3.184
2	–	0.861	2.200	3.061
3	–	0.828	2.199	3.027
4	–	0.817	2.242	3.059
5	–	0.829	2.202	3.031
6	–	0.832	2.274	3.106
7	–	0.841	2.338	3.179
8	–	0.839	2.265	3.104
	1.031	6.691	18.06	25.782

A Fig. 6.2 mostra o comparativo entre o processamento sequencial e paralelo para o tempo total do *assembly* das matrizes globais e o *speedup* alcançado.

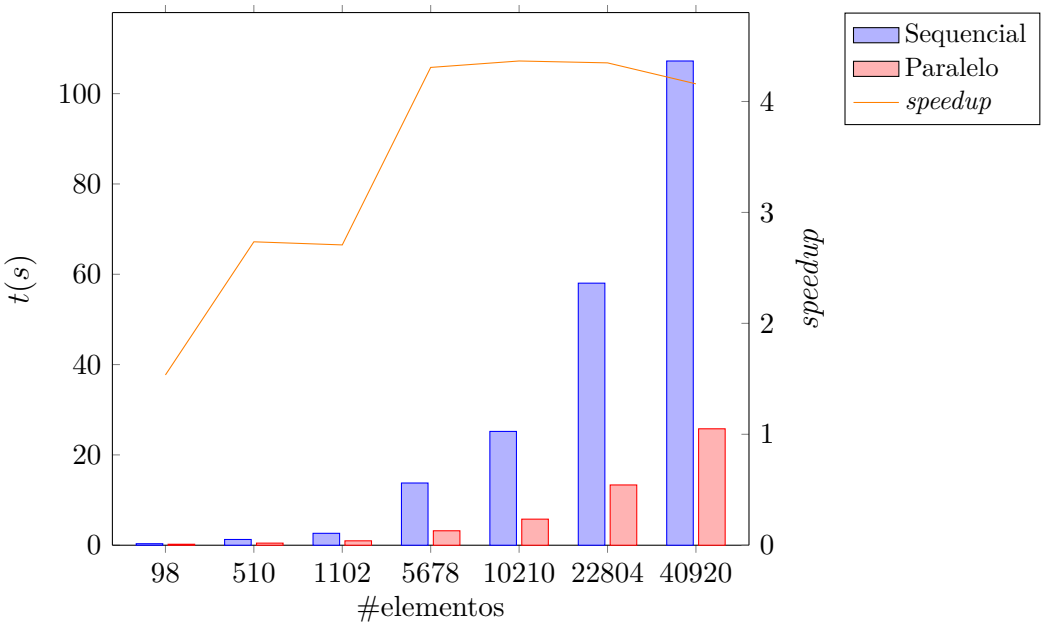
Figura 6.2 – Comparativo do tempo total para o *assembly* das matrizes globais e o *speedup* alcançado.



Fonte: O autor

A Fig. 6.3 mostra o comparativo entre o processamento sequencial e paralelo para o tempo total da simulação e o *speedup* alcançado.

Figura 6.3 – Comparativo do tempo total da simulação e o *speedup* alcançado.



Fonte: O autor

Os resultados comparativos mostram uma considerável diminuição no tempo de processamento, com um *speedup* de aproximadamente 10 para o *assembly* das matrizes

globais para a malha de elementos finitos mais refinada. A curva do *speedup*, como pode ser observado na Fig. 6.2, apresenta uma diminuição em sua inclinação e estabilização acima de 5000 elementos. Ou seja, acima deste valor, aumentando-se o número de elementos não obter-se-á um aumento significativo de *speedup*, como acontece para as malhas menos refinadas.

O tempo total para solução do problema mostra um *speedup* de aproximadamente 4.5 para a malha de elementos finitos mais refinada. Isto se deve ao fato de que a solução do sistema de equações lineares é realizada de forma sequencial utilizando o pacote PARDISO[2] (KUZMIN et al., 2013). A mesma observação anterior – aumento do número de elementos acima de 5000 elementos não obtém-se um considerável aumento de *speedup* – pode ser feita para o comparativo do tempo do total para solução do problema, inclusive, a partir de 22000 elementos, há uma diminuição do *speedup*.

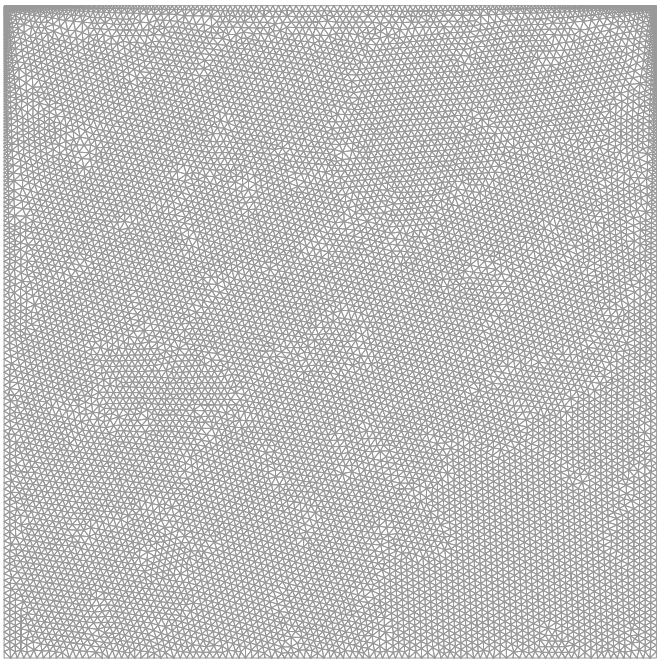
Pode-se observar que em uma malha pouco refinada não há vantagem em se utilizar o processamento paralelo ao invés do sequencial (*speedup* de aproximadamente 1.5). A justificativa para tal fenômeno está relacionado com o tempo necessário para que a GPU inicie o processo de paralelização (inicialização dos blocos de *threads*, das *threads* e outros processos de inicializações internos da placa gráfica).

6.1.1.2 Resultados

O escoamento em uma cavidade bidimensional foi simulado e validado com os dados fornecidos em Ghia et al. (1982) para um escoamento laminar com Reynolds variando entre 1 e 400. Uma malha com 24143 elementos do tipo P2P1 e 48966 nós foi utilizada e é apresentada na Fig. 6.4.

[2]Existe a possibilidade de utilizar arquitetura de memória compartilhada e distribuída para paralelizar a solução do sistema com PARDISO. No caso deste trabalho, o máximo número de *cores* disponíveis na CPU é de 4 e não foi utilizado *cluster* de CPUs. Com isso, se comparar os 4 *cores* da CPU com as 512 *threads* da GPU, pode-se considerar a solução do sistema um processo sequencial.

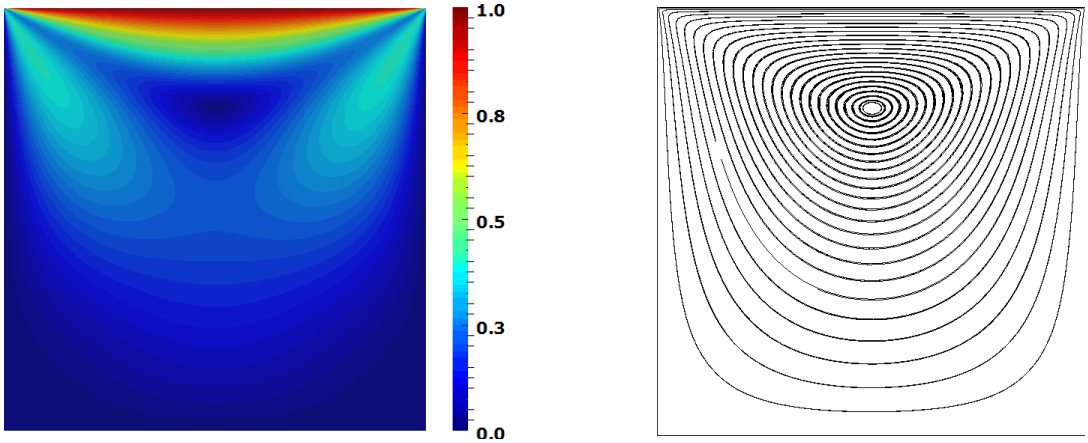
Figura 6.4 – Malha de elementos finitos com 24143 elementos finitos do tipo P2P1 e 48966 nós.



Fonte: O autor

As Fig. 6.5, 6.6, 6.7 e 6.8 apresentam os perfis de velocidade e linhas de corrente no domínio.

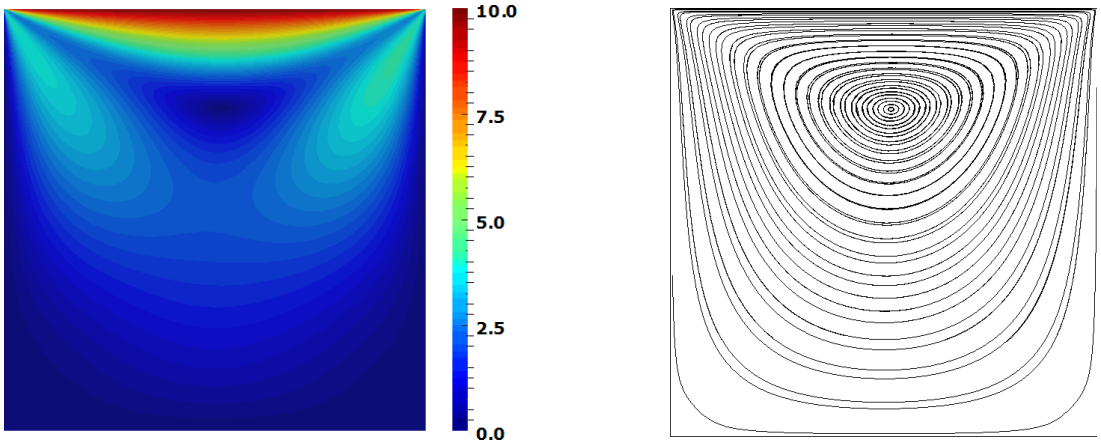
Figura 6.5 – Perfis de velocidade (a) e linhas de corrente (b) para $Re = 1$.



(a) (b)

Fonte: O autor

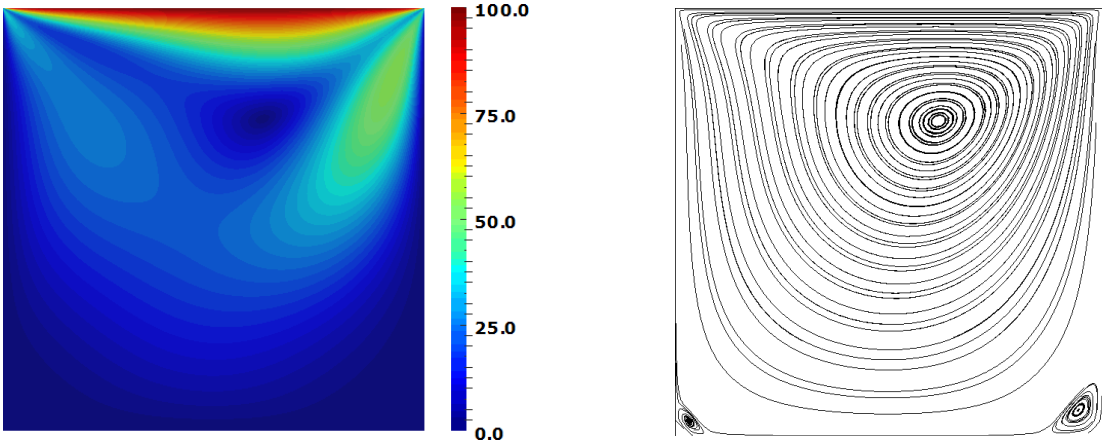
Figura 6.6 – Perfis de velocidade (a) e linhas de corrente (b) para $Re = 10$.



(a) (b)

Fonte: O autor

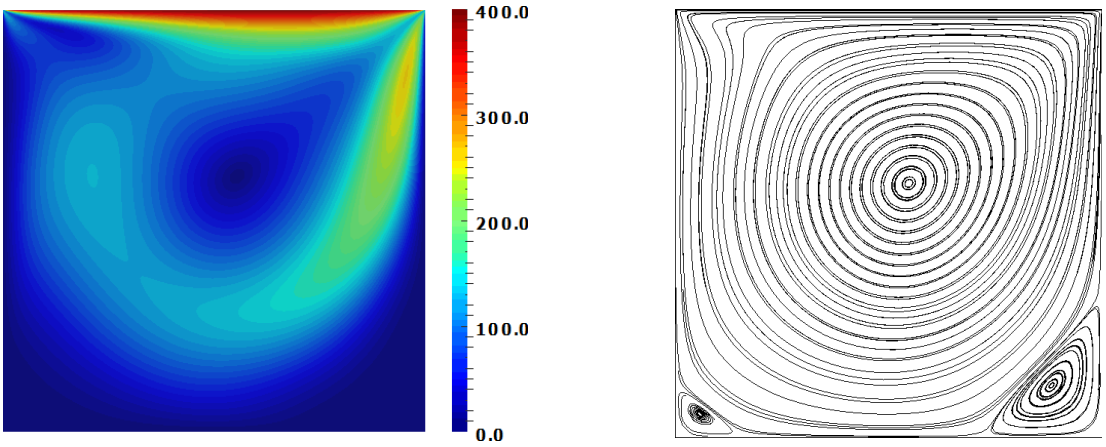
Figura 6.7 – Perfis de velocidade (a) e linhas de corrente (b) para $Re = 100$.



(a) (b)

Fonte: O autor

Figura 6.8 – Perfis de velocidade (a) e linhas de corrente (b) para $Re = 400$.



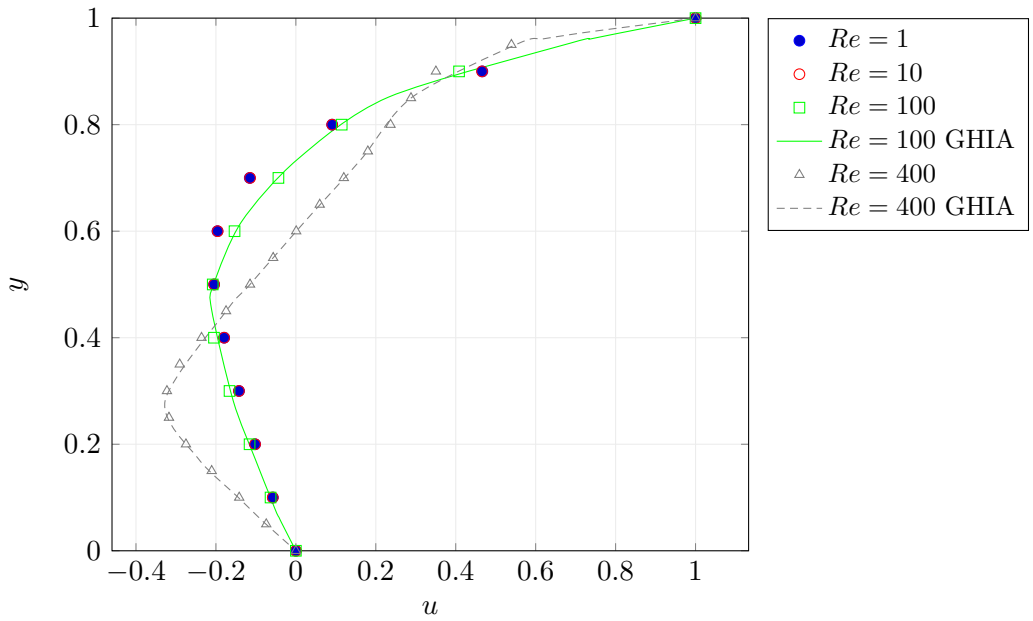
(a) (b)

Fonte: O autor

Pode-se observar que à medida em que o número de Reynolds aumenta, o vórtice principal desloca-se para a direita e há um aparecimento de vórtices secundários na parte inferior da cavidade.

A Fig. 6.9 mostra o perfil de velocidades (normalizada) pelo número de Reynolds para validação e comparação dos resultados entre este trabalho e Ghia et al. (1982) com números de Reynolds variando entre 1 e 400.

Figura 6.9 – Velocidade u ao longo da linha vertical sobre o centro geométrico da cavidade.



Fonte: O autor

6.1.1.3 Análise de desempenho

As Tab. 6.8, 6.9, 6.10, 6.11 apresentam o tempo total para o *assembly* das matrizes globais, solução do sistema de equações e simulação para uma malha com 24143 elementos do tipo P2P1 tanto para o processamento sequencial quanto para o paralelizado na GPU para números de Reynolds variando entre 1 e 400.

Tabela 6.8 – Tempo total para o *assembly* das matrizes globais, solução do sistema de equações e simulação para uma malha com 24143 elementos e $Re = 1$.

#Elementos = 24143				
Sequencial				
#Iteração	*Assembly* **K** e **G** (s)	*Assembly* **C** e $\mathbf{C_t}$ (s)	Solução (s)	Total (s)
–	2.860	–	–	2.860
1	–	4.680	2.103	6.783
2	–	4.769	2.020	6.789
3	–	4.711	2.020	6.731
	2.860	14.160	6.143	23.163
Paralelo				
–	0.611	–	–	0.611
1	–	0.516	2.090	2.606
2	–	0.487	2.012	2.499
3	–	0.498	1.992	2.490
	0.611	1.501	6.094	8.206

Tabela 6.9 – Tempo total para o *assembly* das matrizes globais, solução do sistema de equações e simulação para uma malha com 24143 elementos e $Re = 10$.

#Elementos = 24143				
Sequencial				
#Iteração	*Assembly* **K** e **G** (s)	*Assembly* **C** e $\mathbf{C_t}$ (s)	Solução (s)	Total (s)
–	2.883	–	–	2.883
1	–	4.627	2.105	6.732
2	–	4.657	2.026	6.683
3	–	4.665	2.051	6.716
4	–	4.649	2.005	6.654
	2.883	18.598	8.187	29.668
Paralelo				
–	0.601	–	–	0.601
1	–	0.516	2.154	2.670
2	–	0.498	1.982	2.480
3	–	0.487	1.995	2.482
4	–	0.495	1.976	2.471
	0.601	1.996	8.107	10.704

Tabela 6.10 – Tempo total para o *assembly* das matrizes globais, solução do sistema de equações e simulação para uma malha com 24143 elementos e $Re = 100$.

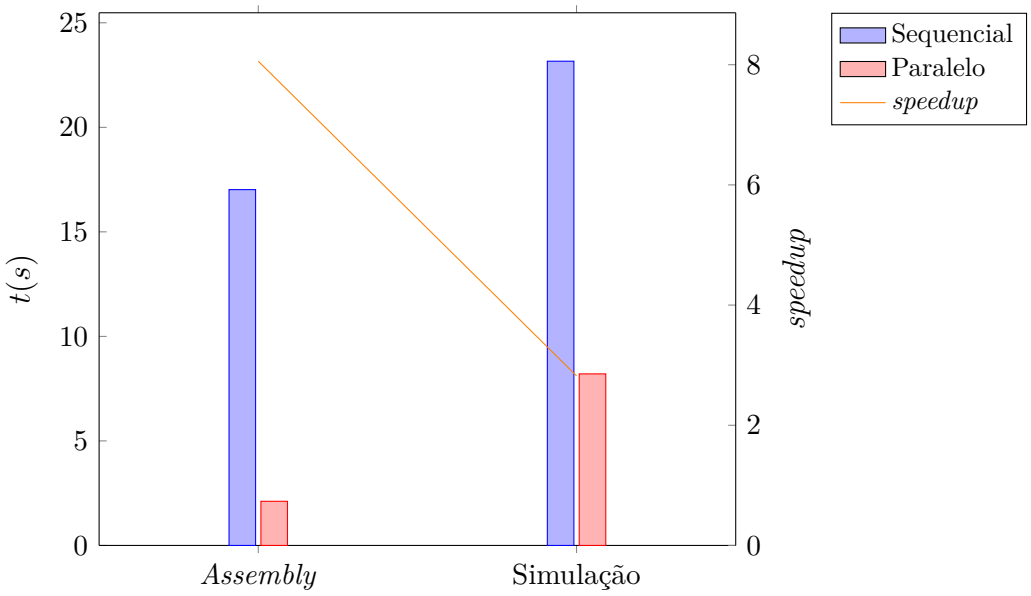
#Elementos = 24143				
Sequencial				
#Iteração	*Assembly* **K** e **G** (s)	*Assembly* **C** e $\mathbf{C_t}$ (s)	Solução (s)	Total (s)
–	2.881	–	–	2.881
1	–	4.686	2.089	6.775
2	–	4.680	2.020	6.700
3	–	4.696	2.047	6.743
4	–	4.679	2.004	6.683
5	–	4.690	2.036	6.726
6	–	4.685	2.026	6.711
	2.881	28.116	12.222	43.219
Paralelo				
–	0.601	–	–	0.601
1	–	0.516	2.030	2.546
2	–	0.486	1.984	2.470
3	–	0.491	2.036	2.527
4	–	0.498	1.994	2.492
5	–	0.490	2.040	2.530
6	–	0.501	2.055	2.556
	0.601	2.982	12.139	15.722

Tabela 6.11 – Tempo total para o *assembly* das matrizes globais, solução do sistema de equações e simulação para uma malha com 24143 elementos e $Re = 400$.

#Elementos = 24143				
Sequencial				
#Iteração	*Assembly* **K** e **G** (s)	*Assembly* **C** e $\mathbf{C_t}$ (s)	Solução (s)	Total (s)
–	2.859	–	–	2.859
1	–	4.711	2.075	6.786
2	–	4.730	2.001	6.731
3	–	4.728	2.023	6.751
4	–	4.711	2.004	6.715
5	–	4.728	2.021	6.749
6	–	4.723	2.054	6.777
7	–	4.728	2.005	6.733
8	–	4.712	2.043	6.755
	2.859	37.771	16.226	56.856
Paralelo				
–	0.601	–	–	0.601
1	–	0.535	2.108	2.643
2	–	0.494	2.072	2.566
3	–	0.498	2.027	2.525
4	–	0.492	2.068	2.560
5	–	0.481	2.012	2.493
6	–	0.492	2.013	2.505
7	–	0.502	2.051	2.553
8	–	0.500	1.994	2.494
	0.601	3.994	16.345	20.940

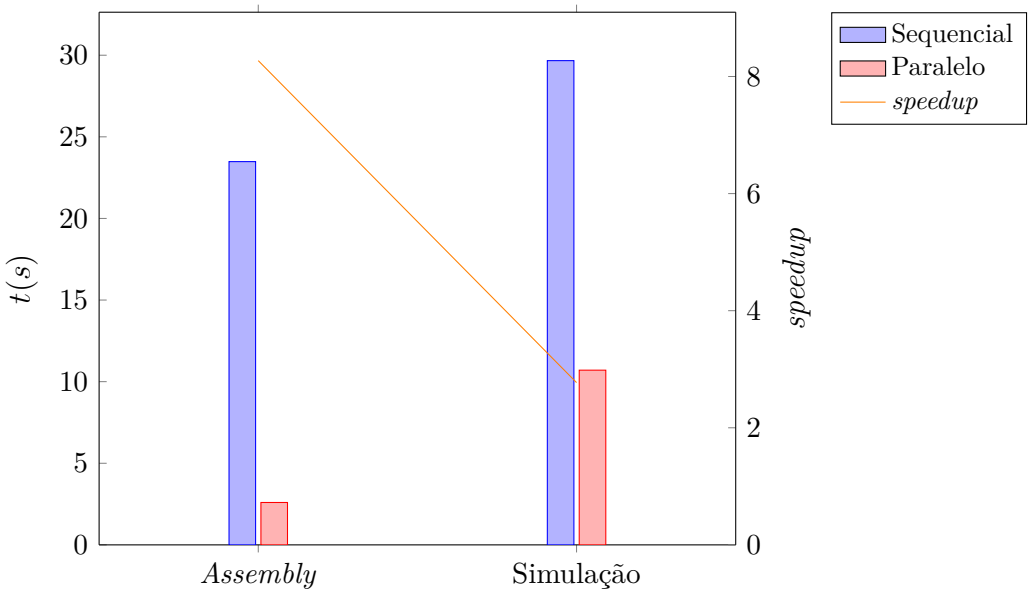
As Fig. 6.10, 6.11, 6.12 e 6.13 mostram o comparativo entre o processamento sequencial e paralelo para o tempo total do *assembly* e da simulação e o *speedup* alcançado para números de Reynolds variando entre 1 e 400.

Figura 6.10 – Comparativo do tempo total para o *assembly* das matrizes globais e da simulação e o *speedup* alcançado para $Re = 1$.



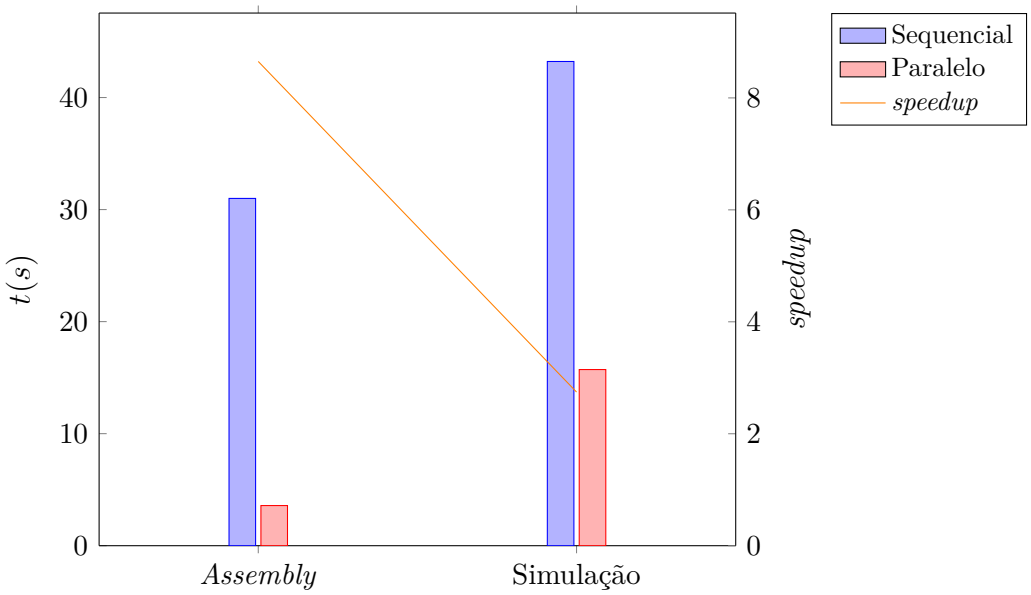
Fonte: O autor

Figura 6.11 – Comparativo do tempo total para o *assembly* das matrizes globais e da simulação e o *speedup* alcançado para $Re = 10$.



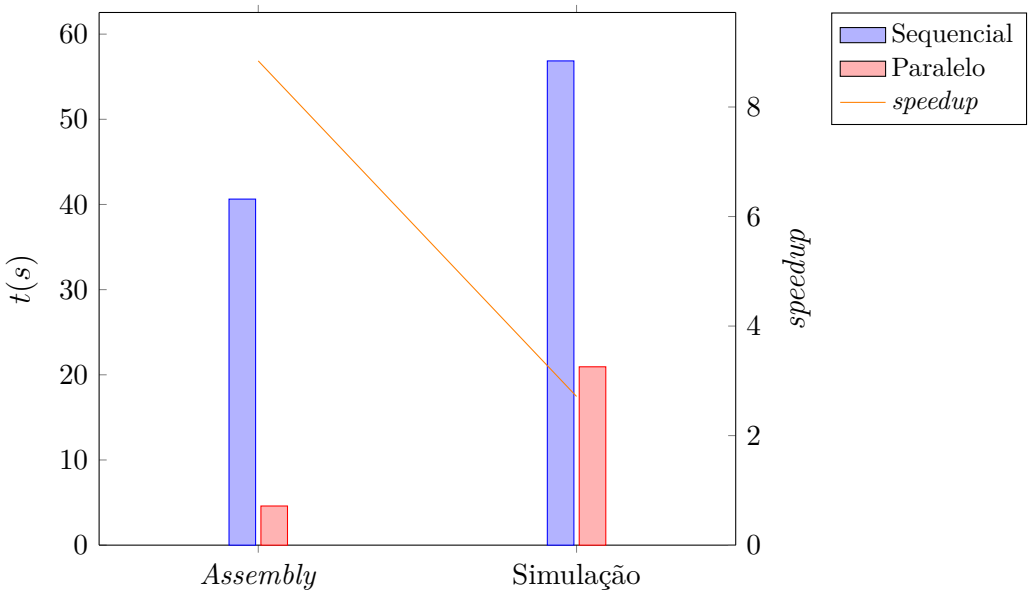
Fonte: O autor

Figura 6.12 – Comparativo do tempo total para o *assembly* das matrizes globais e da simulação e o *speedup* alcançado para $Re = 100$.



Fonte: O autor

Figura 6.13 – Comparativo do tempo total para o *assembly* das matrizes globais e da simulação e o *speedup* alcançado para $Re = 400$.



Fonte: O autor

Os resultados comparativos mostram uma considerável diminuição no tempo de processamento, com um *speedup* de aproximadamente 8 para o *assembly* das matrizes globais e de aproximadamente 3 para o tempo total da simulação. Estes valores vão de encontro com a análise de sensibilidade, apesar de apresentar *speedups* abaixo do esperado. Isto se deve ao fato de a malha possuir um maior refinamento próximo aos nós das extremidades da parede superior, aumentando assim, o tempo de processamento.

6.1.2 Cavidade com parede inferior em formato ondular

O escoamento em uma cavidade com parede inferior em formato ondular (similar ao exemplo apresentado em Zhang et al. (2012)) foi simulado para um escoamento laminar com número de Reynolds igual a 400 em uma malha conforme e em uma malha não-conforme utilizando o método das Fronteiras Imersas.

6.1.2.1 Dinâmica dos fluidos

6.1.2.1.1 Análise de sensibilidade

Para a análise de sensibilidade desta simulação foram utilizadas malhas com número crescente de elementos finitos do tipo P2P1. As Tab. 6.12, 6.13, 6.14, 6.15, 6.16, 6.17 e 6.18 apresentam o tempo total para o *assembly* das matrizes globais, solução do sistema de equações e simulação para uma malha com 98, 512, 1024, 5283, 11667, 21981 e 40285 elementos tanto para o processamento sequencial quanto para o paralelizado na GPU.

Tabela 6.12 – Tempo total para o *assembly* das matrizes globais, solução do sistema de equações e simulação para uma malha com 98 elementos.

#Elementos = 98				
Sequencial				
#Iteração	*Assembly* $\mathbf{K}$ e $\mathbf{G}$ (s)	*Assembly* $\mathbf{C}$ e $\mathbf{C_t}$ (s)	Solução (s)	Total (s)
–	0.025	–	–	0.025
1	–	0.021	0.110	0.131
2	–	0.027	0.007	0.034
3	–	0.027	0.006	0.033
4	–	0.021	0.006	0.027
5	–	0.021	0.005	0.026
6	–	0.020	0.006	0.026
7	–	0.020	0.005	0.025
	0.025	0.157	0.145	0.327
Paralelo				
–	0.015	–	–	0.015
1	–	0.015	0.122	0.137
2	–	0.015	0.000	0.015
3	–	0.003	0.000	0.003
4	–	0.015	0.000	0.015
5	–	0.016	0.015	0.031
6	–	0.015	0.015	0.030
7	–	0.020	0.000	0.020
	0.015	0.099	0.152	0.266

Tabela 6.13 – Tempo total para o *assembly* das matrizes globais, solução do sistema de equações e simulação para uma malha com 512 elementos.

#Elementos = 512				
Sequencial				
#Iteração	*Assembly* **K** e **G** (s)	*Assembly* **C** e $\mathbf{C_t}$ (s)	Solução (s)	Total (s)
–	0.065	–	–	0.065
1	–	0.100	0.124	0.224
2	–	0.099	0.026	0.125
3	–	0.099	0.024	0.123
4	–	0.101	0.025	0.126
5	–	0.097	0.023	0.12
6	–	0.098	0.024	0.122
7	–	0.099	0.024	0.123
8	–	0.100	0.025	0.125
	0.065	0.793	0.295	1.148
Paralelo				
–	0.015	–	–	0.015
1	–	0.033	0.115	0.148
2	–	0.000	0.022	0.022
3	–	0.015	0.015	0.030
4	–	0.000	0.031	0.031
5	–	0.006	0.031	0.037
6	–	0.015	0.033	0.048
7	–	0.012	0.037	0.049
8	–	0.012	0.038	0.050
	0.015	0.093	0.322	0.430

Tabela 6.14 – Tempo total para o *assembly* das matrizes globais, solução do sistema de equações e simulação para uma malha com 1024 elementos.

#Elementos = 1024				
Sequencial				
#Iteração	*Assembly* **K** e **G** (s)	*Assembly* **C** e $\mathbf{C_t}$ (s)	Solução (s)	Total (s)
–	0.124	–	–	0.124
1	–	0.192	0.148	0.34
2	–	0.195	0.053	0.248
3	–	0.192	0.053	0.245
4	–	0.196	0.052	0.248
5	–	0.195	0.053	0.248
6	–	0.196	0.052	0.248
7	–	0.193	0.054	0.247
8	–	0.195	0.052	0.247
	0.124	1.554	0.517	2.195
Paralelo				
–	0.037	–	–	0.037
1	–	0.031	0.138	0.169
2	–	0.033	0.080	0.113
3	–	0.022	0.065	0.087
4	–	0.023	0.067	0.090
5	–	0.021	0.064	0.085
6	–	0.021	0.066	0.087
7	–	0.021	0.061	0.082
8	–	0.011	0.046	0.057
	0.037	0.183	0.587	0.807

Tabela 6.15 – Tempo total para o *assembly* das matrizes globais, solução do sistema de equações e simulação para uma malha com 5283 elementos.

#Elementos = 5283				
Sequencial				
#Iteração	*Assembly* **K** e **G** (s)	*Assembly* **C** e $\mathbf{C_t}$ (s)	Solução (s)	Total (s)
–	0.624	–	–	0.624
1	–	1.000	0.429	1.429
2	–	0.997	0.334	1.331
3	–	1.000	0.338	1.338
4	–	1.007	0.339	1.346
5	–	1.002	0.334	1.336
6	–	0.997	0.336	1.333
7	–	1.004	0.339	1.343
8	–	0.998	0.339	1.337
	0.624	8.005	2.788	11.417
Paralelo				
–	0.147	–	–	0.147
1	–	0.115	0.405	0.520
2	–	0.100	0.223	0.323
3	–	0.100	0.227	0.327
4	–	0.100	0.220	0.320
5	–	0.100	0.221	0.321
6	–	0.100	0.225	0.325
7	–	0.100	0.227	0.327
8	–	0.100	0.235	0.335
	0.147	0.815	1.983	2.945

Tabela 6.16 – Tempo total para o *assembly* das matrizes globais, solução do sistema de equações e simulação para uma malha com 11667 elementos.

#Elementos = 11667				
Sequencial				
#Iteração	*Assembly* **K** e **G** (s)	*Assembly* **C** e $\mathbf{C_t}$ (s)	Solução (s)	Total (s)
–	1.374	–	–	1.374
1	–	2.209	0.93	3.139
2	–	2.209	0.854	3.063
3	–	2.214	0.839	3.053
4	–	2.208	0.843	3.051
5	–	2.222	0.843	3.065
6	–	2.213	0.838	3.051
7	–	2.212	0.838	3.05
8	–	2.221	0.847	3.068
	1.374	17.708	6.832	25.914
Paralelo				
–	0.285	–	–	0.285
1	–	0.25	0.607	0.857
2	–	0.245	0.544	0.789
3	–	0.240	0.530	0.770
4	–	0.249	0.518	0.767
5	–	0.246	0.528	0.774
6	–	0.239	0.530	0.769
7	–	0.247	0.527	0.774
8	–	0.246	0.525	0.771
	0.285	1.962	4.309	6.556

Tabela 6.17 – Tempo total para o *assembly* das matrizes globais, solução do sistema de equações e simulação para uma malha com 21981 elementos.

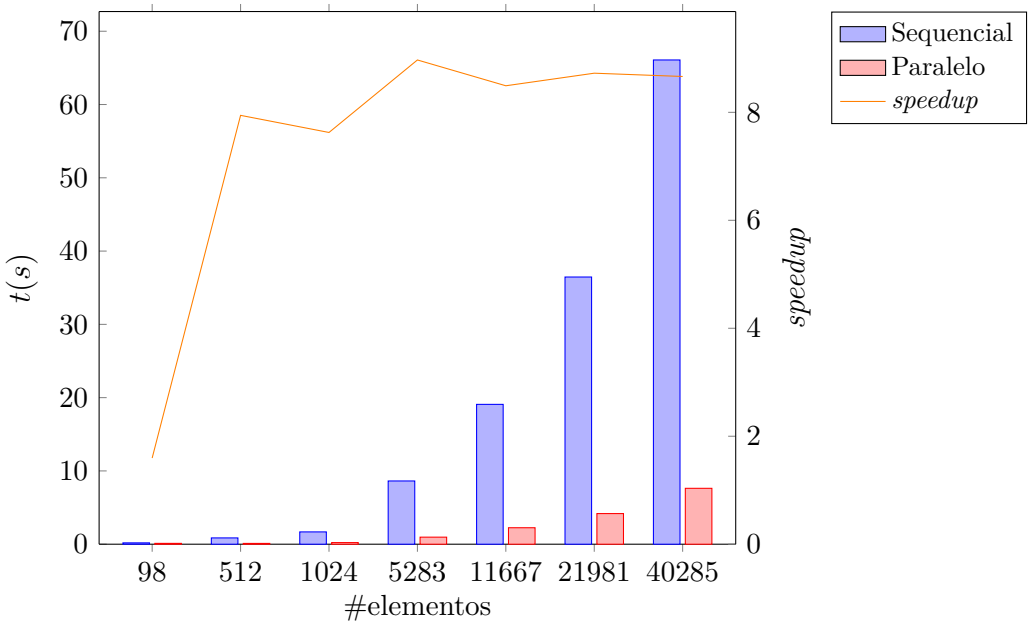
#Elementos = 21981				
Sequencial				
#Iteração	*Assembly* **K** e **G** (s)	*Assembly* **C** e $\mathbf{C_t}$ (s)	Solução (s)	Total (s)
–	2.592	–	–	2.592
1	–	4.242	1.996	6.238
2	–	4.305	1.866	6.171
3	–	4.266	1.783	6.049
4	–	4.22	1.814	6.034
5	–	4.218	1.791	6.009
6	–	4.213	1.78	5.993
7	–	4.207	1.783	5.99
8	–	4.198	1.842	6.04
	2.592	33.869	14.655	51.116
Paralelo				
–	0.552	–	–	0.552
1	–	0.473	1.178	1.651
2	–	0.450	1.052	1.502
3	–	0.448	1.050	1.498
4	–	0.448	1.051	1.499
5	–	0.449	1.102	1.551
6	–	0.452	1.103	1.555
7	–	0.456	1.084	1.54
8	–	0.451	1.062	1.513
	0.552	3.627	8.682	12.861

Tabela 6.18 – Tempo total para o *assembly* das matrizes globais, solução do sistema de equações e simulação para uma malha com 40285 elementos.

#Elementos = 40285				
Sequencial				
#Iteração	*Assembly* **K** e **G** (s)	*Assembly* **C** e $\mathbf{C_t}$ (s)	Solução (s)	Total (s)
–	4.741	–	–	4.741
1	–	7.631	3.610	11.241
2	–	7.703	3.602	11.305
3	–	7.700	3.700	11.400
4	–	7.658	3.628	11.286
5	–	7.691	3.692	11.383
6	–	7.616	3.674	11.290
7	–	7.664	3.937	11.601
8	–	7.676	3.783	11.459
	4.741	61.339	29.626	95.706
Paralelo				
–	1.01	–	–	1.01
1	–	0.845	2.216	3.061
2	–	0.811	2.349	3.16
3	–	0.816	2.233	3.049
4	–	0.835	2.377	3.212
5	–	0.817	2.237	3.054
6	–	0.821	2.265	3.086
7	–	0.854	2.231	3.085
8	–	0.818	2.209	3.027
	1.01	6.617	18.117	25.744

A Fig. 6.14 mostra o comparativo entre o processamento sequencial e paralelo para o tempo total do *assembly* das matrizes globais e o *speedup* alcançado.

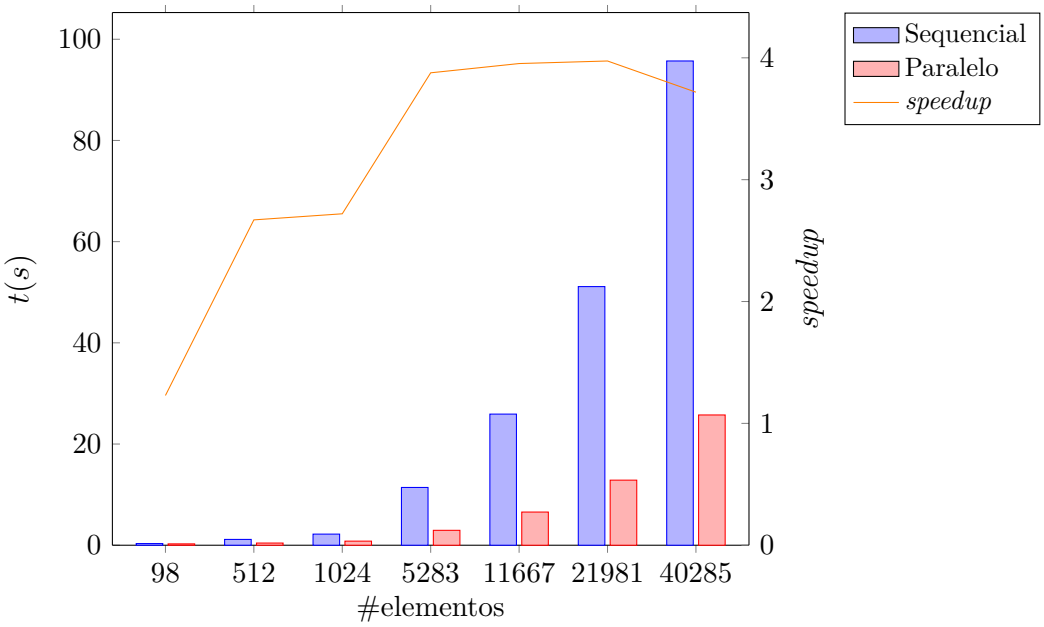
Figura 6.14 – Comparativo do tempo total para o *assembly* das matrizes globais e o *speedup* alcançado.



Fonte: O autor

A Fig. 6.15 mostra o comparativo entre o processamento sequencial e paralelo para o tempo total da simulação e o *speedup* alcançado.

Figura 6.15 – Comparativo do tempo total da simulação e o *speedup* alcançado.



Fonte: O autor

Os resultados comparativos mostram uma considerável diminuição no tempo de processamento, com um *speedup* de aproximadamente 9 para o *assembly* das matrizes

globais para a malha de elementos finitos mais refinada. A curva do *speedup*, como pode ser observado na Fig. 6.14, apresenta uma diminuição em sua inclinação, acima de 5000 elementos. Ou seja, acima deste valor, aumentando-se o número de elementos não obter-se-á um aumento significativo de *speedup*, como acontece para as malhas menos refinadas.

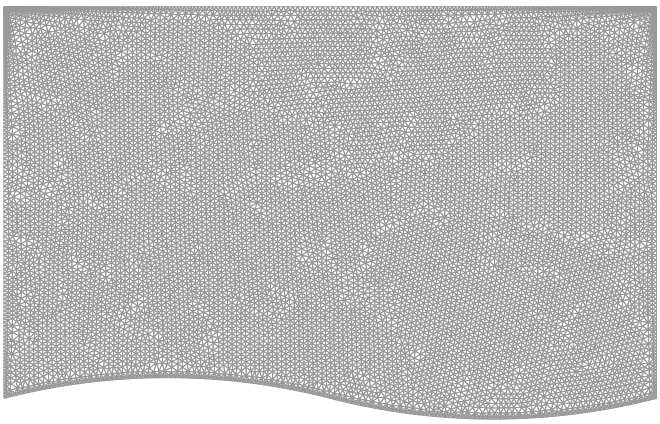
O tempo total para solução do problema mostra um *speedup* de aproximadamente 3.5 para a malha de elementos finitos mais refinada. Isto se deve ao fato de que a solução do sistema de equações lineares é realizada de forma sequencial utilizando o pacote PARDISO. A mesma observação anterior – aumento do número de elementos acima de 5000 elementos não obtém-se um considerável aumento de *speedup* – pode ser feita para o comparativo do tempo do total para solução do problema, inclusive, a partir de 22000 elementos, há uma diminuição desse *speedup*.

Pode-se observar que em uma malha pouco refinada não há vantagem em se utilizar o processamento paralelo ao invés do sequencial (*speedup* de aproximadamente 1.5). A justificativa para tal fenômeno está relacionado com o tempo necessário para que a GPU inicie o processo de paralelização (inicialização dos blocos de *threads*, das *threads* e outros processos de inicializações internos da placa gráfica).

6.1.2.1.2 Resultados

Executou-se o problema de dinâmica dos fluidos com uma malha conforme com 23453 elementos do tipo P2P1 e 47758 nós como mostra a Fig. 6.16.

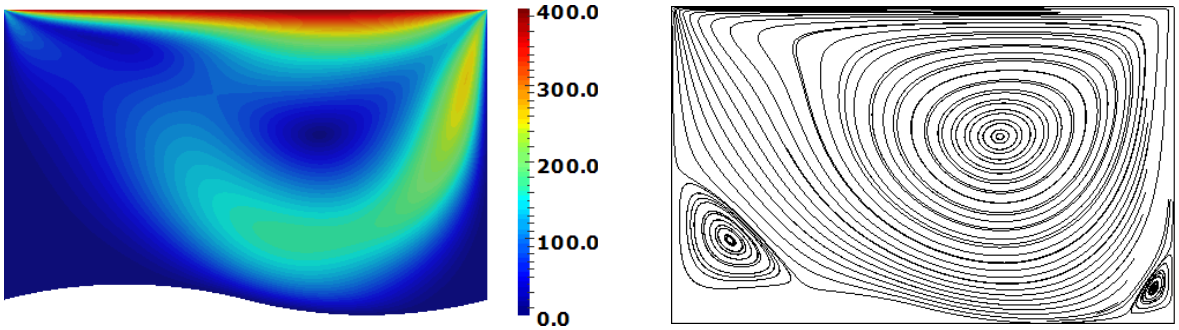
Figura 6.16 – Malha de elementos finitos com 23453 elementos do tipo P2P1 e 47758 nós.



Fonte: O autor

A Fig. 6.17 apresenta os perfis de velocidade e linhas de corrente no domínio.

Figura 6.17 – Perfis de velocidade (a) e linhas de corrente (b) para $Re = 400$ em uma cavidade com malha conforme.



(a) (b)

Fonte: O autor

6.1.2.1.3 Análise de desempenho

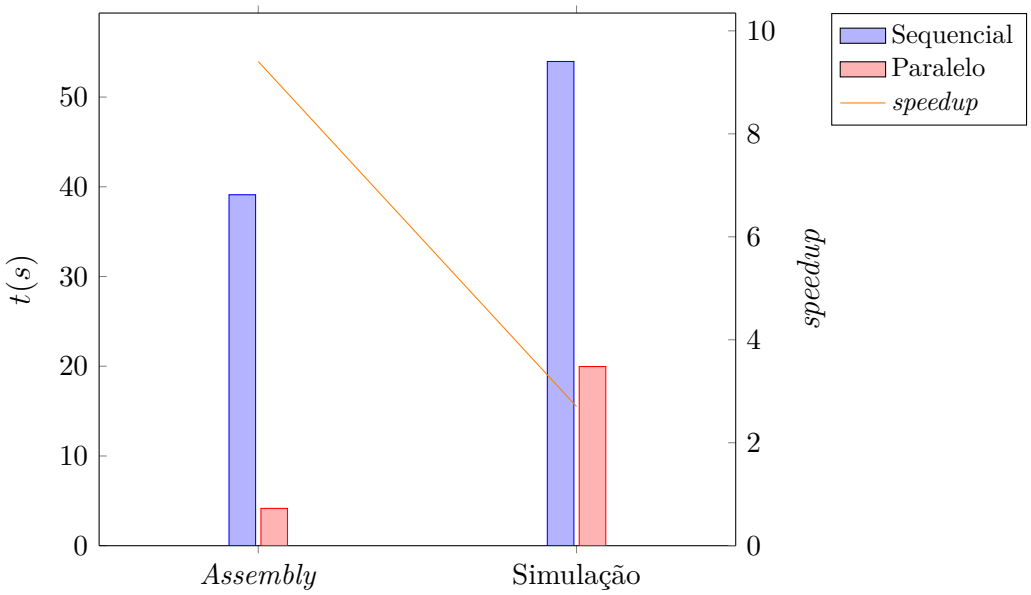
As Tab. 6.19 apresenta o tempo total para o *assembly* das matrizes globais, solução do sistema de equações e simulação para uma malha com 23453 elementos do tipo P2P1 tanto para o processamento sequencial quanto para o paralelizado na GPU.

Tabela 6.19 – Tempo total para o *assembly* das matrizes globais, solução do sistema de equações e simulação para uma malha com 23453 elementos.

#Elementos = 23453				
Sequencial				
#Iteração	*Assembly* $\mathbf{K}$ e $\mathbf{G}$ (s)	*Assembly* $\mathbf{C}$ e $\mathbf{C_t}$ (s)	Solução (s)	Total (s)
–	2.806	–	–	2.806
1	–	4.511	1.873	6.384
2	–	4.538	1.889	6.427
3	–	4.541	1.842	6.383
4	–	4.542	1.842	6.384
5	–	4.526	1.856	6.382
6	–	4.549	1.851	6.400
7	–	4.543	1.842	6.385
8	–	4.558	1.858	6.416
	2.806	36.308	14.853	53.967
Paralelo				
–	0.586	–	–	0.586
1	–	0.523	2.087	2.61
2	–	0.472	2.025	2.497
3	–	0.482	1.827	2.309
4	–	0.477	1.888	2.365
5	–	0.487	1.94	2.427
6	–	0.48	1.857	2.337
7	–	0.476	1.947	2.423
8	–	0.476	1.939	2.415
	0.586	3.873	15.510	19.969

A Fig. 6.18 mostra o comparativo entre o processamento sequencial e paralelo para o tempo total do *assembly* e da simulação e o *speedup* alcançado.

Figura 6.18 – Comparativo do tempo total do *assembly* das matrizes globais e da simulação e o *speedup* alcançado.



Fonte: O autor

Os resultados comparativos mostram uma considerável diminuição no tempo de processamento, com um *speedup* de aproximadamente 9 para o *assembly* das matrizes globais e de aproximadamente 3 para o tempo total da simulação. Estes valores vão de encontro com a análise de sensibilidade, apesar de apresentar *speedups* abaixo do esperado. Isto se deve ao fato de a malha possuir um maior refinamento tanto próximo aos nós das extremidades da parede superior como na parede inferior, aumentando assim, o tempo de processamento.

6.1.2.2 Interação fluido-estrutura pelo método das Fronteiras Imersas

6.1.2.2.1 Análise de sensibilidade

Da mesma forma que no problema de dinâmica dos fluidos, utilizou-se para a análise de sensibilidade malhas com número crescente de elementos finitos do tipo P2P1. As Tab. 6.20, 6.21, 6.22, 6.23, 6.24, 6.25 e 6.26 apresentam o tempo total para o *assembly* das matrizes globais, solução do sistema de equações e simulação para uma malha com 98, 512, 1058, 5618, 11858, 20000 e 40328 elementos tanto para o processamento sequencial quanto para o paralelizado na GPU.

Tabela 6.20 – Tempo total para o *assembly* das matrizes globais, solução do sistema de equações e simulação para uma malha com 98 elementos.

#Elementos = 98				
Sequencial				
#Iteração	*Assembly* **K** e **G** (s)	*Assembly* **C** e $\mathbf{C_t}$ (s)	Solução (s)	Total (s)
–	0.012	–	–	0.012
1	–	0.017	0.105	0.122
2	–	0.017	0.007	0.024
3	–	0.017	0.005	0.022
4	–	0.017	0.005	0.022
5	–	0.016	0.005	0.021
6	–	0.017	0.005	0.022
7	–	0.017	0.006	0.023
	0.012	0.118	0.138	0.268
Paralelo				
–	0.015	–	–	0.015
1	–	0.022	0.100	0.122
2	–	0.015	0.015	0.030
3	–	0.015	0.020	0.035
4	–	0.010	0.016	0.026
5	–	0.011	0.018	0.029
6	–	0.010	0.018	0.028
7	–	0.010	0.018	0.028
	0.015	0.093	0.205	0.313

Tabela 6.21 – Tempo total para o *assembly* das matrizes globais, solução do sistema de equações e simulação para uma malha com 512 elementos.

#Elementos = 512				
Sequencial				
#Iteração	*Assembly* **K** e **G** (s)	*Assembly* **C** e $\mathbf{C_t}$ (s)	Solução (s)	Total (s)
–	0.045	–	–	0.045
1	–	0.066	0.117	0.183
2	–	0.067	0.019	0.086
3	–	0.067	0.018	0.085
4	–	0.067	0.017	0.084
5	–	0.067	0.018	0.085
6	–	0.067	0.017	0.084
7	–	0.066	0.019	0.085
8	–	0.069	0.018	0.087
	0.045	0.536	0.243	0.824
Paralelo				
–	0.031	–	–	0.031
1	–	0.031	0.169	0.200
2	–	0.022	0.048	0.070
3	–	0.024	0.060	0.084
4	–	0.022	0.046	0.068
5	–	0.022	0.047	0.069
6	–	0.022	0.060	0.082
7	–	0.015	0.015	0.030
8	–	0.022	0.015	0.037
	0.031	0.180	0.460	0.671

Tabela 6.22 – Tempo total para o *assembly* das matrizes globais, solução do sistema de equações e simulação para uma malha com 1058 elementos.

#Elementos = 1058				
Sequencial				
#Iteração	*Assembly* **K** e **G** (s)	*Assembly* **C** e $\mathbf{C_t}$ (s)	Solução (s)	Total (s)
–	0.085	–	–	0.085
1	–	0.131	0.141	0.272
2	–	0.133	0.039	0.172
3	–	0.130	0.037	0.167
4	–	0.129	0.037	0.166
5	–	0.132	0.037	0.169
6	–	0.130	0.037	0.167
7	–	0.129	0.037	0.166
8	–	0.131	0.038	0.169
	0.085	1.045	0.403	1.533
Paralelo				
–	0.046	–	–	0.046
1	–	0.062	0.205	0.267
2	–	0.039	0.106	0.145
3	–	0.039	0.106	0.145
4	–	0.039	0.046	0.085
5	–	0.040	0.015	0.055
6	–	0.046	0.031	0.077
7	–	0.031	0.031	0.062
8	–	0.046	0.037	0.083
	0.046	0.342	0.577	0.965

Tabela 6.23 – Tempo total para o *assembly* das matrizes globais, solução do sistema de equações e simulação para uma malha com 5618 elementos.

#Elementos = 5618				
Sequencial				
#Iteração	*Assembly* **K** e **G** (s)	*Assembly* **C** e $\mathbf{C_t}$ (s)	Solução (s)	Total (s)
–	0.417	–	–	0.417
1	–	0.666	0.351	1.017
2	–	0.682	0.230	0.912
3	–	0.687	0.255	0.942
4	–	0.686	0.241	0.927
5	–	0.677	0.249	0.926
6	–	0.682	0.253	0.935
7	–	0.679	0.260	0.939
8	–	0.68	0.2420	0.922
	0.417	5.439	2.081	7.937
Paralelo				
–	0.200	–	–	0.200
1	–	0.168	0.278	0.446
2	–	0.163	0.162	0.325
3	–	0.147	0.216	0.363
4	–	0.158	0.185	0.343
5	–	0.163	0.188	0.351
6	–	0.163	0.191	0.354
7	–	0.162	0.181	0.343
8	–	0.163	0.191	0.354
	0.200	1.287	1.592	3.079

Tabela 6.24 – Tempo total para o *assembly* das matrizes globais, solução do sistema de equações e simulação para uma malha com 11858 elementos.

#Elementos = 11858				
Sequencial				
#Iteração	*Assembly* **K** e **G** (s)	*Assembly* **C** e $\mathbf{C_t}$ (s)	Solução (s)	Total (s)
–	0.863	–	–	0.863
1	–	1.412	0.644	2.056
2	–	1.418	0.522	1.940
3	–	1.420	0.522	1.942
4	–	1.425	0.496	1.921
5	–	1.420	0.503	1.923
6	–	1.420	0.498	1.918
7	–	1.411	0.499	1.910
8	–	1.412	0.495	1.907
	0.863	11.338	4.179	16.380
Paralelo				
–	0.381	–	–	0.381
1	–	0.331	0.459	0.790
2	–	0.317	0.369	0.686
3	–	0.320	0.353	0.673
4	–	0.317	0.380	0.697
5	–	0.321	0.354	0.675
6	–	0.311	0.358	0.669
7	–	0.321	0.364	0.685
8	–	0.324	0.379	0.703
	0.381	2.562	3.016	5.959

Tabela 6.25 – Tempo total para o *assembly* das matrizes globais, solução do sistema de equações e simulação para uma malha com 20000 elementos.

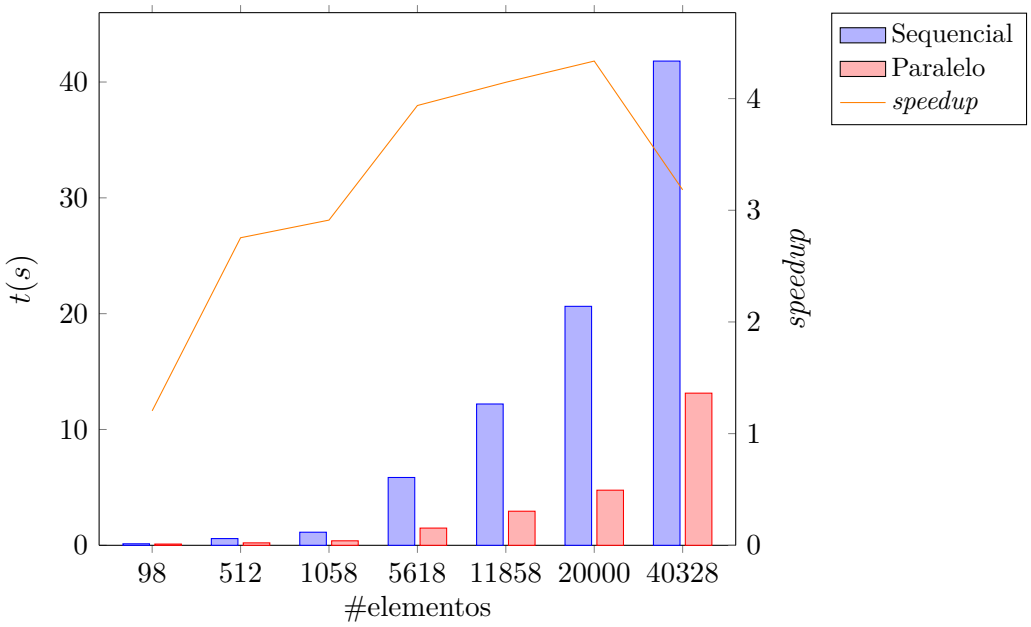
#Elementos = 20000				
Sequencial				
#Iteração	*Assembly* **K** e **G** (s)	*Assembly* **C** e $\mathbf{C_t}$ (s)	Solução (s)	Total (s)
–	1.479	–	–	1.479
1	–	2.373	1.206	3.579
2	–	2.390	0.984	3.374
3	–	2.384	0.977	3.361
4	–	2.404	0.961	3.365
5	–	2.411	0.968	3.379
6	–	2.400	0.963	3.363
7	–	2.389	0.969	3.358
8	–	2.403	0.970	3.373
	1.479	19.154	7.998	28.631
Paralelo				
–	0.627	–	–	0.627
1	–	0.539	0.791	1.330
2	–	0.512	0.711	1.223
3	–	0.512	0.733	1.245
4	–	0.509	0.701	1.210
5	–	0.517	0.827	1.344
6	–	0.511	0.711	1.222
7	–	0.520	0.736	1.256
8	–	0.512	0.710	1.222
	0.627	4.132	5.920	10.679

Tabela 6.26 – Tempo total para o *assembly* das matrizes globais, solução do sistema de equações e simulação para uma malha com 40328 elementos.

#Elementos = 40328				
Sequencial				
#Iteração	*Assembly* **K** e **G** (s)	*Assembly* **C** e $\mathbf{C_t}$ (s)	Solução (s)	Total (s)
–	3.006	–	–	3.006
1	–	4.791	2.349	7.140
2	–	4.960	2.062	7.022
3	–	4.842	2.155	6.997
4	–	4.854	2.070	6.924
5	–	4.866	2.107	6.973
6	–	4.857	2.023	6.880
7	–	4.835	2.036	6.871
8	–	4.798	2.007	6.805
	3.006	38.803	16.809	58.618
Paralelo				
–	4.507	–	–	4.507
1	–	1.110	1.456	2.566
2	–	1.060	1.353	2.413
3	–	1.071	1.322	2.393
4	–	1.066	1.578	2.644
5	–	1.113	1.323	2.436
6	–	1.049	1.315	2.364
7	–	1.071	1.462	2.533
8	–	1.090	1.352	2.442
	4.507	8.630	11.161	24.298

A Fig. 6.19 mostra o comparativo entre o processamento sequencial e paralelo para o tempo total do *assembly* das matrizes globais e o *speedup* alcançado.

Figura 6.19 – Comparativo do tempo total para o *assembly* das matrizes globais e o *speedup* alcançado.

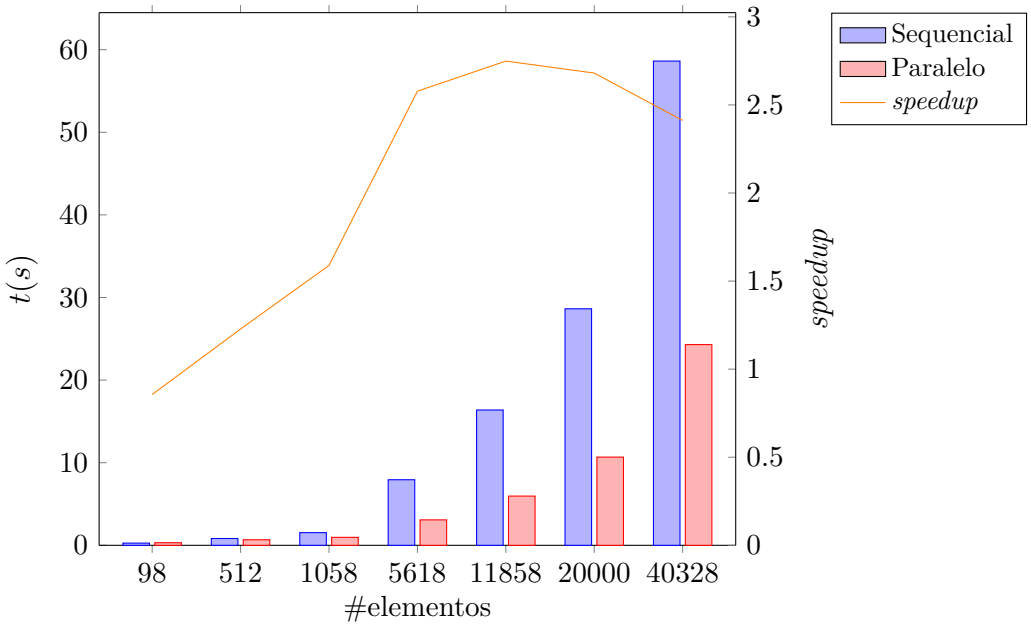


Fonte: O autor

A Fig. 6.20 mostra o comparativo entre o processamento sequencial e paralelo para

o tempo total da simulação e o *speedup* alcançado.

Figura 6.20 – Comparativo do tempo total da simulação e o *speedup* alcançado.



Fonte: O autor

Os resultados comparativos mostram uma considerável diminuição no tempo de processamento, com um *speedup* de aproximadamente 4 para o *assembly* das matrizes globais para a malha com 20000 elementos. A curva do *speedup*, como pode ser observado na Fig. 6.19, apresenta uma diminuição em sua inclinação, acima de 20000 elementos.

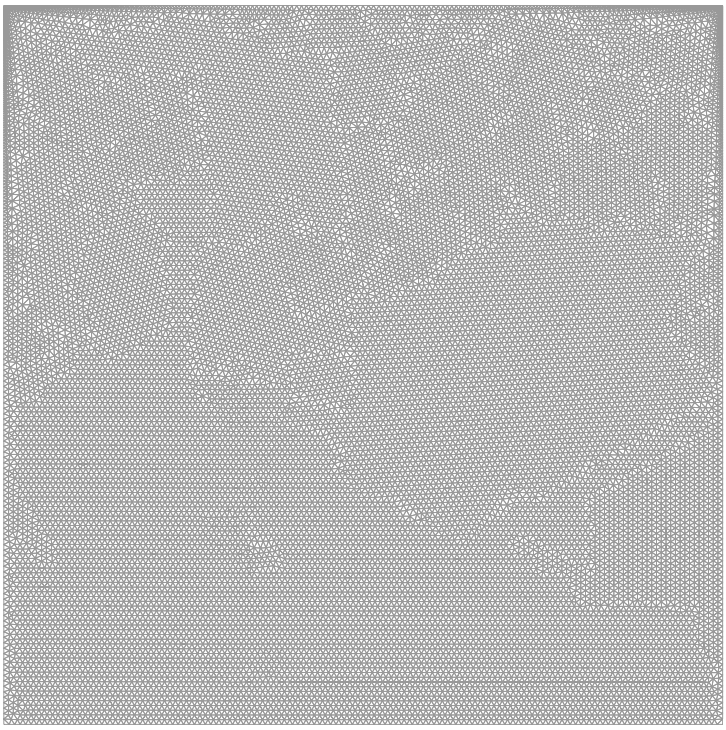
O tempo total para solução do problema mostra um *speedup* de aproximadamente 2.5 para a malha de elementos finitos mais refinada. Isto se deve ao fato de que a solução do sistema de equações lineares é realizada de forma sequencial utilizando o pacote PARDISO. Acima de 12000 elementos a curva do *speedup* apresenta uma diminuição em sua inclinação.

Pode-se observar que em uma malha pouco refinada o processamento sequencial é mais vantajoso que o processamento paralelizado na GPU. A justificativa para tal fenômeno está relacionado com o tempo necessário para que a GPU inicie o processo de paralelização (inicialização dos blocos de *threads*, das *threads* e outros processos de inicializações internos da placa gráfica).

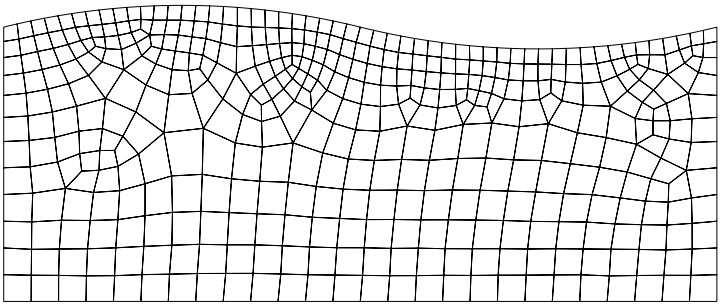
6.1.2.2.2 Resultados

Executou-se o mesmo problema com o método das Fronteiras Imersas, utilizando-se uma malha para o fluido com 41228 elementos do tipo P2P1 e 83341 nós e uma malha para a estrutura constituída de 408 elementos quadrangulares de 9 nós e 1735 nós, sobreposta a essa, simulando a parede (Fig. 6.21).

Figura 6.21 – (a) Malha de elementos finitos do fluido com 41228 elementos do tipo P2P1 e 83341 nós e (b) malha de elementos finitos da estrutura com 408 elementos quadrangulares e 1735 nós.



(a)

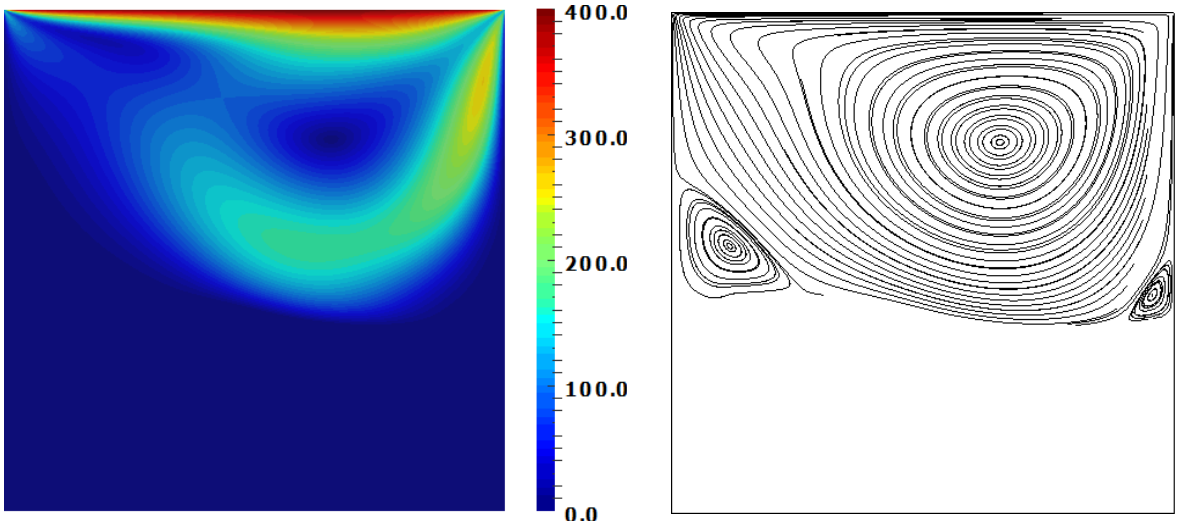


(b)

Fonte: O autor

A Fig. 6.22 apresenta os perfis de velocidade e linhas de corrente no domínio.

Figura 6.22 – Perfis de velocidade (a) e linhas de corrente (b) para $Re = 400$ em uma cavidade com malha não-conforme utilizando o método das Fronteiras Imersas.

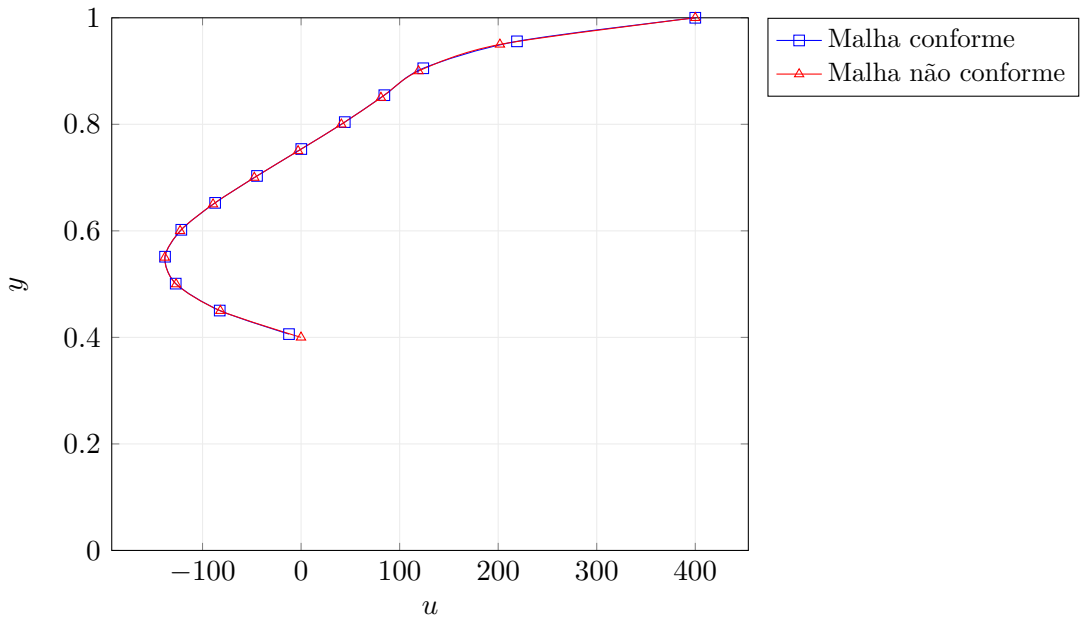


(a) (b)

Fonte: O autor

A Fig. 6.23 mostra o perfil de velocidades para validação e comparação dos resultados entre as duas malhas apresentadas com números de Reynolds igual a 400.

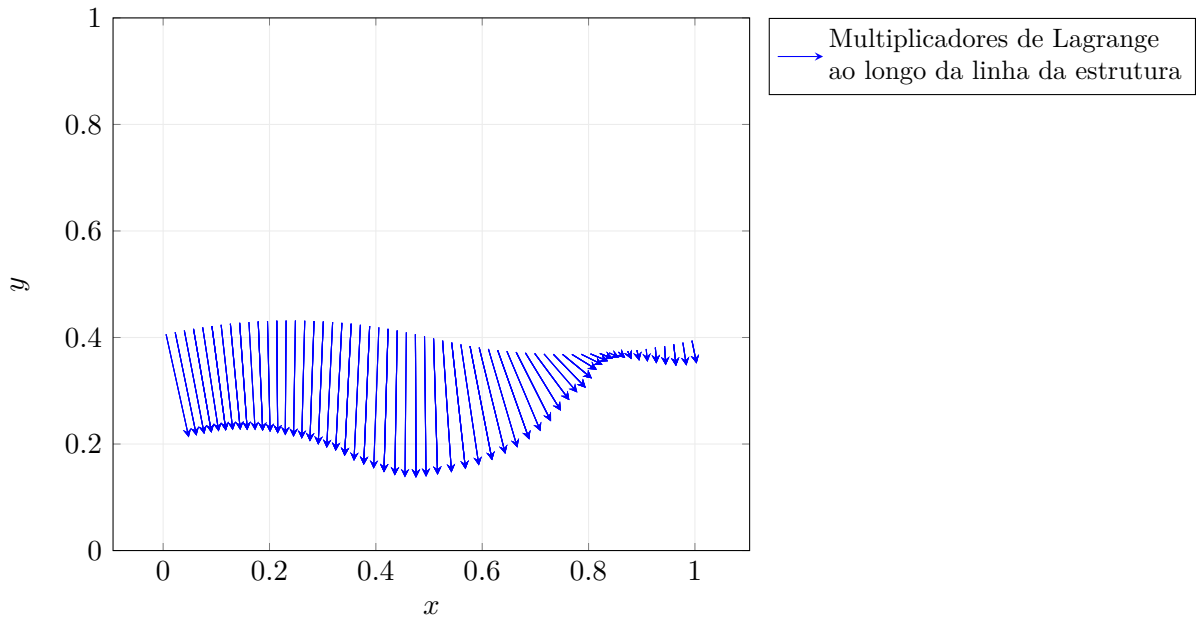
Figura 6.23 – Velocidade u ao longo da linha vertical sobre o centro geométrico da cavidade.



Fonte: O autor

A Fig. 6.24 mostra os multiplicadores de Lagrange ao longo da linha da estrutura.

Figura 6.24 – Multiplicadores de Lagrange ao longo da linha da estrutura.



Fonte: O autor

6.1.2.2.3 Análise de desempenho

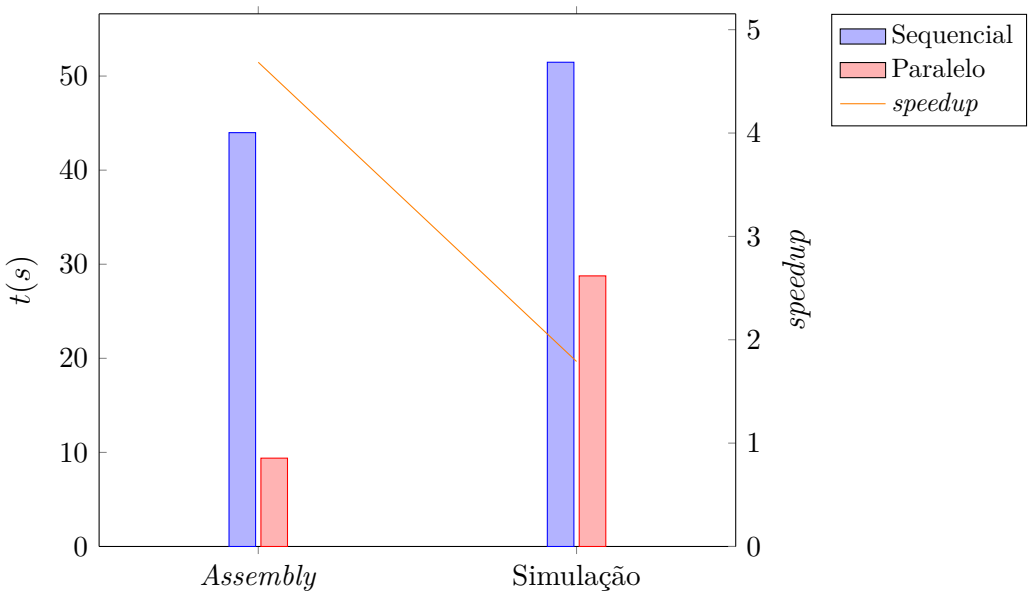
As Tab. 6.27 apresenta o tempo total para o *assembly* das matrizes globais, solução do sistema de equações e simulação para uma malha com 41228 elementos do tipo P2P1 tanto para o processamento sequencial quanto para o paralelizado na GPU.

Tabela 6.27 – Tempo total para o *assembly* das matrizes globais, solução do sistema de equações e simulação para uma malha com 41228 elementos.

#Elementos = 41228				
Sequencial				
#Iteração	*Assembly* **K** e **G** (s)	*Assembly* **C** e $\mathbf{C_t}$ (s)	Solução (s)	Total (s)
–	3.176	–	–	3.176
1	–	5.059	2.336	7.395
2	–	5.429	2.225	7.654
3	–	5.061	2.203	7.264
4	–	5.06	2.228	7.288
5	–	5.042	2.242	7.284
6	–	5.06	2.243	7.303
7	–	5.029	2.252	7.281
8	–	5.07	2.206	7.276
	3.176	40.81	17.935	51.469
Paralelo				
–	1.271	–	–	1.271
1	–	1.049	3.364	4.413
2	–	0.992	3.283	4.275
3	–	1.024	3.085	4.109
4	–	0.997	2.867	3.864
5	–	1.006	2.938	3.944
6	–	1.033	3.107	4.14
7	–	1.009	3.007	4.016
8	–	1.007	3.088	4.095
	1.271	8.117	24.739	28.761

A Fig. 6.25 mostra o comparativo entre o processamento sequencial e paralelo para o tempo total do *assembly* e da simulação e o *speedup* alcançado.

Figura 6.25 – Comparativo do tempo total do *assembly* das matrizes globais e da simulação e o *speedup* alcançado.



Fonte: O autor

Os resultados comparativos mostram uma considerável diminuição no tempo de

processamento, com um *speedup* de aproximadamente 5 para o *assembly* das matrizes globais e de aproximadamente 2 para o tempo total da simulação. Estes valores vão de encontro com a análise de sensibilidade, apesar de apresentar *speedups* abaixo do esperado. Isto se deve ao fato de a malha possuir um maior refinamento próximo aos nós das extremidades da parede superior, aumentando assim, o tempo de processamento.

6.2 Escoamento em torno de um cilindro

Este é outro exemplo clássico da literatura e muito utilizado como um caso de validação tanto para códigos CFD quanto para códigos de interação fluido-estrutura. Para fins de verificação e validação do código em GPU, os parâmetros encontrados em Schaffer et al. (1996) serão utilizados uma vez que esta referência apresenta resultados de simulações de diversos grupos de pesquisa em atividade.

Serão apresentados dois exemplos, um com malha conforme e outro com malha não conforme utilizando o método das Fronteiras Imersas. Os coeficientes de sustentação c_L e de arrasto c_D (Eq. 6.3) foram calculados para uma avaliação quantitativa, validação do *software* e comparação com resultados de outros grupos de pesquisa como Gomes (2013) e Schaffer et al. (1996) – dos grupos de pesquisa escolhidos para o *benchmark*, este trabalho utilizou os resultados do grupo de Bänsch, E. *et al.* da *Univ. Freiburg, Inst. für Angewandte Mathematik* pois resolveram o problema de dinâmica dos fluidos pelo método dos elementos finitos com malha não-estruturada e elementos do tipo P2P1.

Os coeficientes de arrasto e sustentação podem ser calculados da seguinte forma:

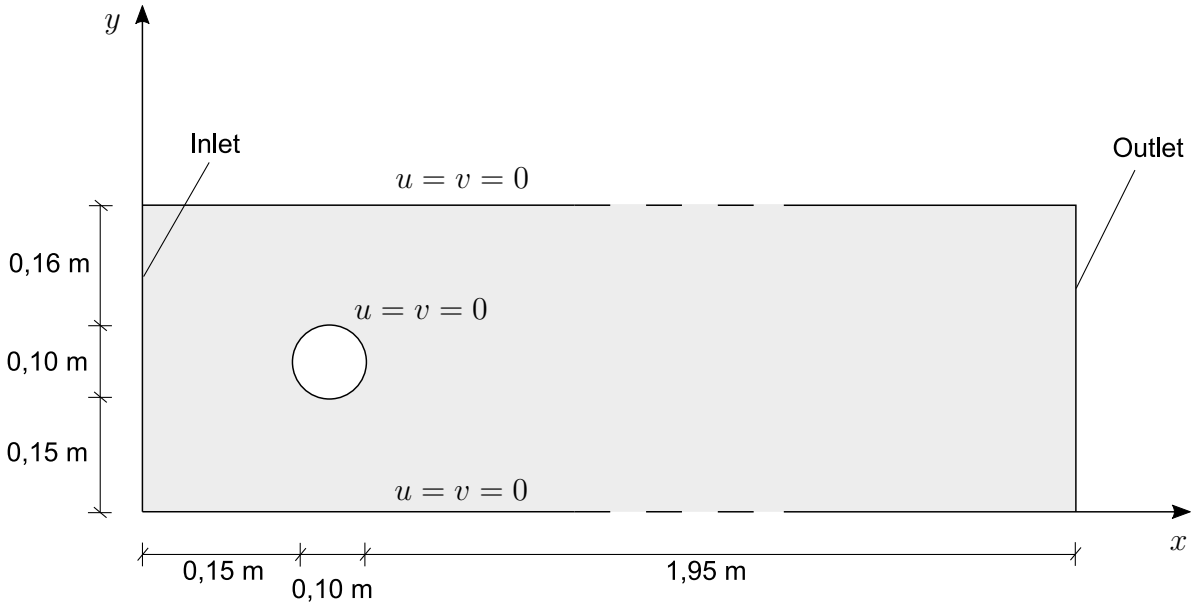
$$c_L = \frac{2F_v}{\rho \overline{U} D} \quad \text{e} \quad c_D = \frac{2F_h}{\rho \overline{U} D} \tag{6.3}$$

onde:

- F_v: Força resultante vertical;
- F_h: Força resultante horizontal;
- ρ: Densidade do fluido;
- $\overline{U}$: Velocidade média do fluido – $\overline{U}(t) = 2U(0, H/2, t)/3$;
- D: Diâmetro do cilindro.

O escoamento se dá através de um canal estreito de comprimento $L = 2,2m$ e altura $H = 0,41m$, com origem do sistema de referência no canto inferior esquerdo. A Fig. 6.26 descreve a geometria e as condições de contorno utilizadas nos exemplos a seguir.

Figura 6.26 – Geometria do modelo.



Fonte: O autor

6.2.1 Dinâmica dos fluidos

6.2.1.1 Análise de sensibilidade

Para a análise de sensibilidade desta simulação foram utilizadas malhas com número crescente de elementos finitos do tipo P2P1. As Tab. 6.28, 6.29, 6.30, 6.31, 6.32, 6.33 e 6.34 apresentam o tempo total para o *assembly* das matrizes globais, solução do sistema de equações e simulação para uma malha com 100, 596, 1056, 5200, 10406, 20322 e 41612 elementos tanto para o processamento sequencial quanto para o paralelizado na GPU.

Tabela 6.28 – Tempo total para o *assembly* das matrizes globais, solução do sistema de equações e simulação para uma malha com 100 elementos.

#Elementos = 100				
Sequencial				
#Iteração	*Assembly* **K** e **G** (s)	*Assembly* **C** e $\mathbf{C_t}$ (s)	Solução (s)	Total (s)
–	0.034	–	–	0.034
1	–	0.015	0.100	0.115
2	–	0.022	0.000	0.022
3	–	0.037	0.000	0.037
4	–	0.015	0.003	0.018
5	–	0.031	0.000	0.031
6	–	0.019	0.000	0.019
	0.034	0.139	0.103	0.242
Paralelo				
–	0.015	–	–	0.015
1	–	0.015	0.100	0.115
2	–	0.003	0.015	0.018
3	–	0.000	0.015	0.015
4	–	0.000	0.015	0.015
5	–	0.000	0.015	0.015
6	–	0.000	0.000	0.000
	0.015	0.018	0.160	0.178

Tabela 6.29 – Tempo total para o *assembly* das matrizes globais, solução do sistema de equações e simulação para uma malha com 596 elementos.

#Elementos = 596				
Sequencial				
#Iteração	*Assembly* **K** e **G** (s)	*Assembly* **C** e $\mathbf{C_t}$ (s)	Solução (s)	Total (s)
–	0.084	–	–	0.084
1	–	0.131	0.115	0.246
2	–	0.131	0.016	0.147
3	–	0.116	0.031	0.147
4	–	0.122	0.015	0.137
5	–	0.135	0.031	0.166
6	–	0.132	0.031	0.163
	0.084	0.767	0.239	1.006
Paralelo				
–	0.017	–	–	0.017
1	–	0.031	0.122	0.153
2	–	0.015	0.015	0.03
3	–	0.015	0.046	0.061
4	–	0.015	0.046	0.061
5	–	0.014	0.041	0.055
6	–	0.014	0.041	0.055
	0.017	0.104	0.311	0.415

Tabela 6.30 – Tempo total para o *assembly* das matrizes globais, solução do sistema de equações e simulação para uma malha com 1056 elementos.

#Elementos = 1056				
Sequencial				
#Iteração	*Assembly* **K** e **G** (s)	*Assembly* **C** e $\mathbf{C_t}$ (s)	Solução (s)	Total (s)
–	0.162	–	–	0.162
1	–	0.238	0.147	0.385
2	–	0.232	0.053	0.285
3	–	0.231	0.046	0.277
4	–	0.222	0.046	0.268
5	–	0.232	0.053	0.285
6	–	0.231	0.047	0.278
	0.162	1.386	0.392	1.778
Paralelo				
–	0.031	–	–	0.031
1	–	0.031	0.147	0.178
2	–	0.022	0.046	0.068
3	–	0.022	0.081	0.103
4	–	0.023	0.065	0.088
5	–	0.024	0.085	0.109
6	–	0.024	0.066	0.09
	0.031	0.146	0.490	0.636

Tabela 6.31 – Tempo total para o *assembly* das matrizes globais, solução do sistema de equações e simulação para uma malha com 5200 elementos.

#Elementos = 5200				
Sequencial				
#Iteração	*Assembly* **K** e **G** (s)	*Assembly* **C** e $\mathbf{C_t}$ (s)	Solução (s)	Total (s)
–	0.736	–	–	0.736
1	–	1.159	0.369	1.528
2	–	1.164	0.285	1.449
3	–	1.160	0.290	1.45
4	–	1.164	0.300	1.464
5	–	1.176	0.284	1.46
6	–	1.156	0.300	1.456
	0.736	6.979	1.828	8.807
Paralelo				
–	0.147	–	–	0.147
1	–	0.105	0.406	0.511
2	–	0.100	0.207	0.307
3	–	0.100	0.216	0.316
4	–	0.100	0.216	0.316
5	–	0.100	0.199	0.299
6	–	0.100	0.212	0.312
	0.147	0.605	1.456	2.061

Tabela 6.32 – Tempo total para o *assembly* das matrizes globais, solução do sistema de equações e simulação para uma malha com 10406 elementos.

#Elementos = 10406				
Sequencial				
#Iteração	*Assembly* **K** e **G** (s)	*Assembly* **C** e $\mathbf{C_t}$ (s)	Solução (s)	Total (s)
–	1.418	–	–	1.418
1	–	2.299	0.767	3.066
2	–	2.323	0.691	3.014
3	–	2.317	0.700	3.017
4	–	2.306	0.694	3.000
5	–	2.323	0.707	3.030
6	–	2.322	0.716	3.038
	1.418	13.89	4.275	18.165
Paralelo				
–	0.27	–	–	0.27
1	–	0.215	0.531	0.746
2	–	0.211	0.456	0.667
3	–	0.211	0.458	0.669
4	–	0.214	0.459	0.673
5	–	0.215	0.460	0.675
6	–	0.216	0.450	0.666
	0.27	1.282	2.814	4.096

Tabela 6.33 – Tempo total para o *assembly* das matrizes globais, solução do sistema de equações e simulação para uma malha com 20322 elementos.

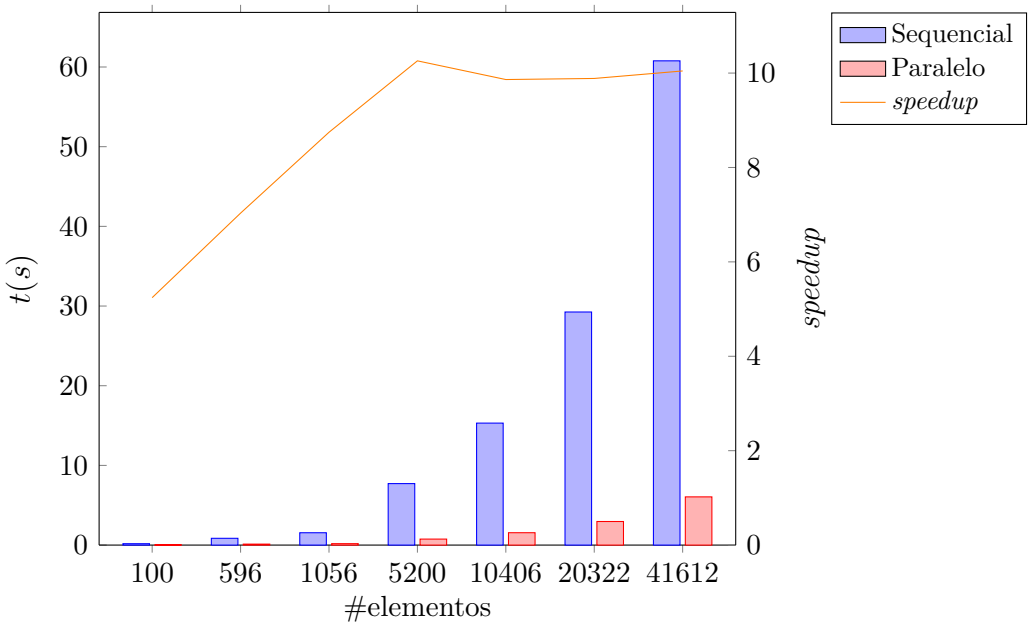
#Elementos = 20322				
Sequencial				
#Iteração	*Assembly* **K** e **G** (s)	*Assembly* **C** e $\mathbf{C_t}$ (s)	Solução (s)	Total (s)
–	2.739	–	–	2.739
1	–	4.386	1.587	5.973
2	–	4.429	1.441	5.87
3	–	4.433	1.485	5.918
4	–	4.410	1.495	5.905
5	–	4.428	1.458	5.886
6	–	4.427	1.492	5.919
	2.739	26.513	8.958	35.471
Paralelo				
–	0.482	–	–	0.482
1	–	0.430	1.055	1.485
2	–	0.407	1.002	1.409
3	–	0.418	0.946	1.364
4	–	0.395	0.959	1.354
5	–	0.414	0.985	1.399
6	–	0.413	0.963	1.376
	0.482	2.477	5.910	8.387

Tabela 6.34 – Tempo total para o *assembly* das matrizes globais, solução do sistema de equações e simulação para uma malha com 41612 elementos.

#Elementos = 41612				
Sequencial				
#Iteração	*Assembly* **K** e **G** (s)	*Assembly* **C** e $\mathbf{C_t}$ (s)	Solução (s)	Total (s)
–	5.696	–	–	5.696
1	–	9.170	3.453	12.623
2	–	9.217	3.404	12.621
3	–	9.174	3.294	12.468
4	–	9.197	3.413	12.610
5	–	9.153	3.366	12.519
6	–	9.166	3.363	12.529
	5.696	55.077	20.293	75.370
Paralelo				
–	1.02	–	–	1.02
1	–	0.852	2.178	3.03
2	–	0.833	2.138	2.971
3	–	0.832	2.079	2.911
4	–	0.846	2.175	3.021
5	–	0.827	2.051	2.878
6	–	0.840	2.042	2.882
	1.020	5.030	12.663	17.693

A Fig. 6.2 mostra o comparativo entre o processamento sequencial e paralelo para o tempo total do *assembly* das matrizes globais e o *speedup* alcançado.

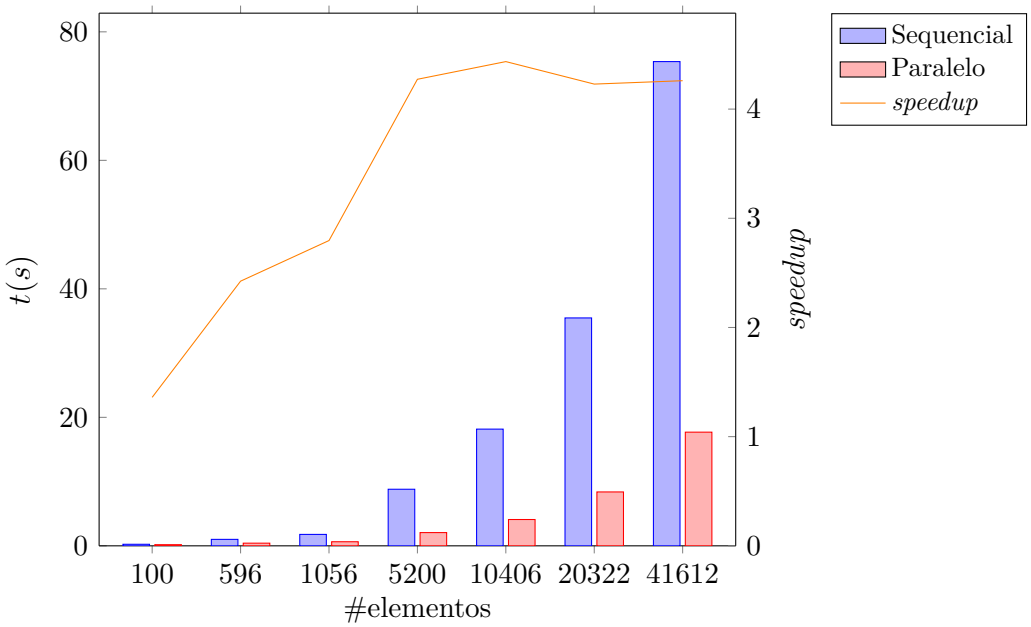
Figura 6.27 – Comparativo do tempo total para o *assembly* das matrizes globais e o *speedup*.



Fonte: O autor

A Fig. 6.28 mostra o comparativo entre o processamento sequencial e paralelo para o tempo total da simulação e o *speedup* alcançado.

Figura 6.28 – Comparativo do tempo total para o *assembly* das matrizes globais e o *speedup*.



Fonte: O autor

Os resultados comparativos mostram uma considerável diminuição no tempo de processamento, com um *speedup* de aproximadamente 10 para o *assembly* das matrizes globais para a malha de elementos finitos mais refinada. A curva do *speedup*, como pode ser observado na Fig. 6.2, apresenta uma diminuição em sua inclinação e estabilização acima de 5000 elementos. Ou seja, acima deste valor, aumentando-se o número de elementos não obter-se-á um aumento significativo de *speedup*, como acontece para as malhas menos refinadas.

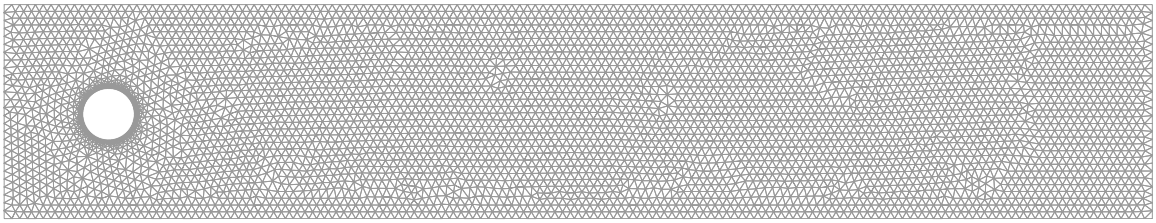
O tempo total para solução do problema mostra um *speedup* de aproximadamente 4.5 para a malha de elementos finitos mais refinada. Isto se deve ao fato de que a solução do sistema de equações lineares é realizada de forma sequencial utilizando o pacote PARDISO (KUZMIN et al., 2013). Acima de 10000 elementos a curva do *speedup* apresenta leve uma diminuição em sua inclinação. Ou seja, a mesma observação anterior – aumento do número de elementos não obtém-se um aumento do *speedup* – pode ser feita no tempo total para solução do problema.

Pode-se observar que em uma malha pouco refinada não há vantagem em se utilizar o processamento paralelo ao invés do sequencial (*speedup* de aproximadamente 1.5). A justificativa para tal fenômeno está relacionado com o tempo necessário para que a GPU inicie o processo de paralelização (inicialização dos blocos de *threads*, das *threads* e outros processos de inicializações internos da placa gráfica).

6.2.2 Resultados

Para o exemplo com uma malha conforme em um problema de dinâmica dos fluidos foi utilizada uma malha com 10138 elementos do tipo P2P1 e 20834 nós e é apresentada na Fig. 6.29.

Figura 6.29 – Malha com 10138 elementos finitos do P2P1 e 20834 nós.



Fonte: O autor

6.2.2.1 Caso estacionário

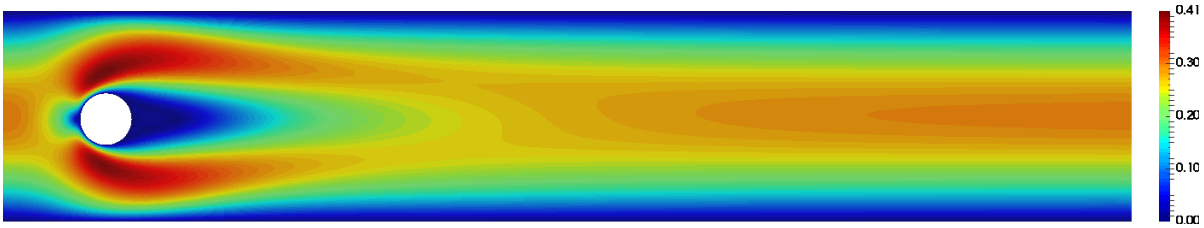
A velocidade na entrada do canal é dada por:

$$U\left(0,y\right)=4U_m y\left(H-y\right)/H^2,\ V=0$$

com $U_m = 0,3m/s$. O número de Reynolds, para esta velocidade e configuração de canal é dado por $Re = 20$.

A Fig. 6.30 apresenta os resultados do campo de velocidade em m/s e pressão em Pa para esta configuração.

Figura 6.30 – Campo de velocidades (a) e de pressão (b) para $Re = 20$.



(a)

(b)

Fonte: O autor

Os resultados quantitativos são apresentados na Tab. 6.35.

Tabela 6.35 – Comparação dos coeficientes de sustentação e arrasto para o caso estacionário.

	c_L	c_D
Presente trabalho	0.0104	5.5837
Gomes (2013)	0.0102	5.5921
Bänsch, E. *et al.* (1996)	0.0110	5.5760

6.2.2.2 Caso transiente

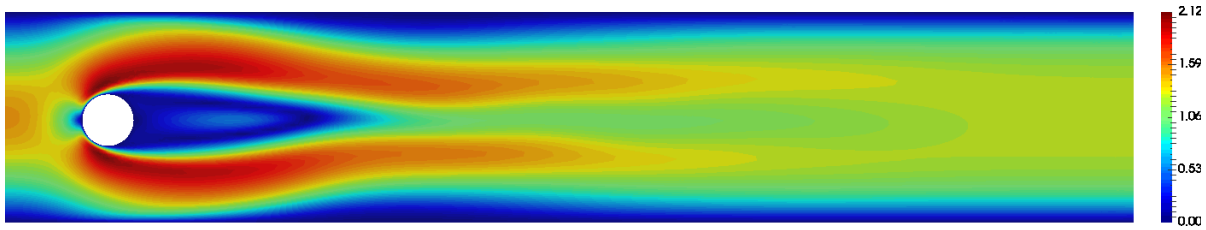
A velocidade na entrada do canal é dada por:

$$U\left(0,y,t\right)=4U_m y\left(H-y\right)/H^2,\ V=0$$

com $U_m = 1,5m/s$. O número de Reynolds, para esta velocidade e configuração de canal é dado por $Re = 100$.

As Fig. 6.31, 6.32, 6.33, 6.34 apresentam os resultados do campo de velocidades em m/s e pressão em Pa para diferentes instantes de tempo.

Figura 6.31 – Campo de velocidades (a) e pressão (b) para $t = 2,5s$.

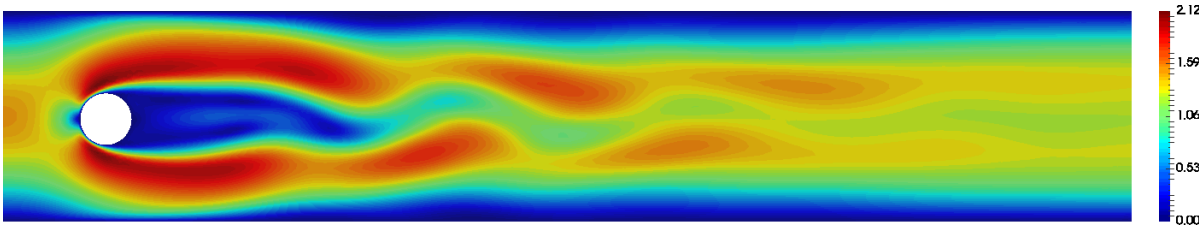


(a)

(b)

Fonte: O autor

Figura 6.32 – Campo de velocidades (a) e pressão (b) para $t = 3,5s$.

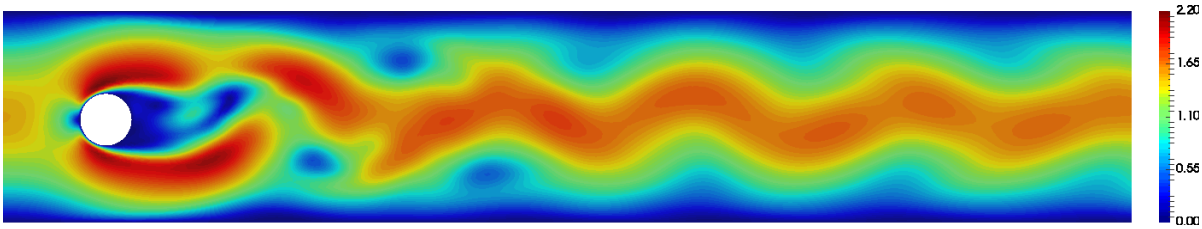


(a)

(b)

Fonte: O autor

Figura 6.33 – Campo de velocidades (a) e pressão (b) para $t = 6,0s$.

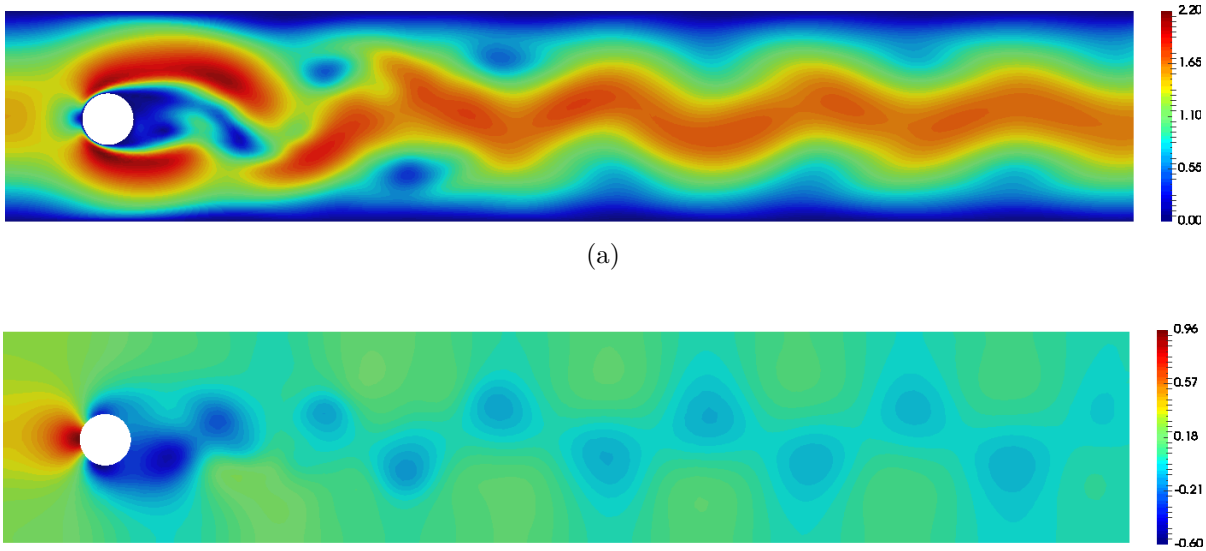


(a)

(b)

Fonte: O autor

Figura 6.34 – Campo de velocidades (a) e pressão (b) para $t = 8,0s$.



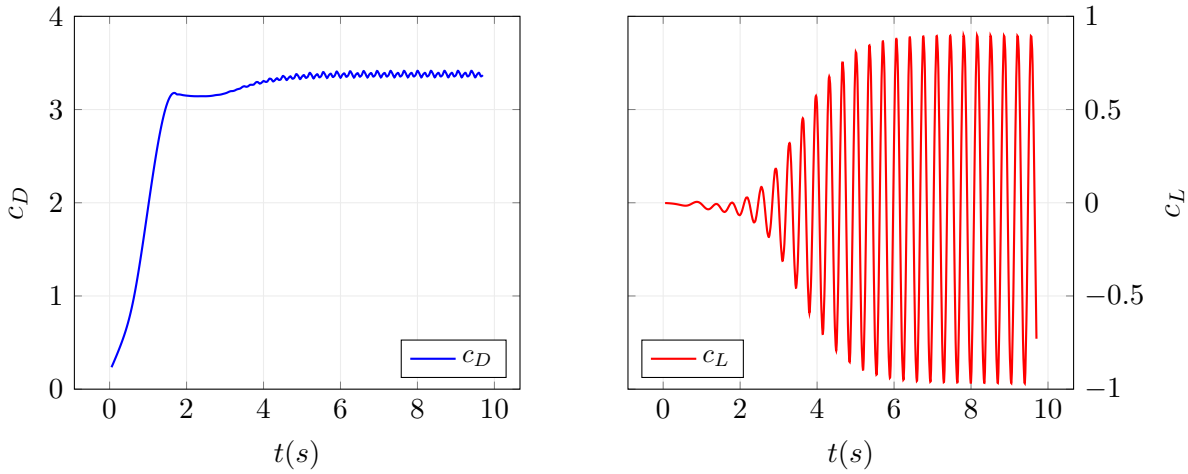
(a)

(b)

Fonte: O autor

A Fig. 6.35 apresenta os coeficientes de arrasto e sustentação para o caso transiente no tempo.

Figura 6.35 – Coeficientes de arrasto e de sustentação no tempo.



Fonte: O autor

Os resultados quantitativos são apresentados na Tab. 6.36.

Tabela 6.36 – Comparação dos coeficientes de sustentação e arrasto máximos para o caso transiente.

	c_{Lmax}	c_{Dmax}
Presente trabalho	0.9635	3.4153
Gomes (2013)	1.0369	3.2450
Bänsch, E. *et al.* (1996)	1.0060	3.2240

6.2.3 Análise de desempenho

6.2.3.1 Caso estacionário

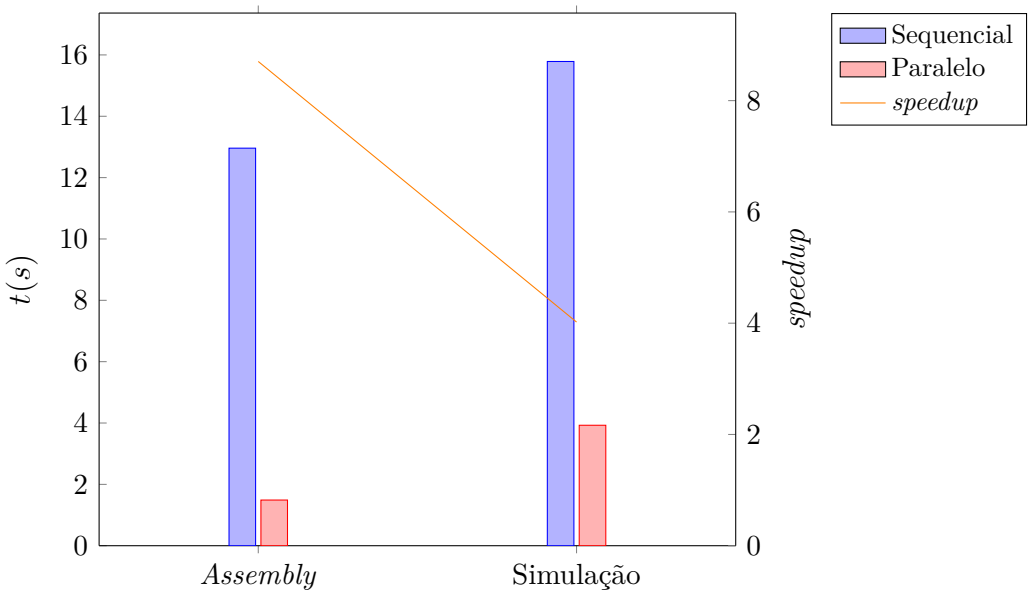
As Tab. 6.37 apresenta o tempo total para o *assembly* das matrizes globais, solução do sistema de equações e simulação para uma malha com 10138 elementos do tipo P2P1 tanto para o processamento sequencial quanto para o paralelizado na GPU para o caso estacionário.

Tabela 6.37 – Tempo total para o *assembly* das matrizes globais, solução do sistema de equações e simulação para uma malha com 10138 elementos.

#Elementos = 10138				
Sequencial				
#Iteração	*Assembly* $\mathbf{K}$ e $\mathbf{G}$ (s)	*Assembly* $\mathbf{C}$ e $\mathbf{C_t}$ (s)	Solução (s)	Total (s)
–	1.219	–	–	1.219
1	–	1.946	0.747	2.693
2	–	1.958	0.653	2.611
3	–	1.959	0.658	2.617
4	–	1.959	0.663	2.622
5	–	1.958	0.662	2.620
6	–	1.961	0.662	2.623
	1.219	11.741	4.045	15.786
Paralelo				
–	0.261	–	–	0.261
1	–	0.224	0.544	0.768
2	–	0.200	0.428	0.628
3	–	0.202	0.429	0.631
4	–	0.201	0.441	0.642
5	–	0.200	0.434	0.634
6	–	0.201	0.423	0.624
	0.261	1.228	2.699	3.927

A Fig. 6.36 mostra o comparativo entre o processamento sequencial e paralelo para o tempo total do *assembly* e da simulação e o *speedup* alcançado.

Figura 6.36 – Comparativo do tempo total do *assembly* das matrizes globais e da simulação e o *speedup* alcançado.



Fonte: O autor

Os resultados comparativos mostram uma considerável diminuição no tempo de processamento, com um *speedup* de aproximadamente 9 para o *assembly* das matrizes globais e de aproximadamente 4 para o tempo total da simulação. Estes valores vão de encontro com a análise de sensibilidade, apesar de apresentar *speedups* abaixo do esperado. Isto se deve ao fato de a malha possuir um maior refinamento próximo ao cilindro, aumentando assim, o tempo de processamento.

6.2.3.2 Caso transiente

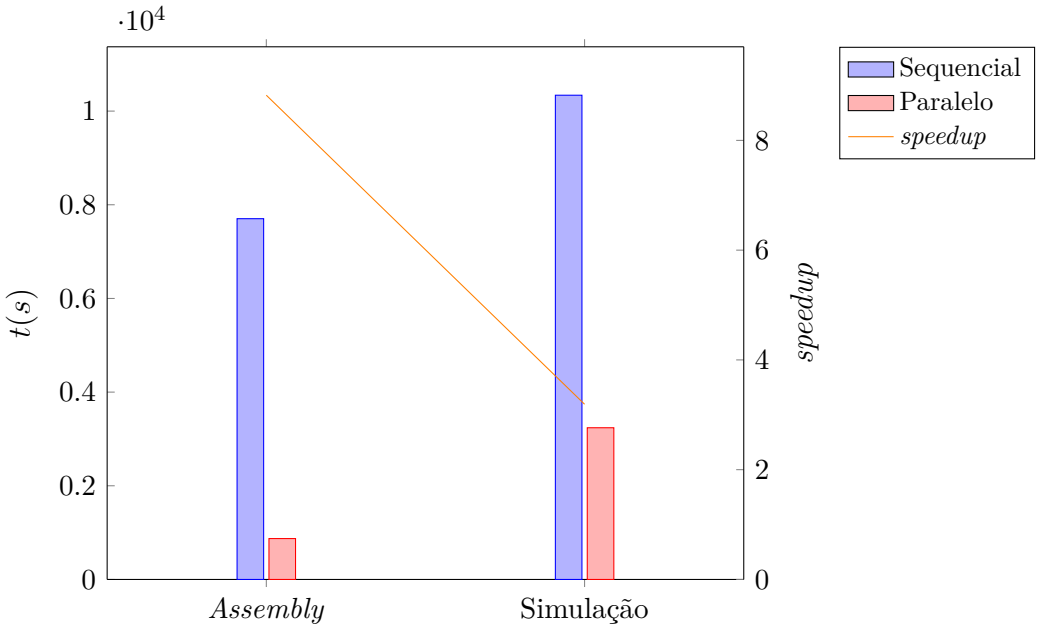
As Tab. 6.38 apresenta o tempo total para o *assembly* das matrizes globais, solução do sistema de equações e simulação para uma malha com 10138 elementos do tipo P2P1 tanto para o processamento sequencial quanto para o paralelizado na GPU para o caso transiente.

Tabela 6.38 – Tempo total para o *assembly* das matrizes globais, solução do sistema de equações e simulação para uma malha com 10138 elementos.

#Elementos = 10138				
Sequencial				
#Time Step	*Assembly* **K**, **G** e **M** (s)	*Assembly* **C** e $\mathbf{C_t}$ (s)	Solução (s)	Total (s)
–	1.987	–	–	1.987
1 – 1001	–	7700.883	2636.213	10337.096
	1.987	7700.883	2636.213	10339.083
Paralelo				
–	0.300	–	–	0.300
1 – 1001	–	872.765	2366.080	3238.845
	0.300	872.765	2366.080	3239.145

A Fig. 6.37 mostra o comparativo entre o processamento sequencial e paralelo para o tempo total do *assembly* e da simulação e o *speedup* alcançado.

Figura 6.37 – Comparativo do tempo total para o *assembly* das matrizes globais e da simulação e o *speedup* alcançado.



Fonte: O autor

Os resultados comparativos mostram uma considerável diminuição no tempo de processamento, com um *speedup* de aproximadamente 9 para o *assembly* das matrizes globais e de aproximadamente 3 para o tempo total da simulação. Estes valores vão de encontro com a análise de sensibilidade, apesar de apresentar *speedups* abaixo do esperado. Isto se deve ao fato de a malha possuir um maior refinamento próximo ao cilindro, aumentando assim, o tempo de processamento.

6.2.4 Interação fluido-estrutura pelo método das Fronteiras Imersas

Neste exemplo utiliza-se uma malha de fluido fixa e um cilindro é posicionado sobre esta malha. Da mesma forma que o exemplo anterior, este é um exemplo clássico da literatura como um caso de validação para códigos de interação fluido-estrutura.

6.2.4.1 Análise de sensibilidade

Para a análise de sensibilidade desta simulação foram utilizadas malhas com número crescente de elementos finitos do tipo P2P1. As Tab. 6.39, 6.40, 6.41, 6.42, 6.43, 6.44 e 6.45 apresentam o tempo total para o *assembly* das matrizes globais, solução do sistema de equações e simulação para uma malha com 108, 518, 1040, 5104, 10816, 20856 e 41292 elementos tanto para o processamento sequencial quanto para o paralelizado na GPU.

Tabela 6.39 – Tempo total para o *assembly* das matrizes globais, solução do sistema de equações e simulação para uma malha com 108 elementos.

#Elementos = 108				
Sequencial				
#Iteração	*Assembly* **K** e **G** (s)	*Assembly* **C** e $\mathbf{C_t}$ (s)	Solução (s)	Total (s)
–	0.034	–	–	0.034
1	–	0.031	0.115	0.146
2	–	0.019	0.015	0.034
3	–	0.037	0	0.037
4	–	0.037	0.015	0.052
5	–	0.032	0.015	0.047
	0.034	0.156	0.16	0.35
Paralelo				
–	0.037	–	–	0.037
1	–	0.031	0.1	0.131
2	–	0.02	0.015	0.035
3	–	0	0.022	0.022
4	–	0	0.022	0.022
5	–	0.006	0.022	0.028
	0.037	0.057	0.181	0.275

Tabela 6.40 – Tempo total para o *assembly* das matrizes globais, solução do sistema de equações e simulação para uma malha com 518 elementos.

#Elementos = 518				
Sequencial				
#Iteração	*Assembly* **K** e **G** (s)	*Assembly* **C** e $\mathbf{C_t}$ (s)	Solução (s)	Total (s)
–	0.083	–	–	0.083
1	–	0.115	0.116	0.231
2	–	0.115	0.021	0.136
3	–	0.115	0.021	0.136
4	–	0.115	0.037	0.152
5	–	0.115	0.021	0.136
6	0.083	0.116	0.036	0.152
	0.083	0.691	0.252	1.026
Paralelo				
–	0.046	–	–	0.046
1	–	0.037	0.131	0.168
2	–	0.02	0.046	0.066
3	–	0.03	0.039	0.069
4	–	0.019	0.032	0.051
5	–	0.019	0.033	0.052
6	–	0.019	0.039	0.058
	0.046	0.144	0.32	0.51

Tabela 6.41 – Tempo total para o *assembly* das matrizes globais, solução do sistema de equações e simulação para uma malha com 1040 elementos.

#Elementos = 1040				
Sequencial				
#Iteração	*Assembly* **K** e **G** (s)	*Assembly* **C** e $\mathbf{C_t}$ (s)	Solução (s)	Total (s)
–	0.146	–	–	0.146
1	–	0.232	0.147	0.379
2	–	0.238	0.064	0.302
3	–	0.232	0.053	0.285
4	–	0.238	0.062	0.3
5	–	0.231	0.053	0.284
6	–	0.238	0.062	0.3
	0.146	1.409	0.441	1.996
Paralelo				
–	0.037	–	–	0.037
1	–	0.046	0.132	0.178
2	–	0.037	0.079	0.116
3	–	0.031	0.075	0.106
4	–	0.032	0.08	0.112
5	–	0.031	0.085	0.116
6	–	0.031	0.046	0.077
	0.037	0.208	0.497	0.742

Tabela 6.42 – Tempo total para o *assembly* das matrizes globais, solução do sistema de equações e simulação para uma malha com 5104 elementos.

#Elementos = 5104				
Sequencial				
#Iteração	*Assembly* **K** e **G** (s)	*Assembly* **C** e $\mathbf{C_t}$ (s)	Solução (s)	Total (s)
–	0.808	–	–	0.808
1	–	1.346	0.385	1.731
2	–	1.366	0.307	1.673
3	–	1.36	0.316	1.676
4	–	1.35	0.316	1.666
5	–	1.356	0.32	1.676
6	–	1.362	0.316	1.678
	0.808	8.14	1.96	10.908
Paralelo				
–	0.196	–	–	0.196
1	–	0.137	0.345	0.482
2	–	0.132	0.225	0.357
3	–	0.132	0.253	0.385
4	–	0.115	0.231	0.346
5	–	0.132	0.249	0.381
6	–	0.132	0.235	0.367
	0.196	0.78	1.538	2.514

Tabela 6.43 – Tempo total para o *assembly* das matrizes globais, solução do sistema de equações e simulação para uma malha com 10816 elementos.

#Elementos = 10816				
Sequencial				
#Iteração	*Assembly* **K** e **G** (s)	*Assembly* **C** e $\mathbf{C_t}$ (s)	Solução (s)	Total (s)
–	1.509	–	–	1.509
1	–	2.376	0.794	3.17
2	–	2.385	0.679	3.064
3	–	2.406	0.686	3.092
4	–	2.386	0.688	3.074
5	–	2.403	0.716	3.119
6	–	2.396	0.696	3.092
	1.509	14.352	4.259	20.12
Paralelo				
–	0.358	–	–	0.358
1	–	0.3	0.536	0.836
2	–	0.269	0.434	0.703
3	–	0.271	0.458	0.729
4	–	0.268	0.451	0.719
5	–	0.275	0.454	0.729
6	–	0.273	0.461	0.734
	0.358	1.656	2.794	4.808

Tabela 6.44 – Tempo total para o *assembly* das matrizes globais, solução do sistema de equações e simulação para uma malha com 20856 elementos.

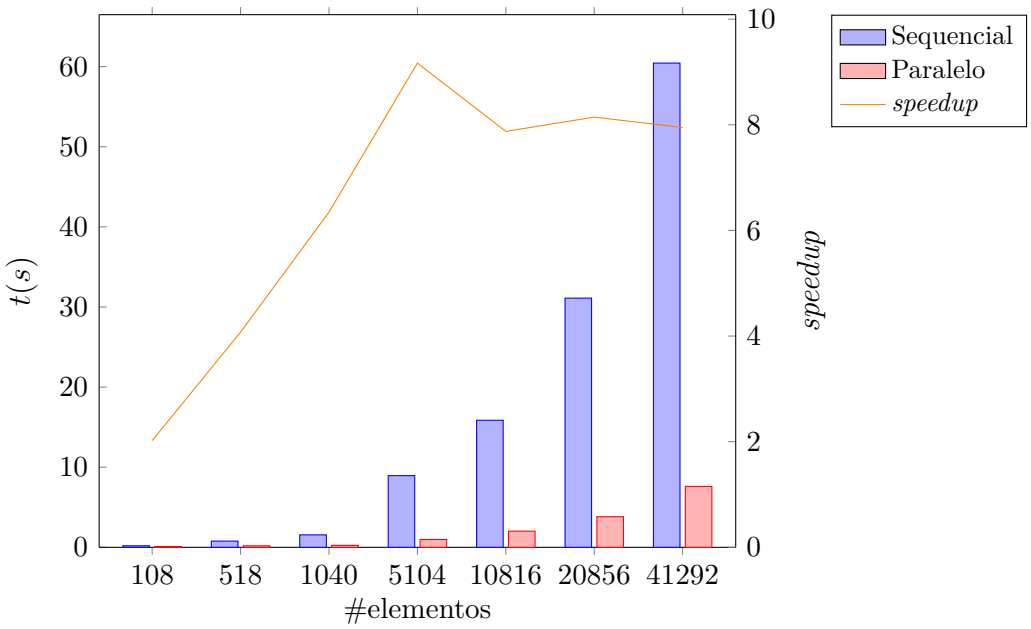
#Elementos = 20856				
Sequencial				
#Iteração	*Assembly* **K** e **G** (s)	*Assembly* **C** e $\mathbf{C_t}$ (s)	Solução (s)	Total (s)
–	2.937	–	–	2.937
1	–	4.666	1.601	6.267
2	–	4.707	1.438	6.145
3	–	4.681	1.448	6.129
4	–	4.682	1.472	6.154
5	–	4.719	1.702	6.421
6	–	4.713	1.695	6.408
	2.937	28.168	9.356	40.461
Paralelo				
–	0.657	–	–	0.657
1	–	0.545	0.953	1.498
2	–	0.519	0.94	1.459
3	–	0.512	0.877	1.389
4	–	0.525	0.872	1.397
5	–	0.535	0.982	1.517
6	–	0.526	0.973	1.499
	0.657	3.162	5.597	9.416

Tabela 6.45 – Tempo total para o *assembly* das matrizes globais, solução do sistema de equações e simulação para uma malha com 41292 elementos.

#Elementos = 41292				
Sequencial				
#Iteração	*Assembly* **K** e **G** (s)	*Assembly* **C** e $\mathbf{C_t}$ (s)	Solução (s)	Total (s)
–	5.708	–	–	5.708
1	–	9.077	3.65	12.727
2	–	9.146	3.848	12.994
3	–	9.138	3.561	12.699
4	–	9.142	3.659	12.801
5	–	9.101	3.635	12.736
6	–	9.13	3.653	12.783
	5.708	54.734	22.006	82.448
Paralelo				
–	1.26	–	–	1.26
1	–	1.051	2.271	3.322
2	–	1.039	2.314	3.353
3	–	1.059	2.284	3.343
4	–	1.076	2.698	3.774
5	–	1.039	2.308	3.347
6	–	1.08	2.268	3.348
	1.26	6.344	14.143	21.747

A Fig. 6.38 mostra o comparativo entre o processamento sequencial e paralelo para o tempo total do *assembly* das matrizes globais e o *speedup* alcançado.

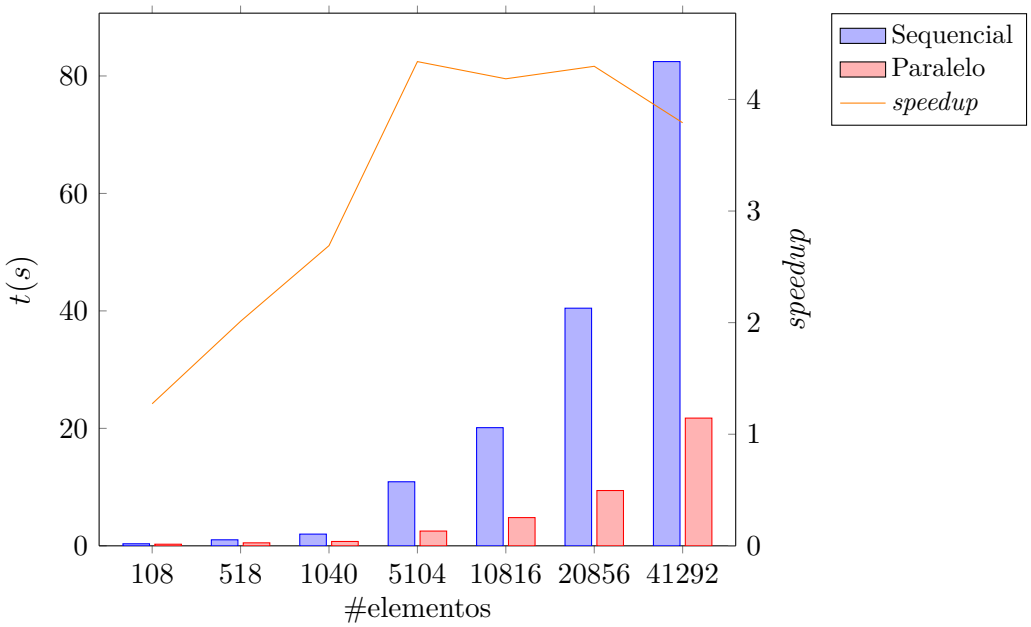
Figura 6.38 – Comparativo do tempo total para o *assembly* das matrizes globais e o *speedup* alcançado.



Fonte: O autor

A Fig. 6.39 mostra o comparativo entre o processamento sequencial e paralelo para o tempo total da simulação e o *speedup* alcançado.

Figura 6.39 – Comparativo do tempo total da simulação e o *speedup* alcançado.



Fonte: O autor

Os resultados comparativos mostram uma considerável diminuição no tempo de processamento, com um *speedup* de aproximadamente 9 para o *assembly* das matrizes globais para a malha com 5104 elementos finitos. A curva do *speedup*, como pode ser observado na Fig. 6.2, apresenta uma diminuição em sua inclinação e estabilização acima de 5000 elementos. Ou seja, acima deste valor, aumentando-se o número de elementos não obter-se-á um aumento significativo de *speedup*, como acontece para as malhas menos refinadas.

O tempo total para solução do problema mostra um *speedup* de aproximadamente 4.5 para a malha com 5104 elementos finitos. Isto se deve ao fato de que a solução do sistema de equações lineares é realizada de forma sequencial utilizando o pacote PARDISO (KUZMIN et al., 2013). A mesma observação anterior – aumento do número de elementos acima de 5000 elementos não obtém-se um considerável aumento de *speedup* – pode ser feita para o comparativo do tempo do total para solução do problema, inclusive, a partir de 208000 elementos, há uma diminuição do *speedup*.

Pode-se observar que em uma malha pouco refinada não há vantagem em se utilizar o processamento paralelo ao invés do sequencial (*speedup* de aproximadamente 1.5). A justificativa para tal fenômeno está relacionado com o tempo necessário para que a GPU inicie o processo de paralelização (inicialização dos blocos de *threads*, das *threads* e outros processos de inicializações internos da placa gráfica).

6.2.4.2 Resultados

Uma malha pra o fluido com 42176 elementos do tipo P2P1 e 84973 nós foram utilizadas e são apresentadas na Fig. 6.40.

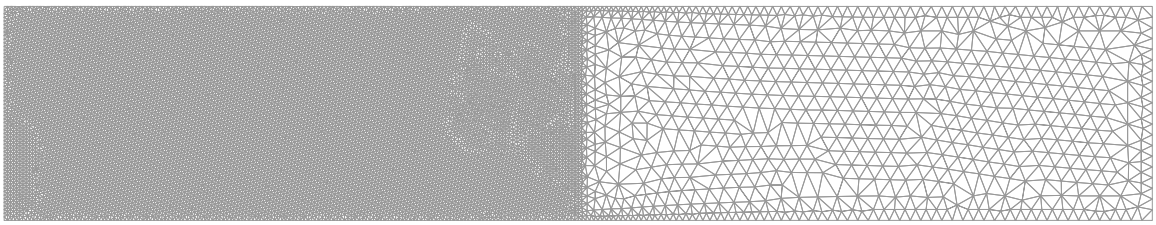


Figura 6.40 – Malha com 42176 elementos finitos do P2P1 e 84973 nós.

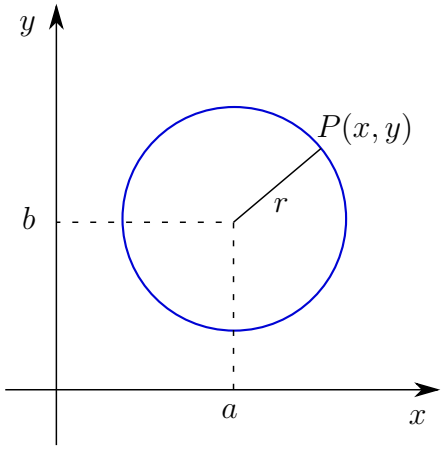
6.2.4.2.1 Solução analítica – Circunferência

Existem duas formas que podem ser utilizadas para gerar o domínio da estrutura no formato de uma circunferência. Gera-se ou uma malha de elementos finitos ou utiliza-se uma equação analítica. A forma utilizada nesta simulação foi por meio de uma equação analítica.

6.2.4.2.2 Equação da circunferência

Suponha-se uma circunferência com raio r posicionado a uma distância a na horizontal e b na vertical da origem conforme Fig. 6.41.

Figura 6.41 – Circunferência.



Fonte: O autor

A equação da circunferência é:

$$(x-a)^2 + (y-b)^2 = r^2 \tag{6.4}$$

onde x e y são as coordenadas dos pontos sobre a circunferência.

6.2.4.2.3 Intersecção entre a circunferência e a malha de fluido

É necessário encontrar os nós de intersecção entre a circunferência e a malha de fluido. Estes nós de intersecção definem a interface da superfície molhada. Para que se possa encontrar estes nós de intersecção, utiliza-se a seguinte equação:

$$x_{int} = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a} \tag{6.5}$$

onde

$$a = m^2 + 1 \tag{6.6}$$

$$b = 2\left(mc - mb - a\right) \tag{6.7}$$

$$c = a^2 + b^2 + c^2 - 2bc - r^2 \tag{6.8}$$

Substituindo na equação de uma reta:

$$y = mx + c \tag{6.9}$$

onde:

- m: Coeficiente angular da reta;
- c: Coeficiente linear da reta.

sendo assim possível encontrar os nós de intersecção entre a circunferência e a malha de fluido.

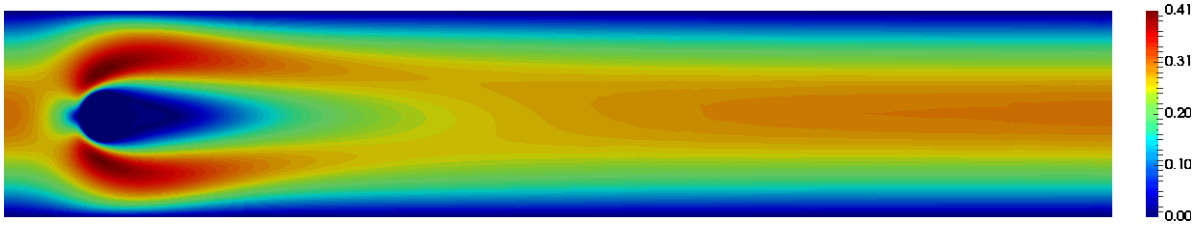
6.2.4.3 Caso estacionário

A velocidade na entrada do canal é igual ao exemplo anterior e é dada por:

$$U\left(0, y\right) = 4U_m y\left(H - y\right)/H^2,\ V = 0$$

com $U_m = 0,3m/s$. O número de Reynolds, para esta velocidade e configuração de canal é dado por $Re = 20$.

A Fig. 6.42 apresenta os resultados do campo de velocidade em m/s e pressão em Pa para esta configuração.



(a)

(b)

Figura 6.42 – Campo de velocidades (a) e de pressão (b) para $Re = 20$.

Os resultados quantitativos são apresentados na Tab. 6.46.

Tabela 6.46 – Comparação dos coeficientes de sustentação e arrasto para o caso estacionário.

	c_L	c_D
Presente trabalho com 180 Multiplicadores de Lagrange	0.0117	5.5790
Gomes (2013) com 170 Multiplicadores de Lagrange	0.0106	5.5867
Bänsch, E. *et al.* (1996)	0.0110	5.5760

6.2.4.3.1 Análise de desempenho

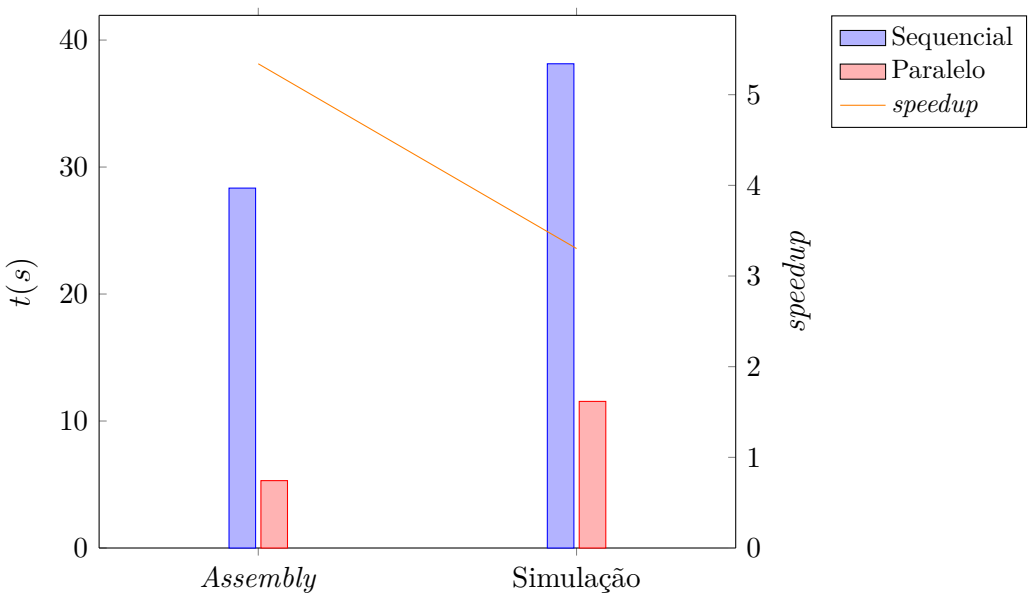
As Tab. 6.47 apresenta o tempo total para o *assembly* das matrizes globais, solução do sistema de equações e simulação para uma malha com 42176 elementos do tipo P2P1 tanto para o processamento sequencial quanto para o paralelizado na GPU.

Tabela 6.47 – Tempo total para o *assembly* das matrizes globais, solução do sistema de equações e simulação para uma malha com 1724 elementos.

#Elementos = 42176				
Sequencial				
#Iteração	*Assembly* **K**, **G** e **M** (s)	*Assembly* **C** e $\mathbf{C_t}$ (s)	Solução (s)	Total (s)
–	2.653	–	–	2.653
1	–	4.277	1.704	5.981
2	–	4.299	1.629	5.928
3	–	4.283	1.625	5.908
4	–	4.276	1.604	5.88
5	–	4.280	1.619	5.899
6	–	4.274	1.61	5.884
	2.653	25.689	9.791	38.133
Paralelo				
–	1.942	–	–	1.942
1	–	0.601	1.097	1.698
2	–	0.540	1.044	1.584
3	–	0.580	1.042	1.622
4	–	0.545	1.022	1.567
5	–	0.536	1.026	1.562
6	–	0.562	1.009	1.571
	1.942	3.364	6.24	11.546

A Fig. 6.43 mostra o comparativo entre o processamento sequencial e paralelo para o tempo total do *assembly* e da simulação e o *speedup* alcançado.

Figura 6.43 – Comparativo do tempo total para o *assembly* das matrizes globais e da simulação e o *speedup* alcançado.



Fonte: O autor

Os resultados comparativos mostram uma considerável diminuição no tempo de processamento, com um *speedup* de aproximadamente 5.5 para o *assembly* das matrizes globais e de aproximadamente 3.5 para o tempo total da simulação. Estes valores vão de encontro com a análise de sensibilidade, apesar de apresentar *speedups* abaixo do

esperado. Isto se deve ao fato de a malha possuir um maior refinamento próximo aos nós das extremidades da parede superior, aumentando assim, o tempo de processamento.

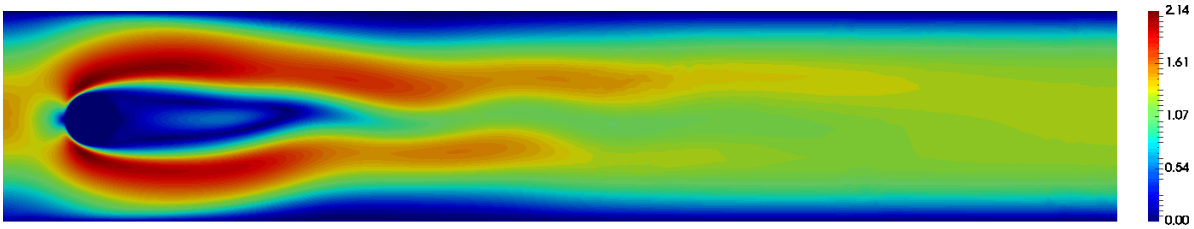
6.2.4.4 Caso transiente

A velocidade na entrada do canal é igual ao exemplo anterior e é dada por:

$$U\left(0,y,t\right)=4U_m y\left(H-y\right)/H^2,\ V=0$$

com $U_m = 1,5m/s$. O número de Reynolds, para esta velocidade e configuração de canal é dado por $Re = 100$.

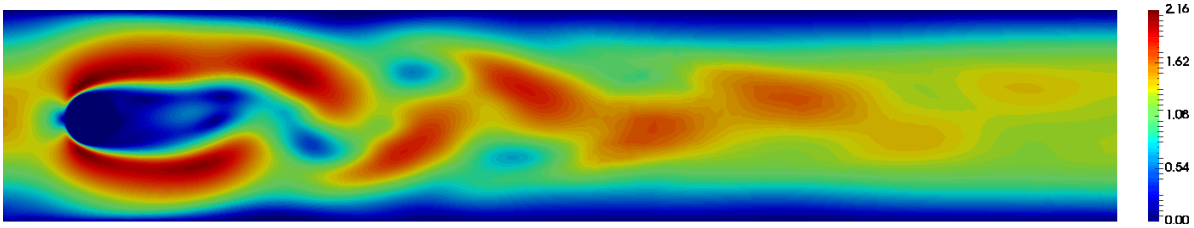
As Fig. 6.44, 6.45, 6.46, 6.47 apresentam os resultados do campo de velocidades em m/s e pressão em Pa para diferentes instantes de tempo.



(a)

(b)

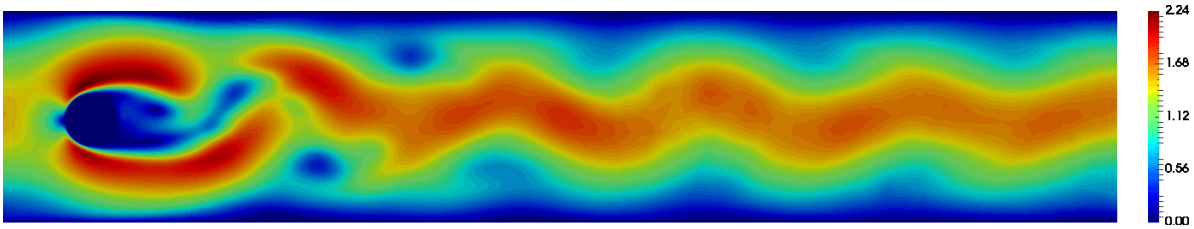
Figura 6.44 – Campo de velocidades (a) e pressão (b) para $t = 2,5s$.



(a)

(b)

Figura 6.45 – Campo de velocidades (a) e pressão (b) para $t = 3,5s$.



(a)

(b)

Figura 6.46 – Campo de velocidades (a) e pressão (b) para $t = 6,0s$.

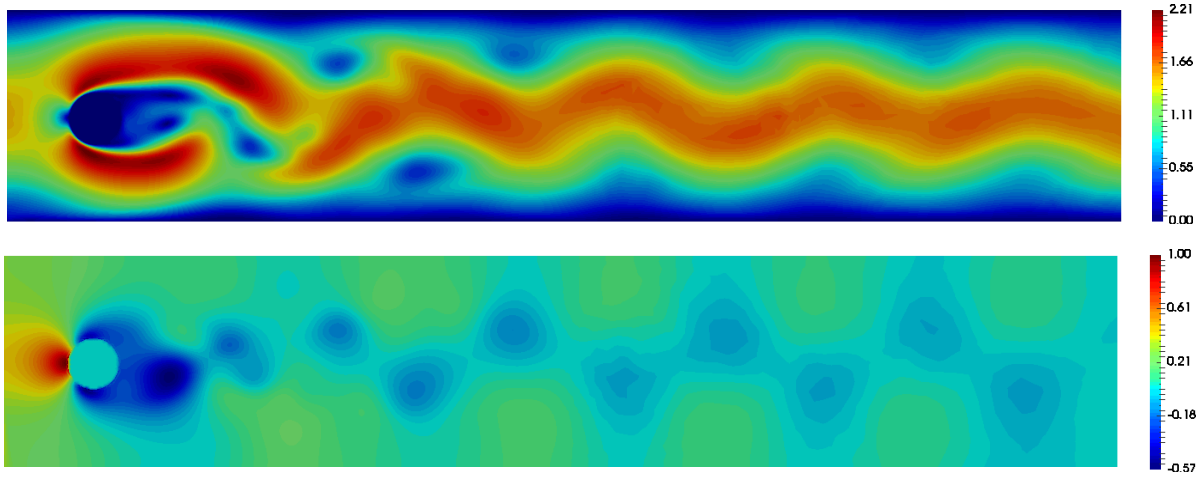
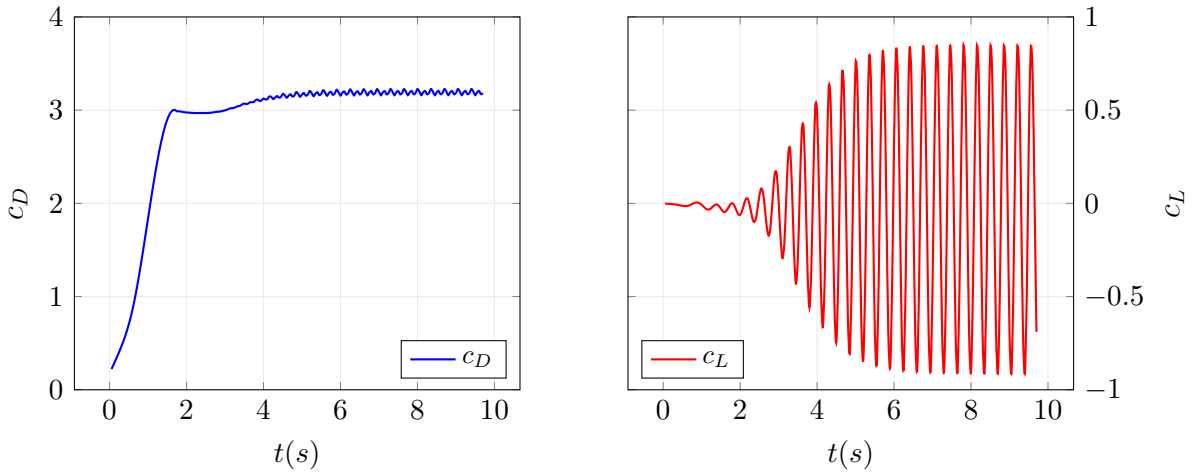


Figura 6.47 – Campo de velocidades (a) e pressão (b) para $t = 8,0s$.

A Fig. 6.48 apresenta os coeficientes de arrasto e sustentação para o caso transiente no tempo.

Figura 6.48 – Coeficientes de arrasto e de sustentação no tempo.



Fonte: O autor

Os resultados quantitativos são apresentados na Tab. 6.48.

Tabela 6.48 – Comparação dos coeficientes de sustentação e arrasto máximos para o caso transiente.

	c_{Lmax}	c_{Dmax}
Presente trabalho com 180 Multiplicadores de Lagrange	0.9103	3.2267
Gomes (2013) com 170 Multiplicadores de Lagrange	1.0369	3.2450
Bänsch, E. *et al.* (1996)	1.0060	3.2240

6.2.4.4.1 Análise de desempenho

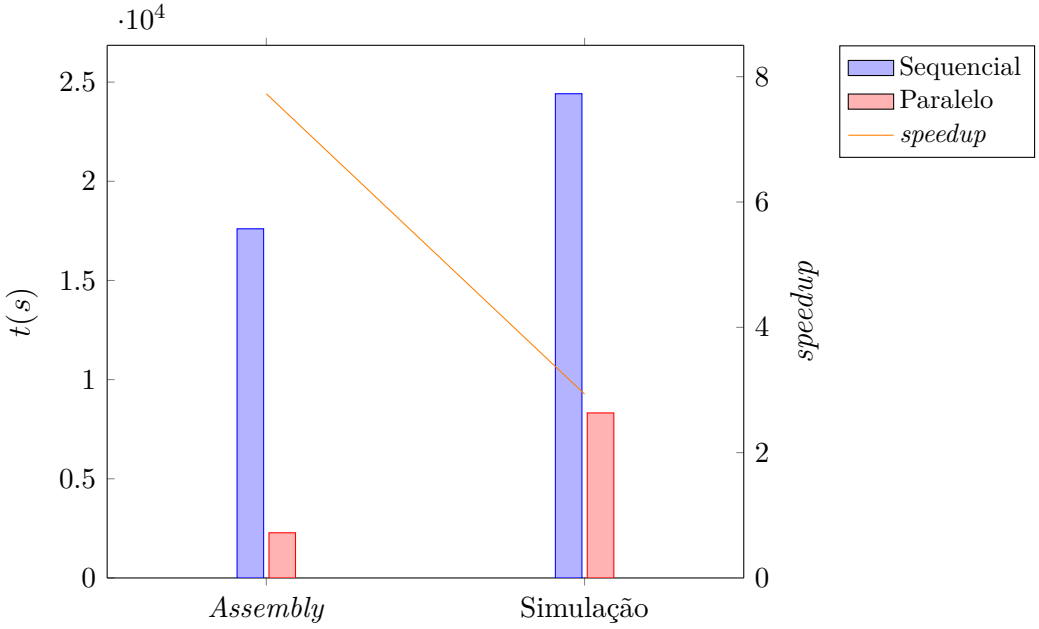
As Tab. 6.49 apresenta o tempo total para o *assembly* das matrizes globais, solução do sistema de equações e simulação para uma malha com 1724 elementos do tipo P2P1 tanto para o processamento sequencial quanto para o paralelizado na GPU.

Tabela 6.49 – Tempo total para o *assembly* das matrizes globais, solução do sistema de equações e simulação para uma malha com 1724 elementos.

#Elementos = 42176				
Sequencial				
#Time Step	*Assembly* **K**, **G** e **M** (s)	*Assembly* **C** e $\mathbf{C_t}$ (s)	Solução (s)	Total (s)
–	4.162	–	–	4.162
1-1001	–	17600.316	6809.306	24409.622
	4.162	17600.316	6809.306	24413.784
Paralelo				
–	0.786	–	–	0.786
1-1001	–	2277.106	6092.685	8319.791
	0.786	2277.106	6092.685	8320.577

A Fig. 6.49 mostra o comparativo entre o processamento sequencial e paralelo para o tempo total do *assembly* e da simulação e o *speedup* alcançado.

Figura 6.49 – Comparativo do tempo total para o *assembly* das matrizes globais e da simulação e o *speedup* alcançado.



Fonte: O autor

Os resultados comparativos mostram uma considerável diminuição no tempo de processamento, com um *speedup* de aproximadamente 8 para o *assembly* das matrizes globais e de aproximadamente 3 para o tempo total da simulação. Estes valores vão de encontro com a análise de sensibilidade, apesar de apresentar *speedups* abaixo do esperado. Isto se deve ao fato de a malha possuir um maior refinamento próximo aos nós das extremidades da parede superior, aumentando assim, o tempo de processamento.

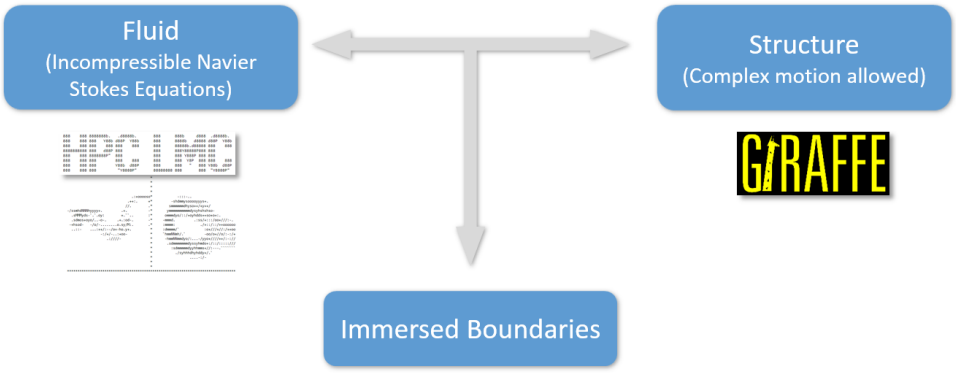
6.3 Co-simulação para análise de VIV (*Vortex Induced Vibration*) – HPC_LMC e GIRAFFE

Esta seção descreve a metodologia utilizada para investigar a hidrodinâmica de um *riser* e a interação solo-estrutura no contato do *riser* com o fundo do mar. As simulações foram realizadas utilizando dois softwares desenvolvidos no Laboratório de Mecânica Computacional da Universidade de São Paulo.

6.3.1 Metodologia

Para a solução da interação fluido-estrutura utilizou-se o *software* desenvolvido neste trabalho, aqui chamado de HPC_LMC (*High Performance Computing* do Laboratório de Mecânica Computacional da Universidade de São Paulo) e para a solução do *riser* utilizou-se um *software* chamado de GIRAFFE (*Generic Interface Readily Accessible for Finite Elements* (Gay Neto, 2016)). A Fig 6.50 mostra a interação entre os dois *softwares*.

Figura 6.50 – Interação entre os dois *softwares* – HPC_LMC e GIRAFFE.



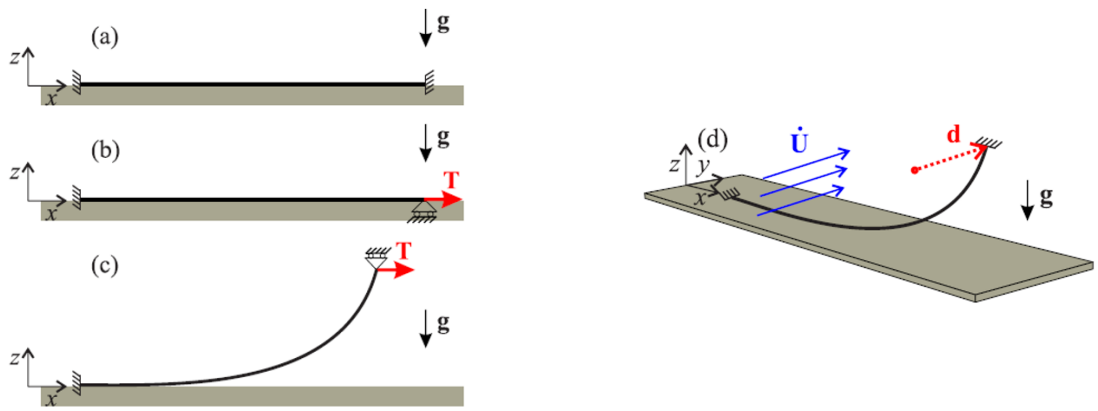
Fonte: O autor

Os passos necessários para que o problema de interação fluido-estrutura seja resolvido são descritos na Tab. 6.50.

Tabela 6.50 – Algoritmo da interação entre os dois *softares* – HPC_LMC e GIRAFFE – para resolução do problema de interação fluido-estrutura.

Algoritmo	
1:	Executa-se o GIRAFFE utilizando-se o arquivo neutro com as informações do *riser*;
2:	O GIRAFFE irá resolver o problema estático e fornecer a catenária do *riser* como ilustrado na Fig. 6.51;
3:	Executa-se o HPC_LMC utilizando-se o arquivo neutro com as informações do fluido. O HPC_LMC irá resolver o problema transiente até que a velocidade do fluido estabilize-se[3];
4:	Para cada passo de tempo:
4.1:	De forma a simular o problema tridimensional, utilizou-se o código de interação fluido-estrutura bidimensional[4], executando de forma paralelizada, diversos planos horizontais pré-definidos. Cada plano é responsável por encontrar as forças geradas sobre o *riser* em uma profundidade específica e posteriormente enviar estes dados para o GIRAFFE. Executa-se o HPC_LMC até que todos os planos tenham a solução do problema com a convergência esperada;
4.2:	Executa-se o GIRAFFE para resolver a estrutura do *riser* com as forças nodais geradas em cada plano – a partir do passo anterior – até que tenha a solução do problema com a convergência esperada;
4.3:	Incrementa-se o passo de tempo com o intervalo definido no arquivo neutro e reexecuta-se o passo 4.
5:	A configuração final após um certo tempo é apresentado na Fig. 6.51.

Figura 6.51 – Passos utilizados para resolver o problema de interação fluido-estrutura (a), (b) e (c) de forma a obter a catenária do *riser* e (d) a configuração final após a imposição da corrente do mar (Gay Neto et al., 2013).



Fonte: O autor

6.3.2 Simulação numérica

As seguintes propriedades forma utilizadas para a simulação numérica:

[3]A velocidade do fluido inicia-se em zero e cresce segundo a equação: $U(t) = 0.5(1 - cos(0.5\pi t))$ (de forma a obter uma velocidade de entrada suave e não gerar picos de pressão) até a velocidade definida no arquivo neutro.

[4]Desta forma pode-se reduzir o número de variáveis do problema, consequentemente reduzindo o tempo de processamento.

Tabela 6.51 – Propriedades físicas do *riser* e do mar (Gay Neto et al., 2013).

Comprimento do *riser*	5000.00	m
Rigidez axial	6.08	GN
Rigidez a flexão	110.80	$MN \cdot m^2$
Rigidez a torção	85.20	$MN \cdot m^2$
Rigidez ao cisalhamento	2.34	GN
Profundidade do mar	1709.00	m
Densidade da água do mar	1000.00	kg/m^3
Velocidade da corrente do mar	0.50	m/s

As propriedades do fluido e os parâmetros utilizados na simulação são:

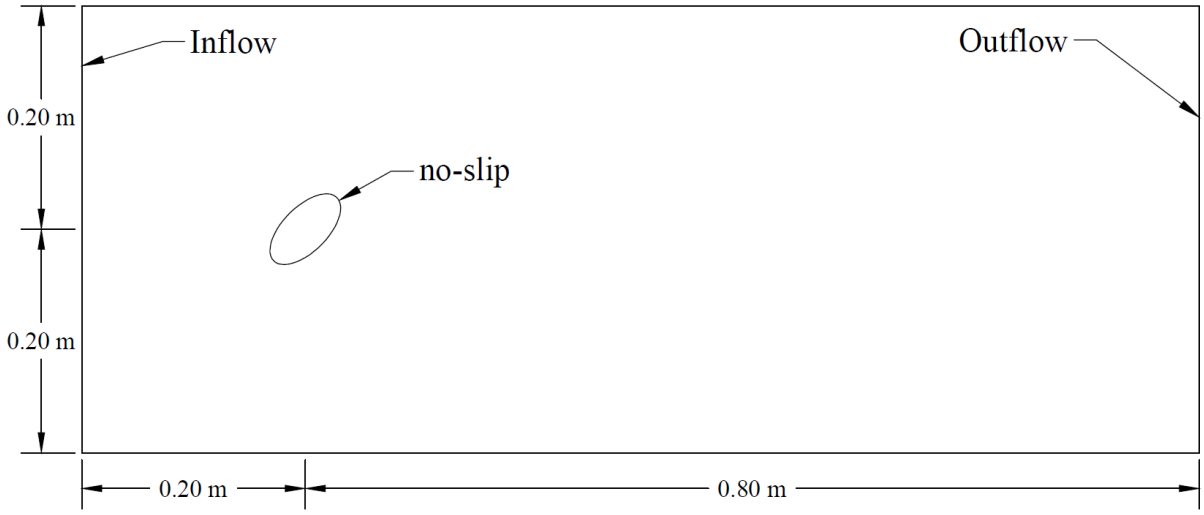
Tabela 6.52 – Propriedades do fluido e os parâmetros utilizados na simulação.

Densidade	1.00	kg/m^3
Viscosidade dinâmica	2.00×10^{-4}	$Pa \cdot s$
Velocidade na entrada	0.50	m/s
Reynolds	100	
Passo de tempo	0.05	s
Tempo máximo de simulação	10.00	s

6.3.3 Geometria

A geometria utilizada na simulação é apresentada na Fig. 6.52

Figura 6.52 – Geometria de um plano horizontal qualquer.



Fonte: O autor

Os parâmetros de discretização é apresentado na Tab. 6.53.

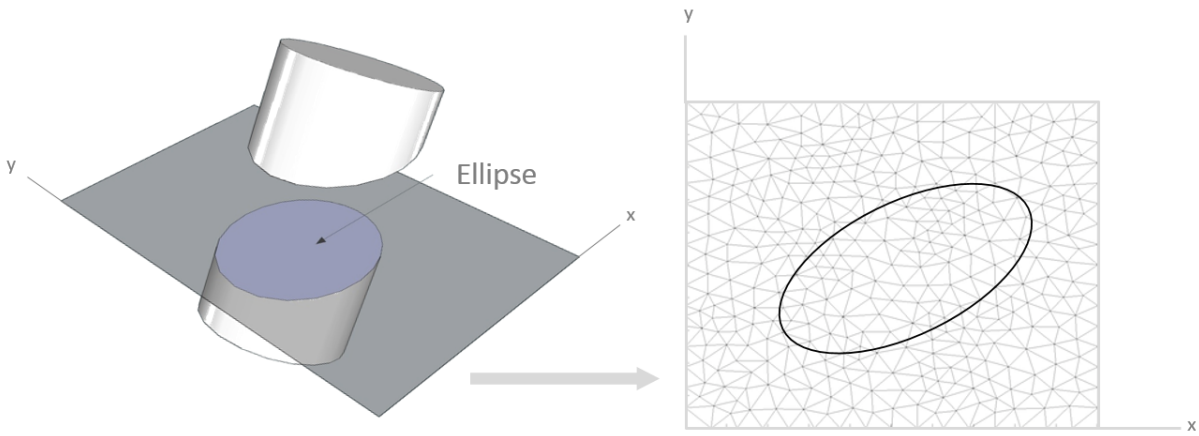
Tabela 6.53 – Discretização da malha de elementos finitos.

Tamanho base do elemento	0.015	m
Número de elementos	57230	
Número de nós	115161	

6.3.4 Solução analítica – Elipse

De forma a simular a corrente do mar aplicada ao *riser* foi criado um plano horizontal em diferentes profundidades. Cada plano é posicionado de acordo com a posição do *riser* em uma determinada profundidade e é responsável por resolver a interação fluido-estrutura e enviar as forças e momentos resultantes para o GIRAFFE. Como cada plano é horizontal a seção de corte entre o plano e o *riser* resulta em uma elipse (Fig. 6.53).

Figura 6.53 – Seção de corte.

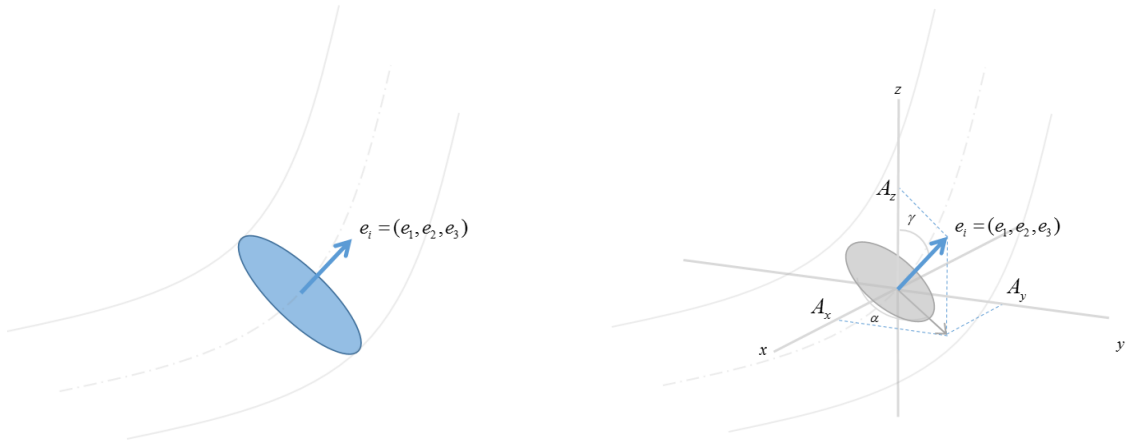


Fonte: O autor

O GIRAFFE retorna (em um determinado instante de tempo) o vetor tangente para um nó específico (Fig. 6.54a). Com este vetor é possível encontrar todos os parâmetros da elipse rotacionada como é apresentado na Fig. 6.54b.

Figura 6.54 – Vetor tangente e parâmetros da elipse.

(a) Vetor tangente. (b) Parâmetros da elipse.



Fonte: O autor

Existem duas formas que podem ser utilizadas para gerar o domínio da estrutura no formato de uma elipse. Gera-se ou uma malha de elementos finitos ou utiliza-se uma equação analítica. Como não é possível conhecer *a priori* a posição do riser em uma

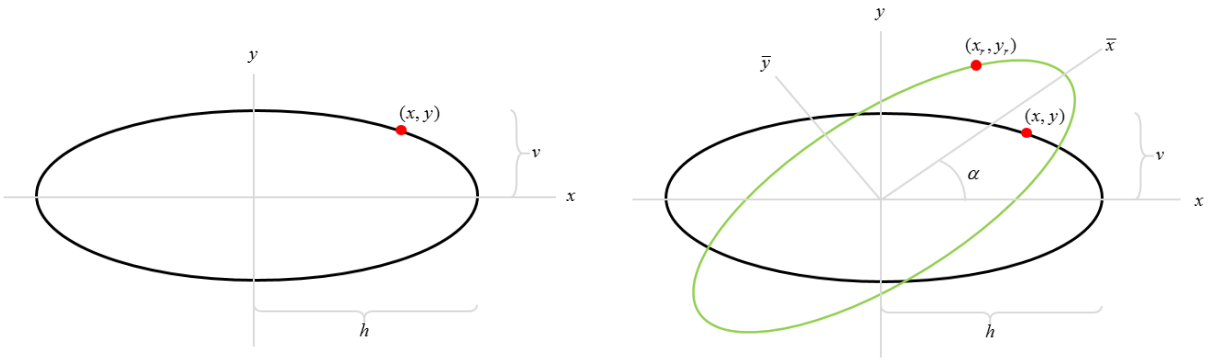
determinada profundidade – dificultando assim a geração da malha – a melhor forma é por meio de uma equação analítica.

6.3.5 Equação da elipse rotacionada

Suponha-se uma elipse com raio horizontal h e raio vertical v centrado na origem (Fig. 6.55a).

Figura 6.55 – Elipse e elipse rotacionada.

(a) Elipse. (b) Elipse rotacionada.



Fonte: O autor

A equação da elipse não-rotacionada é:

$$\frac{x^2}{h^2} + \frac{y^2}{v^2} = 1 \tag{6.10}$$

onde x e y são as coordenadas dos pontos sobre a elipse.

As equações que rotacionam um ponto em relação a origem de um ângulo α (anti-horário) são:

$$x_r = x\cos\alpha + y\sin\alpha \tag{6.11}$$

$$y_r = y\cos\alpha - x\sin\alpha \tag{6.12}$$

Substituindo estas equações na Eq. 6.10, tem-se a equação da elipse rotacionada (Fig. 6.55b):

$$\frac{(x\cos\alpha + y\sin\alpha)^2}{h^2} + \frac{(y\cos\alpha - x\sin\alpha)^2}{v^2} = 1 \tag{6.13}$$

6.3.6 Intersecção entre a elipse e a malha de fluido

É necessário encontrar os nós de intersecção entre a elipse rotacionada e a malha de fluido. Estes nós de intersecção definem a interface da superfície molhada. Para que se possa encontrar estes nós de intersecção, utiliza-se a seguinte equação:

$$x_{int} = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a} \tag{6.14}$$

onde

$$a = v^2\left(\cos^2\alpha + 2m\cos\alpha\sin\alpha + m^2\sin^2\alpha\right) + h^2\Big(m^2\cos^2\alpha - 2m\cos\alpha\sin\alpha + \\ + \sin^2\alpha\Big) \tag{6.15}$$

$$b = 2v^2 b_1\left(\cos\alpha\sin\alpha + m\sin^2\alpha\right) + 2h^2 b_1\left(m\cos^2\alpha - \cos\alpha\sin\alpha\right) \tag{6.16}$$

$$c = b_1^2\left(v^2\sin^2\alpha + h^2\cos^2\alpha\right) - h^2v^2 \tag{6.17}$$

Substituindo na equação de uma reta:

$$y = mx + b_1 \tag{6.18}$$

onde:

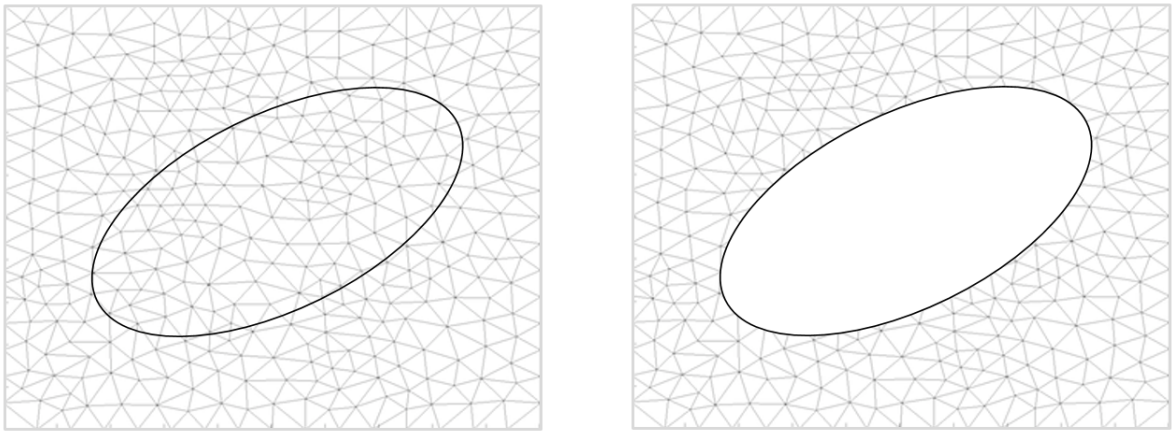
- m: Coeficiente angular da reta;
- b_1: Coeficiente linear da reta.

sendo assim possível encontrar os nós de intersecção entre a elipse e a malha de fluido.

As Fig. 6.56a e 6.56b mostram a elipse e a superfície molhada após encontrar os nós de intersecção e ocultando os elementos de fluido que estejam sob a estrutura.

Figura 6.56 – Elipse e superfície molhada

(a) Elipse. (b) Superfície molhada.

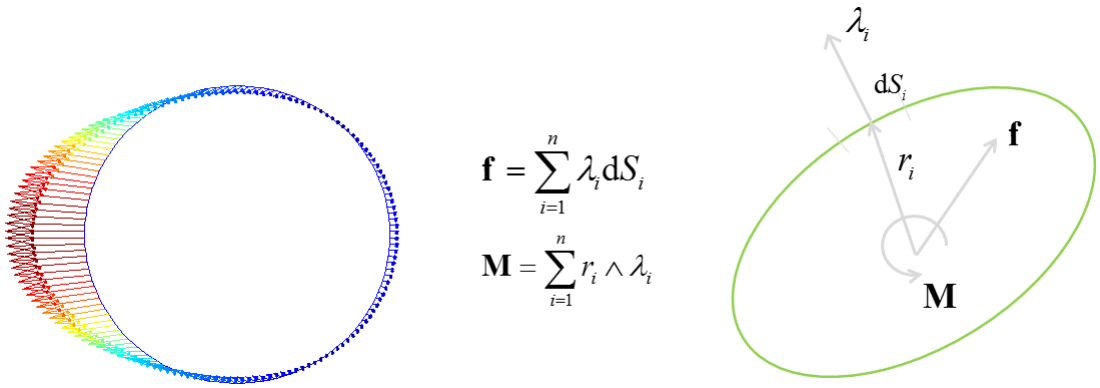


Fonte: O autor

6.3.7 Forças resultantes – Multiplicadores de Lagrange

As forças resultantes aplicadas ao *riser* são dadas pelos multiplicadores de Lagrange. A Fig. 6.57 ilusta como os vetores das forças resultantes e o momento são calculados.

Figura 6.57 – Vetor de forças resultantes e momento gerados pelos multiplicadores de Lagrange e enviados ao GIRAFFE.



Fonte: O autor

6.3.8 Caso estacionário

De forma a testar a solução analítica da elipse no HPC_LMC, foi simulado um escoamento em torno de uma elipse rígida (simulando a seção gerada pela intersecção entre o plano horizontal e o riser) em um problema estacionário.

Os parâmetros utilizados são:

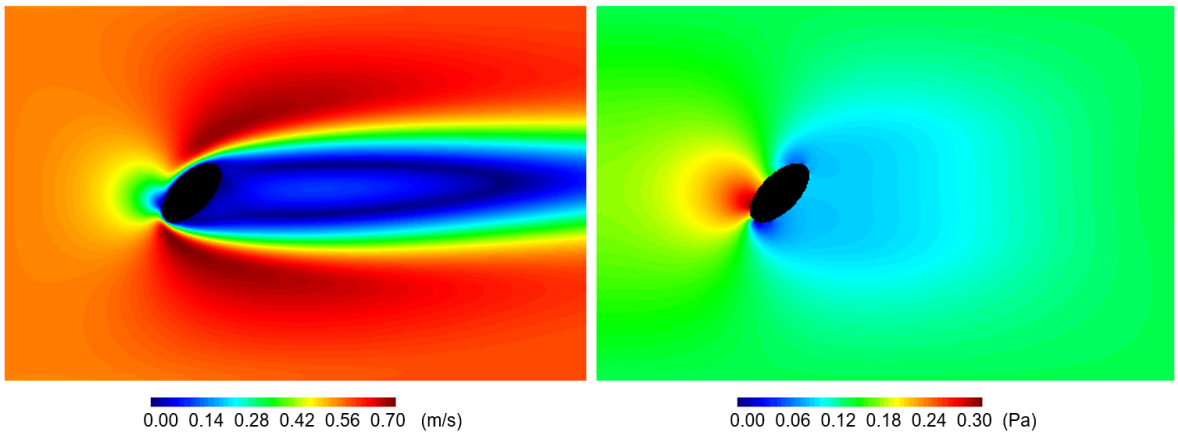
Tabela 6.54 – Parâmetros utilizados no caso estacionário.

Velocidade na entrada	0.20	m/s
Reynolds	40	

6.3.8.1 Resultados

A Fig. 6.58 mostra os resultados (campo de velocidade e pressão) para o caso estacionário.

Figura 6.58 – Escoamento em torno de uma elipse rígida; Campo de velocidade (esquerda) e pressão (direita) com $Re = 40$.



Fonte: O autor

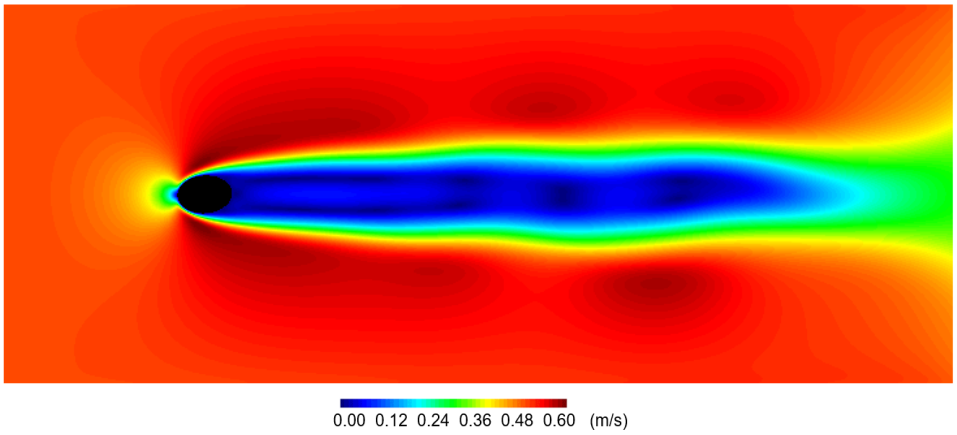
6.3.9 Caso transiente

Para o caso transiente foi utilizada a mesma geometria, propriedades do fluido, parâmetros da simulação e metodologia de análise como descrito nas seções anteriores. Utilizou-se uma velocidade crescente na entrada de 0.00 a 0.50 m/s nos primeiros dois segundos e posteriormente uma velocidade constante na entrada de 0.50 m/s até o fim da simulação.

6.3.9.1 Resultados

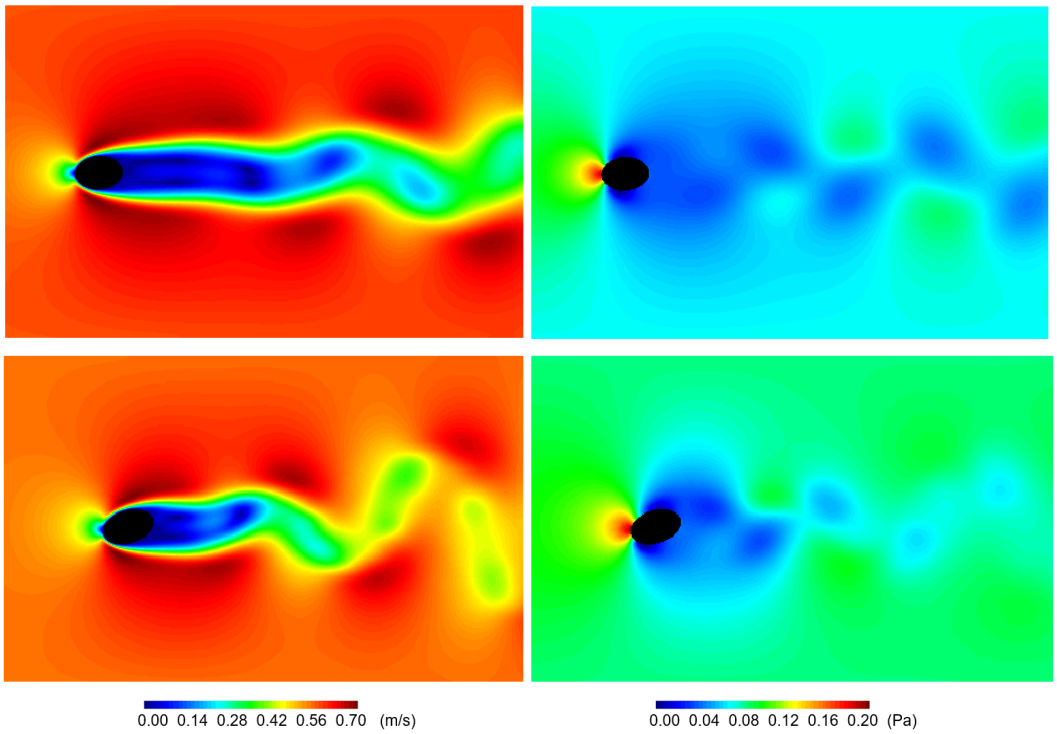
Primeiramente é apresentado o campo de velocidades em $t = 2.00\,\mathrm{s}$ com uma velocidade constante na entrada de 0.50 m/s. Posteriormente, para cada plano horizontal, são ilustrados os campos de velocidades e pressões em diferentes instantes de tempo ($t = 6.00\,\mathrm{s}$, $t = 7.50\,\mathrm{s}$ e $t = 9.00\,\mathrm{s}$) nas profundidades $Z = 100.00\,\mathrm{m}$ e $Z = 400.00\,\mathrm{m}$.

Figura 6.59 – Campo de velocidade em $t = 2.00\,\mathrm{s}$ e $Z = 100.00\,\mathrm{m}$.



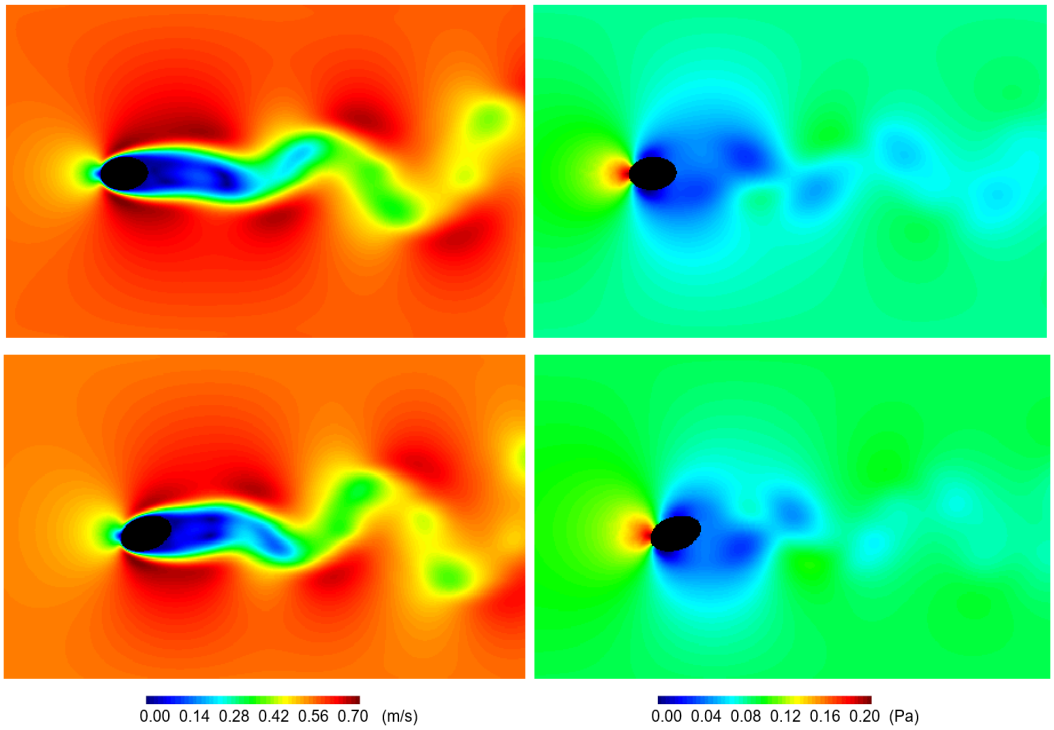
Fonte: O autor

Figura 6.60 – Escoamento em torno de uma seção de corte; Campo de velocidade (esquerda) e pressão (direita) com $Re = 100$ e $t = 6.00\,\mathrm{s}$; Profundidades $Z = 100.00\,\mathrm{m}$ (superior) e $Z = 400.00\,\mathrm{m}$ (inferior).



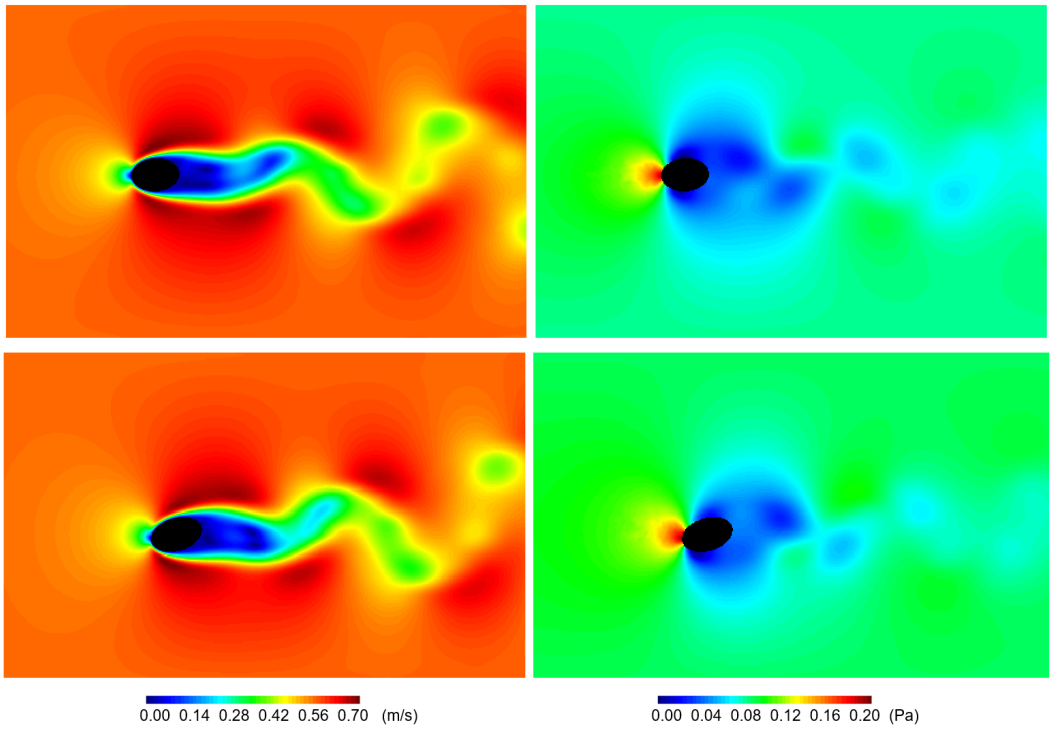
Fonte: O autor

Figura 6.61 – Escoamento em torno de uma seção de corte; Campo de velocidade (esquerda) e pressão (direita) com $Re = 100$, $t = 7.50\,\mathrm{s}$; Profundidades $Z = 100.00\,\mathrm{m}$ (superior) e $Z = 400.00\,\mathrm{m}$ (inferior).



Fonte: O autor

Figura 6.62 – Escoamento em torno de uma seção de corte; Campo de velocidade (esquerda) e pressão (direita) com $Re = 100$, $t = 9.00\,\mathrm{s}$; Profundidades $Z = 100.00\,\mathrm{m}$ (superior) e $Z = 400.00\,\mathrm{m}$ (inferior).



Fonte: O autor

7 CONCLUSÃO

Este trabalho apresentou uma implementação computacional para a solução da equação de Navier-Stokes para problemas bidimensionais e tridimensionais, tanto estacionários como transientes, utilizando o Método dos Elementos Finitos. Os elementos de Taylor-Hood P2P1 estabilizados para a condição LBB para problemas bidimensionais foram utilizados para a malha de elementos finitos. Foi desenvolvido um algoritmo em C++ para execução sequencial e um para execução em paralelo de forma a resolver os problemas de dinâmica dos fluidos e interação fluido-estrutura tanto em computadores sem placas de vídeo NVIDIA®, tanto em computadores que possuem o poder da GPU e podem aproveitar o máximo de sua performance. Alguns exemplos foram realizados para validação da implementação computacional e para análise da performance entre o código sequencial e paralelo. As seguintes conclusões podem ser destacadas:

- O código computacional desenvolvido é robusto para escoamentos laminares até valores de Reynolds próximos de 500 (problema do escoamento em uma cavidade) e facilmente extensível para outros problemas;
- Há um *speedup* considerável de cerca de 10 para o *assembly* das matrizes globais e 4 para a solução total do problema em comparação com o mesmo código sequencial (todos os problemas);
- O maior gargalo (para que não se obtenha maiores *speedups*) está na solução do sistema de equações lineares, dependente da quantidade de memória RAM disponível e do poder de processamento da CPU. Até a escrita deste trabalho, o método direto para a solução do sistemas de equações lineares (PARDISO) não permitia a sua paralelização na GPU. Uma possibilidade e muito utilizada na literatura, é a utilização do método iterativo e *multigrid*;
- Um gargalo muito comum na utilização de GPUs para solução de problemas científicos é a transferência de dados entre o *host* (CPU) e o *device* (GPU). Neste trabalho é apresentado uma forma de evitar este gargalo utilizando a memória compartilhada da GPU como também o acesso contíguo aos dados nodais dos elementos para o *assembly* das matrizes locais e globais, como apresentado no Cap. 2;
- Não foi discutido neste trabalho a utilização de *clusters* de GPUs. Muito comum nos dias de hoje, o *cluster* de GPUs possibilita a execução de problemas científicos que utilizem uma quantidade de dados massiva, como por exemplo análise tridimensional com um grande refinamento da malha de elementos finitos, e a turbulência sem a utilização de modelos numéricos como o DNS (*Direct Numerical Simulation*).

7.1 Trabalhos futuros

Dando prosseguimento ao trabalho aqui iniciado, são sugeridos os seguintes estudos futuros:

- Implementar a estabilização numérica para altos Reynolds aumentando o leque de problemas que podem ser resolvidos com o código computacional implementado;
- Implementar a solução do sistema de equações lineares pelo método iterativo (por exemplo, GMRES com pré-condicionadores);
- Utilizar o poder do *cluster* de GPUs possibilitando o estudo de problemas científicos que utilizem uma quantidade massiva de dados, como por exemplo o DNS;
- Implementar a interação fluido-estrutura tridimensional.

REFERÊNCIAS

AHRENS, J.; GEVECI, B.; LAW, C. *ParaView: An End-User Tool for Large Data Visualization*. [S.l.]: Elsevier, 2005.

AMDAHL, G. M. Validity of the Single Processor Approach to Achieving Large-Scale Computing Capabilities. In: *AFIPS spring joint computer conference*. [S.l.: s.n.], 1967.

BAIGES, J.; CODINA, R.; HENKE, F.; SHAHMIRI, S.; WALL, W. A symmetric method for weakly imposing Dirichlet boundary conditions in embedded Finite Element meshes. *International Journal for Numerical Methods in Engineering*, v. 90, p. 636–658, 2012.

BREZZI, F.; FORTIN, M. *Mixed and hybrid finite element methods*. [S.l.]: Springer-Verlag, 1991.

CECKA, C.; LEW, A. J.; DARVE, E. Assembly of Finite Element Methods on graphics processors. *International Journal for Numerical Methods in Engineering*, v. 85, n. 5, p. 640–669, 2011.

CONTI, T. d. N. *Aplicação do método da expansão em funções hierárquicas na solução das equações de Navier-Stokes em duas dimensões para fluidos compressíveis em alta velocidade*. Tese (Doutorado) — Universidade de São Paulo, 2006.

ERTURK, E. Discussions on driver cavity flow. *International Journal for Numerical Methods in Fluids*, v. 60, p. 275–294, 2009.

FARBER, R. *CUDA application design and development*. [S.l.]: Elsevier, 2011.

FORSTER, C.; WALL, W. A.; RAMM, E. Artificial added mass instabilities in sequential staggered coupling of nonlinear structures and incompressible viscous flows. *Computer Methods in Applied Mechanics and Engineering*, v. 196, p. 1278–1293, 2007.

FOWLER, M. *UML Distilled: A Brief Guide to the Standard Object Modeling Language*. [S.l.]: Addison-Wesley, 2004.

GAMNITZER, P. *Residual-based variational multiscale methods for turbulent flows and fluid-structure interaction*. Tese (Doutorado) — Technischen Universitat Munchen, 2010.

GAO, G. R.; SATO, M.; AYGUADé, E. Guest editors introduction: Special issue on OpenMP. *International Journal of Parallel Programming*, v. 36, n. 3, p. 287–288, 2008.

GASCHE, J. L.; BARBI, F.; VILLAR, M. M. An Efficient Immersed Boundary Method for Solving the Unsteady Flow through Actual Geometries of Reed Valves. In: *International Compressor Engineering Conference*. [S.l.: s.n.], 2012.

GAWLIK, E. S.; KABARIA, H.; LEW, A. J. High-order methods for low Reynolds number flows around moving obstacles based on universal meshes. *International Journal for Numerical Methods in Engineering*, 2015.

Gay Neto, A. *Giraffe User's Manual v.1.0.115*. [S.l.], 2016. Disponível em: <http://sites.poli.usp.br/p/alfredo.gay/GIRAFFE%20User's%20Manual%20v1.0.115.pdf>.

Gay Neto, A.; MALTA, E. R.; PIMENTA, P. M. Catenary riser sliding and rolling on seabed during induced lateral movement. *Marine Structures*, v. 41, p. 223–243, 2015.

Gay Neto, A.; MARTINS, C. A.; PIMENTA, P. M. Static analysis of offshore risers with a geometrically-exact 3D beam model subjected to unilateral contact. *Computational Mechanics*, v. 53, p. 125–145, 2013.

Gay Neto, A.; PIMENTA, P. M.; WRIGGERS, P. Contact between rolling beams and flat surfaces. *International Journal for Numerical Methods in Engineering*, v. 97, p. 683–706, 2014.

GERSTENBERGER, A. *An XFEM based fixed-grid approach to fluid-structure interaction*. Tese (Doutorado) — Technischen Universistat Munchen, 2010.

GERSTENBERGER, A.; WALL, W. A. Enhancement of fixed-grid methods towards complex fluid-structure interaction applications. *International Journal for Numerical Methods in Fluids*, v. 57, p. 1227–1248, 2008.

GERSTENBERGER, A.; WALL, W. A. An embedded Dirichlet formulation for 3D continua. *International Journal for Numerical Methods in Engineering*, v. 82, p. 537–563, 2010.

GHIA, U.; GHIA, K. N.; SHIN, C. T. High-Re solutions for incompressible flow using Navier-Stokes equations and a multigrid method. *Journal of Computational Physics*, v. 48, p. 387–411, 1982.

GöDDEKE, D. *Fast and accurate Finite-Element multigrid solvers for PDE simulations on GPU clusters*. Tese (Doutorado) — Technischen Universitat Dortmund, 2011.

GODDEKE, D.; BUIJSSEN, S. H. M.; WOBKER, H.; TUREK, S. GPU acceleration of an unmodified parallel finite element Navier-Stokes solver. In: *2009 International Conference on High Performance Computing & Simulation*. [S.l.: s.n.], 2009.

GOMES, H. C. *Método dos Elementos Finitos com Fronteiras Imersas aplicado a problemas de dinâmica dos fluidos e interação fluido-estrutura*. Tese (Doutorado) — Universidade de São Paulo, 2013.

GOMES, H. C.; PIMENTA, P. M. Embedded interface with discontinuous Lagrange multipliers for fluid-structure interaction analysis. *International Journal for Computational Methods in Engineering Science and Mechanics*, v. 16, p. 98–111, 2015.

HEISTER, T.; KRONBICHLER, M.; BANGERTH, W. Generic Finite Element programming for massively parallel flow simulations. In: *V European Conference on Computational Fluid Dynamics*. Lisboa, Portugal: [s.n.], 2010.

HEISTER, T.; KRONBICHLER, M.; BANGERTH, W. Massively parallel finite element programming. *Lecture Notes in Computer Science*, v. 6305, p. 122–131, 2010.

HSIEH, S.-H.; YANG, Y.-S.; TSAI, P.-L. Improved mesh partiioning for parallel substructure Finite Element computations. In: *The 7th East Asia-Pacific Conference on Structural Engineering and Construction*. Kochi, Japan: [s.n.], 1999.

ILINCA, F.; HETU, J.-F. A finite element immersed boundary method for fluid flow around rigid objects. *International Journal for Numerical Methods in Fluids*, 2010.

JOHN, V. A comparison of parallel solvers for the incompressible Navier-Stokes equations. *Computing and Visualization in Science*, v. 1, n. 4, p. 193–200, 1999.

KANG, S.; IACCARINO, G.; HAM, F. Dns of buoyancy-dominated turbulent flows on a bluff body using the immersed boundary method. *Journal of Computational Physics*, v. 228, p. 3189–3208, 2009.

KARLSSON, N. *An incompressible Navier-Stokes equations solver on the GPU using CUDA*. Tese (Doutorado) — Chalmers University of Technology, 2013.

KINDRATENKO, V. *Numerical computations with GPUs*. [S.l.]: Springer, 2014.

KOLLMANNSBERGER, S.; OZCAN, A.; BAIGES, J.; RUESS, M.; RANK, E.; REALI, A. Parameter-free, weak imposition of Dirichlet boundary conditions and coupling of trimmed and non-conforming patches. *International Journal for Numerical Methods in Engineering*, v. 101, p. 670–699, 2015.

KUTTLER, U.; WALL, W. A. Fixed-point fluid-structure interaction solvers with dynamic relaxation. *Computational Mechanics*, v. 43, p. 61–72, 2008.

KUZMIN, A.; LUISIER, M.; SCHENK, O. Fast methods for computing selected elements of the greens function in massively parallel nanoelectronic device simulations. In: WOLF, F.; MOHR, B.; MEY, D. (Ed.). *Euro-Par 2013 Parallel Processing*. Springer Berlin Heidelberg, 2013, (Lecture Notes in Computer Science, v. 8097). p. 533–544. ISBN 978-3-642-40046-9. Disponível em: <http://dx.doi.org/10.1007/978-3-642-40047-6_54>.

LESOINNE, M.; FARHAT, C. Geometric conservation laws for flow problems with moving boundaries and deformable meshes, and their impact on aeroelastic computations. *Computational Methods Applied to Mechanical Engineering*, v. 134, p. 71–90, 1996.

LIEU, T.; FARHAT, C.; LESOINNE, M. Reduced-order fluid/structure modeling of a complete aircraft configuration. *Computer Methods in Applied Mechanics and Engineering*, v. 195, n. 41-43, p. 5730–5742, 2006.

LIRKOV, I.; MARGENOV, S.; WASNIEWSKI, J. Large-Scale scientific computing. In: *8th International Conferente, LSSC 2011*. Sozopol, Bulgaria: [s.n.], 2011.

MELENDO, A.; COLL, A.; PASENAU, M.; ESCOLANO, E.; MONROS, A. *www.gidhome.com*. 2015. [Online; accessed Nov-2015]. Disponível em: <http://www.gidhome.com>.

MITTAL, S.; TEZDUYAR, T. E. Massively parallel Finite Element computation of incompressible flows involving fluid-body interactions. *Computer Methods in Applied Mechanics and Engineering*, v. 112, n. 1-4, p. 253–282, 1994. ISSN 00457825.

MOLYNEAUX, I. *The Art of Application Performance Testing*. [S.l.]: O'Reilly, 2009.

MORAES, S. R. d. S. *Computação paralela em cluster de GPU aplicado a problema da Engenharia Nuclear*. Tese (Doutorado) — Instituto de Engenharia Nuclear, 2012.

PADUA, D. *Encyclopedia of parallel computing*. [S.l.]: Springer, 2011.

PIMENTA, P. M.; CAMPELLO, E. M. B.; WRIGGERS, P. A fully nonlinear multi-parameter shell model with thickness variation and a triangular shell Finite Element. *Computational Mechanics*, v. 34, p. 181–193, 2004.

SANDERS, J.; KANDROT, E. *CUDA by example: An introduction to General-Purpose programming*. [S.l.]: Addison-Wesley, 2011.

SCHAFFER, M.; TUREK, S.; DURST, F.; KRAUSE, E.; RANNACHER, R. Benchmark computations of laminar flow around a cylinder. *Flow Simulation with High-Performance Computers II*, v. 48, p. 547–566, 1996.

SCHROEDER, W.; MARTIN, K.; LORENSEN, B. *The Visualization Toolkit*. [S.l.]: Kitware, 2006.

SEO, J. H.; MITTAL, R. A sharp-interface immersed boundary method with improved mass conservation and reduced pressure oscillations. *Journal of Computational Physics*, v. 230, p. 7347–7363, 2011.

SHAHMIRI, S. *A hybrid ALE-fixed-grid approach for fluid-structure interaction*. Tese (Doutorado) — Technischen Universistat Munchen, 2014.

SUDHAKAR, Y.; ALMEIDA, J. P. M. de; WALL, W. A. An accurate, robust, and easy-to-implement method for integration over arbitrary polyhedra: Application to embedded interface methods. *Journal of Computational Physics*, v. 273, p. 393–415, 2014.

TEZDUYAR, T. E.; SATHE, S.; SENGA, M.; AURELI, L. Finite element modeling of fluidstructure interactions with spacetime and advanced mesh update techniques. In: *10th International Conference on Numerical Methods in Continuum Mechanics*. Zilina, Slovakia: [s.n.], 2005.

WALL, W. A.; GERSTENBERGER, A.; GAMNITZER, P.; FöRSTER, C.; RAMM, E. Large deformation fluid-structure interaction - Advances in ALE methods and new fixed grid approaches. *Fluid-Structure Interaction: Modelling, Simulation, Optimisation*, v. 53, p. 195–232, 2006.

WALL, W. A.; KUTTLER, U.; GERSTENBERGER, A.; GEE, M.; FORSTER, C. Advances in computational fluid-thin-walled-structure interaction - Formulations and Solvers. *New Trends in Thin Structures: Formulation, Optimization and Coupled Problems*, v. 519, n. 2001, p. 175–203, 2010.

WANG, C. Y. Exact solutions of the steady-state Navier-Stokes equations. *Annual Review of Fluid Mechanics*, v. 23, p. 159–177, 1991.

YAO, J.; LIU, G. R.; NARMONEVA, D. A.; HINTON, R. B.; ZHANG, Z. Q. Immersed smoothed finite element method for fluid-structure interaction simulation of aortic valves. *Computational Mechanics*, v. 50, n. 6, p. 789–804, 2012.

YILDIRIM, B.; LIN, S.; MATHUR, S.; MURTHY, J. Y. A parallel implementation of fluid-solid interaction solver using an Immersed Boundary Method. *Computers and Fluids*, Elsevier Ltd., v. 86, p. 251–274, 2013.

ZHANG, Z.-Q.; LIU, G. R.; KHOO, B. C. Immersed smoothed finite element method fot two dimensional fluid-structure interaction. *International Journal for Numerical Methods in Engineering*, v. 90, p. 1292–1320, 2012.

ZHANG, Z. Q.; LIU, G. R.; KHOO, B. C. A three dimensional immersed smoothed finite element method (3D IS-FEM) for fluid-structure interaction problems. *Computational Mechanics*, v. 51, n. 2, p. 129–150, 2013.

APÊNDICE A – CUDA

Neste apêndice é apresentado o código-fonte da implementação computacional em CUDA e todas as funções auxiliares como manipulação de matrizes, verificação de erros associados ao CUDA e o *assembly* das matrizes.

- *CUDA_Macros.cuh*

```
/*****************************************************************************/
//                                HPC - LMC                                  //
//---------------------------------------------------------------------------//
/*!
 * \brief   CUDA_Macros
 *
 * Esta classe armazena as macros do projeto CUDA
 *
 * \author  Luiz Felipe Couto
 */
//////////////////////////////////////////////////////////////////////////////

#ifndef HPC_LMC_FEM_CUDA_CORE_CUDA_MACROS_H
#define HPC_LMC_FEM_CUDA_CORE_CUDA_MACROS_H


//---------------------------------------------------------------------------
// Include and Forward declaration
//---------------------------------------------------------------------------
#include <cuda_runtime.h>
#include <math.h>
#include <stdio.h>

#define IDX2C(i, j, ld) (((j) * (ld)) + (i))

static void checkCudaErrors(cudaError_t err,
                            const char *file,
                            int line ) {

    if (err != cudaSuccess) {
        printf("%s in %s at line %d\n", cudaGetErrorString(err), file, line);
    }
}

#define HANDLE_ERROR(err) (checkCudaErrors(err, __FILE__, __LINE__))

#if defined(USE_SINGLE_PRECISION)
#define Precision float
#elif defined(USE_DOUBLE_PRECISION)
#define Precision double
#endif

#endif
```

- *CUDA_MatrixOperation.cuh*

```
/*****************************************************************************/
//                                HPC - LMC                                  //
//---------------------------------------------------------------------------//
/*!
 * \brief   Header MatrixOperation
 *
 * Este arquivo é responsável pelas operações com matrizes pequenas na GPU
 *
```

```
 * \author  Luiz Felipe Couto
 */
////////////////////////////////////////////////////////////////////////////////

#ifndef HPC_LMC_FEM_CUDA_CORE_MATRIXOPERATION_H
#define HPC_LMC_FEM_CUDA_CORE_MATRIXOPERATION_H

//------------------------------------------------------------------------------
// Include and Forward declaration
//------------------------------------------------------------------------------
#include "CUDA_Macros.cuh"

//------------------------------------------------------------------------------
// Prototype definition
//------------------------------------------------------------------------------

//! Este método realiza a soma de duas matrizes
/*!
 * \param M Número de linhas
 * \param N Número de colunas
 * \param A Matriz A
 * \param B Matriz B
 * \param C Matriz C
 */
__device__
void sum(   int                 M,
            int                 N,
            const Precision*    A,
            const Precision*    B,
            Precision*          C);

//! Este método realiza a subtração de duas matrizes
/*!
 * \param M Número de linhas
 * \param N Número de colunas
 * \param A Matriz A
 * \param B Matriz B
 * \param C Matriz C
 */
__device__
void subtract(  int                 M,
                int                 N,
                const Precision*    A,
                const Precision*    B,
                Precision*          C);

//! Este método realiza a multiplicação de duas matrizes
/*!
 * Operação: A(M x K) * B(K x N) = C(M x N)
 *
 * \param M Número de linhas da matriz A ou C
 * \param N Número de colunas da matriz B ou C
 * \param K Número de colunas de A ou linhas de B
 * \param A Matriz A
 * \param B Matriz B
 * \param C Matriz C
 */
__device__
void multiply(  int                 M,
                int                 N,
                int                 K,
                const Precision*    A,
                const Precision*    B,
                Precision*          C);

//! Este método realiza a multiplicação de matriz por escalar
/*!
 * \param M Número de linhas da matriz A ou C
 * \param N Número de colunas da matriz B ou C
 * \param scalar Escalar
 * \param A Matriz A
 * \param B Matriz B
 */
__device__
void multiplyScalar(int                 M,
                    int                 N,
```

```
                    Precision           scalar,
                    const Precision*    A,
                    Precision*          B);

//! Este método realiza a transposição da matriz
/*!
 * \param M Número de linhas
 * \param N Número de colunas
 * \param A Matriz A
 * \param A_T Matriz transposta de A
 */
__device__
void transpose( int                 M,
                int                 N,
                const Precision*    A,
                Precision*          A_T);

//! Este método realiza o cálculo do determinante da matriz
/*!
 * \param M Número de linhas
 * \param N Número de colunas
 * \param A Matriz A
 * \result
 */
__device__
Precision determinant(  int         M,
                        int         N,
                        Precision*  A);

//! Este método realiza o cálculo da inversa da matriz
/*!
 * \param M Número de linhas
 * \param determinant Determinante da matriz A
 * \param A Matriz A
 * \param AInverse Inversa da matriz A
 */
__device__
void inverse(   size_t              M,
                Precision           determinant,
                const Precision*    A,
                Precision*          AInverse);

//! Este método realiza retira um bloco de uma matriz
/*!
 * \param iStartRow Linha inicial
 * \param iStartCol Coluna inicial
 * \param iNumberRows Número de linhas do bloco
 * \param iNumberCols Número de colunas do bloco
 * \param A Matriz original
 * \param A_Block Bloco da matriz original
 * \param ldA_Block Leading do bloco
 */
__device__
void block( size_t              iStartRow,
            size_t              iStartCol,
            size_t              iNumberRows,
            size_t              iNumberCols,
            const Precision*    A,
            Precision*          A_Block,
            size_t              ldA_Block);

//==============================================================================
// Implementation
//==============================================================================

//----------------------------- sum ----------------------------------------
__device__
void sum(   int                 M,
            int                 N,
            const Precision*    A,
            const Precision*    B,
            Precision*          C)
{
    for (int m = 0; m < M; ++m)
    {
        for (int n = 0; n < N; ++n)
```

```
161         {
162             C[IDX2C(m, n, M)] = A[IDX2C(m, n, M)] + B[IDX2C(m, n, M)];
163         }
164     }
165 }
166
167 //----------------------------- subtract -----------------------------------
168 __device__
169 void subtract(  int                 M,
170                 int                 N,
171                 const Precision*    A,
172                 const Precision*    B,
173                 Precision*          C)
174 {
175     for (int m = 0; m < M; ++m)
176     {
177         for (int n = 0; n < N; ++n)
178         {
179             C[IDX2C(m, n, M)] = A[IDX2C(m, n, M)] - B[IDX2C(m, n, M)];
180         }
181     }
182 }
183
184 //----------------------------- multiply -----------------------------------
185 __device__
186 void multiply(  int                 M,
187                 int                 N,
188                 int                 K,
189                 const Precision*    A,
190                 const Precision*    B,
191                 Precision*          C)
192 {
193     for (int m = 0; m < M; ++m)
194     {
195         for (int n = 0; n < N; ++n)
196         {
197             C[IDX2C(m, n, M)] = 0.;
198
199             for (int k = 0; k < K; ++k)
200             {
201                 C[IDX2C(m, n, M)] += A[IDX2C(m, k, M)] * B[IDX2C(k, n, K)];
202             }
203         }
204     }
205 }
206
207 //----------------------------- multiplyScalar -----------------------------
208 __device__
209 void multiplyScalar(int                 M,
210                     int                 N,
211                     Precision           scalar,
212                     const Precision*    A,
213                     Precision*          B)
214 {
215     for (int m = 0; m < M; ++m)
216     {
217         for (int n = 0; n < N; ++n)
218         {
219             B[IDX2C(m, n, M)] = scalar * A[IDX2C(m, n, M)];
220         }
221     }
222 }
223
224 //----------------------------- transpose ----------------------------------
225 __device__
226 void transpose( int                 M,
227                 int                 N,
228                 const Precision*    A,
229                 Precision*          A_T)
230 {
231     for (int m = 0; m < M; ++m)
232     {
233         for (int n = 0; n < N; ++n)
234         {
235             A_T[IDX2C(n, m, N)] = A[IDX2C(m, n, M)];
236         }
```

```
    }
}

//----------------------------- determinant ---------------------------------
__device__
Precision determinant(  int         M,
                        int         N,
                        Precision*  A)
{
    Precision determinant = 0.;

    if (M == 2 && N == 2)
    {
        determinant =   A[IDX2C(0, 0, M)] * A[IDX2C(1, 1, M)] -
                        A[IDX2C(1, 0, M)] * A[IDX2C(0, 1, M)];
    }
    else if (M == 3 && N == 3)
    {
        determinant =   A[IDX2C(0, 0, M)] * A[IDX2C(1, 1, M)] * A[IDX2C(2, 2, M)] +
                        A[IDX2C(0, 1, M)] * A[IDX2C(1, 2, M)] * A[IDX2C(2, 0, M)] +
                        A[IDX2C(0, 2, M)] * A[IDX2C(2, 1, M)] * A[IDX2C(1, 0, M)] -
                        (   A[IDX2C(2, 0, M)] * A[IDX2C(1, 1, M)] * A[IDX2C(0, 2, M)] +
                            A[IDX2C(2, 1, M)] * A[IDX2C(1, 2, M)] * A[IDX2C(0, 0, M)] +
                            A[IDX2C(2, 2, M)] * A[IDX2C(0, 1, M)] * A[IDX2C(1, 0, M)]);
    }

    return determinant;
}

//----------------------------- inverse -------------------------------------
__device__
void inverse(   size_t              M,
                Precision           determinant,
                const Precision*    A,
                Precision*          AInverse)
{
    if (M == 2)
    {
        AInverse[IDX2C(0, 0, M)] = (1. / determinant) * A[IDX2C(1, 1, M)];
        AInverse[IDX2C(0, 1, M)] = -(1. / determinant) * A[IDX2C(0, 1, M)];
        AInverse[IDX2C(1, 0, M)] = -(1. / determinant) * A[IDX2C(1, 0, M)];
        AInverse[IDX2C(1, 1, M)] = (1. / determinant) * A[IDX2C(0, 0, M)];
    }
}

//----------------------------- block ---------------------------------------
__device__
void block( size_t              iStartRow,
            size_t              iStartCol,
            size_t              iNumberRows,
            size_t              iNumberCols,
            const Precision*    A,
            Precision*          A_Block,
            size_t              ldA_Block)
{
    for (size_t i = 0; i < iNumberRows; ++i)
    {
        for (int j = 0; j < iNumberCols; ++j)
        {
            A_Block[IDX2C(iStartRow + i, iStartCol + j, ldA_Block)] = A[IDX2C(i, j,
                iNumberRows)];
        }
    }
}

#endif
```

- *CUDA_NumericalIntegration.cuh*

```
/*****************************************************************************/
//                                HPC - LMC                                  //
//---------------------------------------------------------------------------//
/*!
```

```
5   * \brief   Classe CUDA_NumericalIntegration
6   *
7   * Este header é responsável pela definição dos pontos de integração
8   *
9   * \author  Luiz Felipe Couto
10  */
11 ////////////////////////////////////////////////////////////////////////////////
12
13 #ifndef HPC_LMC_FEM_CUDA_CORE_CUDA_NUMERICALINTEGRATION_H
14 #define HPC_LMC_FEM_CUDA_CORE_CUDA_NUMERICALINTEGRATION_H
15
16 //------------------------------------------------------------------------------
17 // Include and Forward declaration
18 //------------------------------------------------------------------------------
19 #include "CUDA_Macros.cuh"
20
21 //------------------------------------------------------------------------------
22 // Enumeration definition
23 //------------------------------------------------------------------------------
24 //! Enumeration que define o tipo de integração utilizar.
25 /*!
26  * Se utilizada a integração de Gauss, utilizar a quadratura adequada ao tipo
27  * de integração.
28  */
29 enum IntegrationElementType
30 {
31     TRIANGLE,           //!< Triângulo
32     QUADRILATERAL       //!< Quadrilátero
33 };
34
35 //------------------------------------------------------------------------------
36 // Prototype definition
37 //------------------------------------------------------------------------------
38
39 //! Define as regras de integração
40 /*!
41  * \param integrationElementType Tipo de elemento de integração
42  * \param iIntegrationOrder Ordem de integração
43  * \param rules Regras de integração
44  */
45 __device__
46 void getIntegrationRules(   IntegrationElementType  integrationElementType,
47                             size_t                  iIntegrationOrder,
48                             Precision*              rules);
49
50 //------------------------------------------------------------------------------
51 // Device Function
52 //------------------------------------------------------------------------------
53
54 //----------------------------- initializeIntegrationRules -------------------
55 __device__
56 void getIntegrationRules(   IntegrationElementType  integrationElementType,
57                             size_t                  iIntegrationOrder,
58                             Precision*              rules)
59 {
60     if (integrationElementType == TRIANGLE)
61     {
62         if (iIntegrationOrder == 1)
63         {
64             rules[IDX2C(0, 0, 12)] = 1.000000000000000;
65             rules[IDX2C(0, 1, 12)] = 0.333333333333333;
66             rules[IDX2C(0, 2, 12)] = 0.333333333333333;
67             rules[IDX2C(0, 3, 12)] = 0.333333333333333;
68         }
69         else if (iIntegrationOrder == 3)
70         {
71             rules[IDX2C(0, 0, 12)] = 0.333333333333333;
72             rules[IDX2C(0, 1, 12)] = 0.666666666666667;
73             rules[IDX2C(0, 2, 12)] = 0.166666666666667;
74             rules[IDX2C(0, 3, 12)] = 0.000000000000000;
75
76             rules[IDX2C(1, 0, 12)] = 0.333333333333333;
77             rules[IDX2C(1, 1, 12)] = 0.166666666666667;
78             rules[IDX2C(1, 2, 12)] = 0.666666666666667;
79             rules[IDX2C(1, 3, 12)] = 0.000000000000000;
80
```

```
            rules[IDX2C(2, 0, 12)] = 0.333333333333333;
            rules[IDX2C(2, 1, 12)] = 0.166666666666667;
            rules[IDX2C(2, 2, 12)] = 0.166666666666667;
            rules[IDX2C(2, 3, 12)] = 0.000000000000000;
        }
        else if (iIntegrationOrder == 4)
        {
            rules[IDX2C(0, 0, 12)] = -0.562500000000000;
            rules[IDX2C(0, 1, 12)] = 0.333333333333333;
            rules[IDX2C(0, 2, 12)] = 0.333333333333333;
            rules[IDX2C(0, 3, 12)] = 0.000000000000000;

            rules[IDX2C(1, 0, 12)] = 0.520833333333333;
            rules[IDX2C(1, 1, 12)] = 0.600000000000000;
            rules[IDX2C(1, 2, 12)] = 0.200000000000000;
            rules[IDX2C(1, 3, 12)] = 0.000000000000000;

            rules[IDX2C(2, 0, 12)] = 0.520833333333333;
            rules[IDX2C(2, 1, 12)] = 0.200000000000000;
            rules[IDX2C(2, 2, 12)] = 0.600000000000000;
            rules[IDX2C(2, 3, 12)] = 0.000000000000000;

            rules[IDX2C(3, 0, 12)] = 0.520833333333333;
            rules[IDX2C(3, 1, 12)] = 0.200000000000000;
            rules[IDX2C(3, 2, 12)] = 0.200000000000000;
            rules[IDX2C(3, 3, 12)] = 0.000000000000000;
        }
        else if (iIntegrationOrder == 6)
        {
            rules[IDX2C(0, 0, 12)] = 0.109951743655322;
            rules[IDX2C(0, 1, 12)] = 0.816847572980459;
            rules[IDX2C(0, 2, 12)] = 0.091576213509771;
            rules[IDX2C(0, 3, 12)] = 0.000000000000000;

            rules[IDX2C(1, 0, 12)] = 0.109951743655322;
            rules[IDX2C(1, 1, 12)] = 0.091576213509771;
            rules[IDX2C(1, 2, 12)] = 0.816847572980459;
            rules[IDX2C(1, 3, 12)] = 0.000000000000000;

            rules[IDX2C(2, 0, 12)] = 0.109951743655322;
            rules[IDX2C(2, 1, 12)] = 0.091576213509771;
            rules[IDX2C(2, 2, 12)] = 0.091576213509771;
            rules[IDX2C(2, 3, 12)] = 0.000000000000000;

            rules[IDX2C(3, 0, 12)] = 0.223381589678011;
            rules[IDX2C(3, 1, 12)] = 0.108103018168070;
            rules[IDX2C(3, 2, 12)] = 0.445948490915965;
            rules[IDX2C(3, 3, 12)] = 0.000000000000000;

            rules[IDX2C(4, 0, 12)] = 0.223381589678011;
            rules[IDX2C(4, 1, 12)] = 0.445948490915965;
            rules[IDX2C(4, 2, 12)] = 0.108103018168070;
            rules[IDX2C(4, 3, 12)] = 0.000000000000000;

            rules[IDX2C(5, 0, 12)] = 0.223381589678011;
            rules[IDX2C(5, 1, 12)] = 0.445948490915965;
            rules[IDX2C(5, 2, 12)] = 0.445948490915965;
            rules[IDX2C(5, 3, 12)] = 0.000000000000000;
        }
        else if (iIntegrationOrder == 7)
        {
            rules[IDX2C(0, 0, 12)] = 0.225000000000000;
            rules[IDX2C(0, 1, 12)] = 0.333333333333333;
            rules[IDX2C(0, 2, 12)] = 0.333333333333333;
            rules[IDX2C(0, 3, 12)] = 0.000000000000000;

            rules[IDX2C(1, 0, 12)] = 0.125939180544827;
            rules[IDX2C(1, 1, 12)] = 0.797426985353087;
            rules[IDX2C(1, 2, 12)] = 0.101286507323456;
            rules[IDX2C(1, 3, 12)] = 0.000000000000000;

            rules[IDX2C(2, 0, 12)] = 0.125939180544827;
            rules[IDX2C(2, 1, 12)] = 0.101286507323456;
            rules[IDX2C(2, 2, 12)] = 0.797426985353087;
            rules[IDX2C(2, 3, 12)] = 0.000000000000000;

```

```
157             rules[IDX2C(3, 0, 12)] = 0.125939180544827;
158             rules[IDX2C(3, 1, 12)] = 0.101286507323456;
159             rules[IDX2C(3, 2, 12)] = 0.101286507323456;
160             rules[IDX2C(3, 3, 12)] = 0.000000000000000;
161
162             rules[IDX2C(4, 0, 12)] = 0.132394152788506;
163             rules[IDX2C(4, 1, 12)] = 0.059715871789770;
164             rules[IDX2C(4, 2, 12)] = 0.470142064105115;
165             rules[IDX2C(4, 3, 12)] = 0.000000000000000;
166
167             rules[IDX2C(5, 0, 12)] = 0.132394152788506;
168             rules[IDX2C(5, 1, 12)] = 0.470142064105115;
169             rules[IDX2C(5, 2, 12)] = 0.059715871789770;
170             rules[IDX2C(5, 3, 12)] = 0.000000000000000;
171
172             rules[IDX2C(6, 0, 12)] = 0.132394152788506;
173             rules[IDX2C(6, 1, 12)] = 0.470142064105115;
174             rules[IDX2C(6, 2, 12)] = 0.470142064105115;
175             rules[IDX2C(6, 3, 12)] = 0.000000000000000;
176         }
177         else if (iIntegrationOrder == 9)
178         {
179             rules[IDX2C(0, 0, 12)] = 0.205950504760887;
180             rules[IDX2C(0, 1, 12)] = 0.124949503233232;
181             rules[IDX2C(0, 2, 12)] = 0.437525248383384;
182             rules[IDX2C(0, 3, 12)] = 0.000000000000000;
183
184             rules[IDX2C(1, 0, 12)] = 0.205950504760887;
185             rules[IDX2C(1, 1, 12)] = 0.437525248383384;
186             rules[IDX2C(1, 2, 12)] = 0.124949503233232;
187             rules[IDX2C(1, 3, 12)] = 0.000000000000000;
188
189             rules[IDX2C(2, 0, 12)] = 0.205950504760887;
190             rules[IDX2C(2, 1, 12)] = 0.437525248383384;
191             rules[IDX2C(2, 2, 12)] = 0.437525248383384;
192             rules[IDX2C(2, 3, 12)] = 0.000000000000000;
193
194             rules[IDX2C(3, 0, 12)] = 0.063691414286223;
195             rules[IDX2C(3, 1, 12)] = 0.797112651860071;
196             rules[IDX2C(3, 2, 12)] = 0.165409927389841;
197             rules[IDX2C(3, 3, 12)] = 0.000000000000000;
198
199             rules[IDX2C(4, 0, 12)] = 0.063691414286223;
200             rules[IDX2C(4, 1, 12)] = 0.797112651860071;
201             rules[IDX2C(4, 2, 12)] = 0.037477420750088;
202             rules[IDX2C(4, 3, 12)] = 0.000000000000000;
203
204             rules[IDX2C(5, 0, 12)] = 0.063691414286223;
205             rules[IDX2C(5, 1, 12)] = 0.165409927389841;
206             rules[IDX2C(5, 2, 12)] = 0.797112651860071;
207             rules[IDX2C(5, 3, 12)] = 0.000000000000000;
208
209             rules[IDX2C(6, 0, 12)] = 0.063691414286223;
210             rules[IDX2C(6, 1, 12)] = 0.165409927389841;
211             rules[IDX2C(6, 2, 12)] = 0.037477420750088;
212             rules[IDX2C(6, 3, 12)] = 0.000000000000000;
213
214             rules[IDX2C(7, 0, 12)] = 0.063691414286223;
215             rules[IDX2C(7, 1, 12)] = 0.037477420750088;
216             rules[IDX2C(7, 2, 12)] = 0.797112651860071;
217             rules[IDX2C(7, 3, 12)] = 0.000000000000000;
218
219             rules[IDX2C(8, 0, 12)] = 0.063691414286223;
220             rules[IDX2C(8, 1, 12)] = 0.037477420750088;
221             rules[IDX2C(8, 2, 12)] = 0.165409927389841;
222             rules[IDX2C(8, 3, 12)] = 0.000000000000000;
223         }
224         else if (iIntegrationOrder == 12)
225         {
226             rules[IDX2C(0, 0, 12)] = 0.050844906370207;
227             rules[IDX2C(0, 1, 12)] = 0.873821971016996;
228             rules[IDX2C(0, 2, 12)] = 0.063089014491502;
229             rules[IDX2C(0, 3, 12)] = 0.000000000000000;
230
231             rules[IDX2C(1, 0, 12)] = 0.050844906370207;
232             rules[IDX2C(1, 1, 12)] = 0.063089014491502;
```

```
233             rules[IDX2C(1, 2, 12)] = 0.873821971016996;
234             rules[IDX2C(1, 3, 12)] = 0.000000000000000;
235
236             rules[IDX2C(2, 0, 12)] = 0.050844906370207;
237             rules[IDX2C(2, 1, 12)] = 0.063089014491502;
238             rules[IDX2C(2, 2, 12)] = 0.063089014491502;
239             rules[IDX2C(2, 3, 12)] = 0.000000000000000;
240
241             rules[IDX2C(3, 0, 12)] = 0.116786275726379;
242             rules[IDX2C(3, 1, 12)] = 0.501426509658179;
243             rules[IDX2C(3, 2, 12)] = 0.249286745170910;
244             rules[IDX2C(3, 3, 12)] = 0.000000000000000;
245
246             rules[IDX2C(4, 0, 12)] = 0.116786275726379;
247             rules[IDX2C(4, 1, 12)] = 0.249286745170910;
248             rules[IDX2C(4, 2, 12)] = 0.501426509658179;
249             rules[IDX2C(4, 3, 12)] = 0.000000000000000;
250
251             rules[IDX2C(5, 0, 12)] = 0.116786275726379;
252             rules[IDX2C(5, 1, 12)] = 0.249286745170910;
253             rules[IDX2C(5, 2, 12)] = 0.249286745170910;
254             rules[IDX2C(5, 3, 12)] = 0.000000000000000;
255
256             rules[IDX2C(6, 0, 12)] = 0.082851075618374;
257             rules[IDX2C(6, 1, 12)] = 0.636502499121399;
258             rules[IDX2C(6, 2, 12)] = 0.310352451033785;
259             rules[IDX2C(6, 3, 12)] = 0.000000000000000;
260
261             rules[IDX2C(7, 0, 12)] = 0.082851075618374;
262             rules[IDX2C(7, 1, 12)] = 0.636502499121399;
263             rules[IDX2C(7, 2, 12)] = 0.053145049844816;
264             rules[IDX2C(7, 3, 12)] = 0.000000000000000;
265
266             rules[IDX2C(8, 0, 12)] = 0.082851075618374;
267             rules[IDX2C(8, 1, 12)] = 0.310352451033785;
268             rules[IDX2C(8, 2, 12)] = 0.636502499121399;
269             rules[IDX2C(8, 3, 12)] = 0.000000000000000;
270
271             rules[IDX2C(9, 0, 12)] = 0.082851075618374;
272             rules[IDX2C(9, 1, 12)] = 0.310352451033785;
273             rules[IDX2C(9, 2, 12)] = 0.053145049844816;
274             rules[IDX2C(9, 3, 12)] = 0.000000000000000;
275
276             rules[IDX2C(10, 0, 12)] = 0.082851075618374;
277             rules[IDX2C(10, 1, 12)] = 0.053145049844816;
278             rules[IDX2C(10, 2, 12)] = 0.636502499121399;
279             rules[IDX2C(10, 3, 12)] = 0.000000000000000;
280
281             rules[IDX2C(11, 0, 12)] = 0.082851075618374;
282             rules[IDX2C(11, 1, 12)] = 0.053145049844816;
283             rules[IDX2C(11, 2, 12)] = 0.310352451033785;
284             rules[IDX2C(11, 3, 12)] = 0.000000000000000;
285         }
286     }
287     else if (integrationElementType == QUADRILATERAL)
288     {
289         if (iIntegrationOrder == 1)
290         {
291             rules[IDX2C(0, 0, 12)] = 2.000000000000000;
292             rules[IDX2C(0, 1, 12)] = 0.000000000000000;
293         }
294         else if (iIntegrationOrder == 2)
295         {
296             rules[IDX2C(0, 0, 12)] = 1.000000000000000;
297             rules[IDX2C(0, 1, 12)] = -0.577350269189626;
298
299             rules[IDX2C(1, 0, 12)] = 1.000000000000000;
300             rules[IDX2C(1, 1, 12)] = 0.577350269189626;
301         }
302         else if (iIntegrationOrder == 3)
303         {
304             rules[IDX2C(0, 0, 12)] = 0.888888888888888;
305             rules[IDX2C(0, 1, 12)] = 0.000000000000000;
306
307             rules[IDX2C(1, 0, 12)] = 0.555555555555555;
308             rules[IDX2C(1, 1, 12)] = -0.774596669241483;
```

```
309
310             rules[IDX2C(2, 0, 12)] = 0.555555555555555;
311             rules[IDX2C(2, 1, 12)] = 0.774596669241483;
312         }
313         else if (iIntegrationOrder == 4)
314         {
315             rules[IDX2C(0, 0, 12)] = 0.652145154862546;
316             rules[IDX2C(0, 1, 12)] = -0.339981043584856;
317
318             rules[IDX2C(1, 0, 12)] = 0.652145154862546;
319             rules[IDX2C(1, 1, 12)] = 0.339981043584856;
320
321             rules[IDX2C(2, 0, 12)] = 0.347854845137454;
322             rules[IDX2C(2, 1, 12)] = -0.861136311594053;
323
324             rules[IDX2C(3, 0, 12)] = 0.347854845137454;
325             rules[IDX2C(3, 1, 12)] = 0.861136311594053;
326         }
327         else if (iIntegrationOrder == 5)
328         {
329             rules[IDX2C(0, 0, 12)] = 0.568888888888888;
330             rules[IDX2C(0, 1, 12)] = 0.000000000000000;
331
332             rules[IDX2C(1, 0, 12)] = 0.478628670499366;
333             rules[IDX2C(1, 1, 12)] = -0.538469310105683;
334
335             rules[IDX2C(2, 0, 12)] = 0.478628670499366;
336             rules[IDX2C(2, 1, 12)] = 0.538469310105683;
337
338             rules[IDX2C(3, 0, 12)] = 0.236926885056189;
339             rules[IDX2C(3, 1, 12)] = -0.906179845938664;
340
341             rules[IDX2C(4, 0, 12)] = 0.236926885056189;
342             rules[IDX2C(4, 1, 12)] = 0.906179845938664;
343         }
344         else if (iIntegrationOrder == 6)
345         {
346             rules[IDX2C(0, 0, 12)] = 0.467913934572691;
347             rules[IDX2C(0, 1, 12)] = -0.238619186083197;
348
349             rules[IDX2C(1, 0, 12)] = 0.467913934572691;
350             rules[IDX2C(1, 1, 12)] = 0.238619186083197;
351
352             rules[IDX2C(2, 0, 12)] = 0.360761573048139;
353             rules[IDX2C(2, 1, 12)] = -0.661209386466265;
354
355             rules[IDX2C(3, 0, 12)] = 0.360761573048139;
356             rules[IDX2C(3, 1, 12)] = 0.661209386466265;
357
358             rules[IDX2C(4, 0, 12)] = 0.171324492379170;
359             rules[IDX2C(4, 1, 12)] = -0.932469514203152;
360
361             rules[IDX2C(5, 0, 12)] = 0.171324492379170;
362             rules[IDX2C(5, 1, 12)] = 0.932469514203152;
363         }
364         else if (iIntegrationOrder == 7)
365         {
366             rules[IDX2C(0, 0, 12)] = 0.417959183673469;
367             rules[IDX2C(0, 1, 12)] = 0.000000000000000;
368
369             rules[IDX2C(1, 0, 12)] = 0.381830050505119;
370             rules[IDX2C(1, 1, 12)] = -0.405845151377397;
371
372             rules[IDX2C(2, 0, 12)] = 0.381830050505119;
373             rules[IDX2C(2, 1, 12)] = 0.405845151377397;
374
375             rules[IDX2C(3, 0, 12)] = 0.279705391489277;
376             rules[IDX2C(3, 1, 12)] = -0.741531185599394;
377
378             rules[IDX2C(4, 0, 12)] = 0.279705391489277;
379             rules[IDX2C(4, 1, 12)] = 0.741531185599394;
380
381             rules[IDX2C(5, 0, 12)] = 0.129484966168870;
382             rules[IDX2C(5, 1, 12)] = -0.949107912342759;
383
384             rules[IDX2C(6, 0, 12)] = 0.129484966168870;
```

```
385             rules[IDX2C(6, 1, 12)] = 0.949107912342759;
386         }
387         else if (iIntegrationOrder == 8)
388         {
389             rules[IDX2C(0, 0, 12)] = 0.362683783378362;
390             rules[IDX2C(0, 1, 12)] = -0.183434642495650;
391
392             rules[IDX2C(1, 0, 12)] = 0.362683783378362;
393             rules[IDX2C(1, 1, 12)] = 0.183434642495650;
394
395             rules[IDX2C(2, 0, 12)] = 0.313706645877887;
396             rules[IDX2C(2, 1, 12)] = -0.525532409916329;
397
398             rules[IDX2C(3, 0, 12)] = 0.313706645877887;
399             rules[IDX2C(3, 1, 12)] = 0.525532409916329;
400
401             rules[IDX2C(4, 0, 12)] = 0.222381034453374;
402             rules[IDX2C(4, 1, 12)] = -0.796666477413627;
403
404             rules[IDX2C(5, 0, 12)] = 0.222381034453374;
405             rules[IDX2C(5, 1, 12)] = 0.796666477413627;
406
407             rules[IDX2C(6, 0, 12)] = 0.101228536290376;
408             rules[IDX2C(6, 1, 12)] = -0.960289856497536;
409
410             rules[IDX2C(7, 0, 12)] = 0.101228536290376;
411             rules[IDX2C(7, 1, 12)] = 0.960289856497536;
412         }
413         else if (iIntegrationOrder == 9)
414         {
415             rules[IDX2C(0, 0, 12)] = 0.330239355001260;
416             rules[IDX2C(0, 1, 12)] = 0.000000000000000;
417
418             rules[IDX2C(1, 0, 12)] = 0.312347077040003;
419             rules[IDX2C(1, 1, 12)] = -0.324253423403809;
420
421             rules[IDX2C(2, 0, 12)] = 0.312347077040003;
422             rules[IDX2C(2, 1, 12)] = 0.324253423403809;
423
424             rules[IDX2C(3, 0, 12)] = 0.260610696402935;
425             rules[IDX2C(3, 1, 12)] = -0.613371432700590;
426
427             rules[IDX2C(4, 0, 12)] = 0.260610696402935;
428             rules[IDX2C(4, 1, 12)] = 0.613371432700590;
429
430             rules[IDX2C(5, 0, 12)] = 0.180648160694857;
431             rules[IDX2C(5, 1, 12)] = -0.836031107326636;
432
433             rules[IDX2C(6, 0, 12)] = 0.180648160694857;
434             rules[IDX2C(6, 1, 12)] = 0.836031107326636;
435
436             rules[IDX2C(7, 0, 12)] = 0.081274388361574;
437             rules[IDX2C(7, 1, 12)] = -0.968160239507626;
438
439             rules[IDX2C(8, 0, 12)] = 0.081274388361574;
440             rules[IDX2C(8, 1, 12)] = 0.968160239507626;
441         }
442         else if (iIntegrationOrder == 10)
443         {
444             rules[IDX2C(0, 0, 12)] = 0.295524224714753;
445             rules[IDX2C(0, 1, 12)] = -0.148874338981631;
446
447             rules[IDX2C(1, 0, 12)] = 0.295524224714753;
448             rules[IDX2C(1, 1, 12)] = 0.148874338981631;
449
450             rules[IDX2C(2, 0, 12)] = 0.269266719309996;
451             rules[IDX2C(2, 1, 12)] = -0.433395394129247;
452
453             rules[IDX2C(3, 0, 12)] = 0.269266719309996;
454             rules[IDX2C(3, 1, 12)] = 0.433395394129247;
455
456             rules[IDX2C(4, 0, 12)] = 0.219086362515982;
457             rules[IDX2C(4, 1, 12)] = -0.679409568299024;
458
459             rules[IDX2C(5, 0, 12)] = 0.219086362515982;
460             rules[IDX2C(5, 1, 12)] = 0.679409568299024;
```

```
461
462             rules[IDX2C(6, 0, 12)] = 0.149451349150581;
463             rules[IDX2C(6, 1, 12)] = -0.865063366688985;
464
465             rules[IDX2C(7, 0, 12)] = 0.149451349150581;
466             rules[IDX2C(7, 1, 12)] = 0.865063366688985;
467
468             rules[IDX2C(8, 0, 12)] = 0.066671344308688;
469             rules[IDX2C(8, 1, 12)] = -0.973906528517172;
470
471             rules[IDX2C(9, 0, 12)] = 0.066671344308688;
472             rules[IDX2C(9, 1, 12)] = 0.973906528517172;
473         }
474         else if (iIntegrationOrder == 12)
475         {
476             rules[IDX2C(0, 0, 12)] = 0.249147045813403;
477             rules[IDX2C(0, 1, 12)] = -0.125333408511469;
478
479             rules[IDX2C(1, 0, 12)] = 0.249147045813403;
480             rules[IDX2C(1, 1, 12)] = 0.125333408511469;
481
482             rules[IDX2C(2, 0, 12)] = 0.233492536538355;
483             rules[IDX2C(2, 1, 12)] = -0.367831498918180;
484
485             rules[IDX2C(3, 0, 12)] = 0.233492536538355;
486             rules[IDX2C(3, 1, 12)] = 0.367831498918180;
487
488             rules[IDX2C(4, 0, 12)] = 0.203167426723066;
489             rules[IDX2C(4, 1, 12)] = -0.587317954286617;
490
491             rules[IDX2C(5, 0, 12)] = 0.203167426723066;
492             rules[IDX2C(5, 1, 12)] = 0.587317954286617;
493
494             rules[IDX2C(6, 0, 12)] = 0.160078328543346;
495             rules[IDX2C(6, 1, 12)] = -0.769902674194305;
496
497             rules[IDX2C(7, 0, 12)] = 0.160078328543346;
498             rules[IDX2C(7, 1, 12)] = 0.769902674194305;
499
500             rules[IDX2C(8, 0, 12)] = 0.106939325995318;
501             rules[IDX2C(8, 1, 12)] = -0.904117256370475;
502
503             rules[IDX2C(9, 0, 12)] = 0.106939325995318;
504             rules[IDX2C(9, 1, 12)] = 0.904117256370475;
505
506             rules[IDX2C(10, 0, 12)] = 0.047175336386512;
507             rules[IDX2C(10, 1, 12)] = -0.981560634246719;
508
509             rules[IDX2C(11, 0, 12)] = 0.047175336386512;
510             rules[IDX2C(11, 1, 12)] = 0.981560634246719;
511         }
512     }
513 }
514
515 #endif
```

- *CUDA_MeshCFD.cuh*

```
1  /*****************************************************************************/
2  //                                HPC - LMC                                  //
3  //---------------------------------------------------------------------------//
4  /*!
5   * \brief   Classe CUDA_MeshCFD
6   *
7   * Esta classe é responsável pelas operações com os elementos do tipo CFD
8   *
9   * \author  Luiz Felipe Couto
10  */
11 ////////////////////////////////////////////////////////////////////////////////
12
13 #ifndef HPC_LMC_FEM_CUDA_FEM_CFD_CUDA_MESHCFD_H
14 #define HPC_LMC_FEM_CUDA_FEM_CFD_CUDA_MESHCFD_H
15
```

```
16 //-----------------------------------------------------------------------------
17 // Include and Forward declaration
18 //-----------------------------------------------------------------------------
19 #include "../../../core/cuda/CUDA_NumericalIntegration.cuh"
20
21 #include "../../cuda/CUDA_Mesh.cuh"
22
23 //-----------------------------------------------------------------------------
24 // Prototype definition
25 //-----------------------------------------------------------------------------
26 __device__
27 void calculate_k_g_m_e( MeshCFDType     meshCFDType,
28                         size_t          iNumberDimensions,
29                         Precision*      nodesCoordinates,
30                         Precision       RHO,
31                         Precision       MI,
32                         unsigned int*   integrationOrder_XSI,
33                         unsigned int*   integrationOrder_ETA,
34                         unsigned int*   integrationOrder_ZETA,
35                         Precision*      k_e,
36                         Precision*      g_e,
37                         Precision*      m_e);
38
39 __device__
40 void calculate_c_c_t_e( MeshCFDType     meshCFDType,
41                         size_t          iNumberDimensions,
42                         Precision*      nodesCoordinates,
43                         Precision*      u_e,
44                         Precision*      v_e,
45                         Precision*      w_e,
46                         unsigned int*   integrationOrder_XSI,
47                         unsigned int*   integrationOrder_ETA,
48                         unsigned int*   integrationOrder_ZETA,
49                         Precision*      c_e,
50                         Precision*      c_t_e);
51
52 ////////////////////////////////////////////////////////////////////////////////
53
54 __device__
55 void integrateElement_k_g_m(    MeshCFDType meshCFDType,
56                                MatrixType  matrixType,
57                                size_t      iNumberDimensions,
58                                size_t      iNumberNodes_u,
59                                size_t      iNumberNodes_p,
60                                Precision*  nodesCoordinates,
61                                Precision   RHO,
62                                Precision   MI,
63                                size_t      iIntegrationOrder_XSI,
64                                size_t      iIntegrationOrder_ETA,
65                                size_t      iIntegrationOrder_ZETA,
66                                Precision*  Nu,
67                                Precision*  Np,
68                                Precision*  Nu_DOF,
69                                Precision*  Nu_DOF_T,
70                                Precision*  Nu_DOF_T_Nu_DOF,
71                                Precision*  NuDerivatives,
72                                Precision*  div_Nu,
73                                Precision*  div_Nu_T,
74                                Precision*  div_Nu_T_Np,
75                                Precision*  Ju,
76                                Precision*  JuInverse,
77                                Precision*  NuDerivativesTransformed,
78                                Precision*  Bu,
79                                Precision*  Bu_T,
80                                Precision*  C,
81                                Precision*  Bu_T_C,
82                                Precision*  Bu_T_C_Bu,
83                                Precision   JuDeterminant,
84                                Precision*  k_e_int,
85                                Precision*  k_e_int_Temp,
86                                Precision*  g_e_int,
87                                Precision*  g_e_int_Temp,
88                                Precision*  m_e_int,
89                                Precision*  m_e_int_Temp);
90
91 __device__
```

```
92  void integrateElement_c_c_t(MeshCFDType meshCFDType,
93                              MatrixType  matrixType,
94                              size_t      iNumberDimensions,
95                              size_t      iNumberNodes_u,
96                              Precision*  nodesCoordinates,
97                              Precision*  u_e,
98                              Precision*  v_e,
99                              Precision*  w_e,
100                             size_t      iIntegrationOrder_XSI,
101                             size_t      iIntegrationOrder_ETA,
102                             size_t      iIntegrationOrder_ZETA,
103                             Precision*  Nu,
104                             Precision*  NuDerivatives,
105                             Precision*  Ju,
106                             Precision*  JuInverse,
107                             Precision*  NuDerivativesTransformed,
108                             Precision*  B,
109                             Precision*  Nu_DOF,
110                             Precision*  Nu_DOF_T,
111                             Precision*  e_dot_u,
112                             Precision*  Nu_DOF_T_e_dot_u,
113                             Precision*  NuDerivatives_x_DOF,
114                             Precision*  NuDerivatives_y_DOF,
115                             Precision*  NuDerivatives_x_DOF_NuDerivatives_y_DOF,
116                             Precision*  Nu_DOF_T_NuDerivatives_x_DOF_NuDerivatives_y_DOF,
117                             Precision*  gradu,
118                             Precision*  Nu_DOF_T_gradu,
119                             Precision*  Nu_DOF_T_gradu_Nu_DOF,
120                             Precision   JuDeterminant,
121                             Precision*  c_e_int,
122                             Precision*  c_e_int_Temp,
123                             Precision*  c_t_e_int,
124                             Precision*  c_t_e_int_Temp);
125
126 ////////////////////////////////////////////////////////////////////////////////
127
128 __device__
129 void do_calculate_k_g_m_e(  MeshCFDType meshCFDType,
130                             MatrixType  matrixType,
131                             size_t      iNumberDimensions,
132                             size_t      iNumberNodes_u,
133                             size_t      iNumberNodes_p,
134                             Precision*  nodesCoordinates,
135                             Precision   RHO,
136                             Precision   MI,
137                             Precision   d_XSI,
138                             Precision   d_ETA,
139                             Precision   d_ZETA,
140                             Precision*  Nu,
141                             Precision*  Np,
142                             Precision*  Nu_DOF,
143                             Precision*  Nu_DOF_T,
144                             Precision*  Nu_DOF_T_Nu_DOF,
145                             Precision*  NuDerivatives,
146                             Precision*  div_Nu,
147                             Precision*  div_Nu_T,
148                             Precision*  div_Nu_T_Np,
149                             Precision*  Ju,
150                             Precision*  JuInverse,
151                             Precision*  NuDerivativesTransformed,
152                             Precision*  Bu,
153                             Precision*  Bu_T,
154                             Precision*  C,
155                             Precision*  Bu_T_C,
156                             Precision*  Bu_T_C_Bu,
157                             Precision   JuDeterminant,
158                             Precision*  k_e,
159                             Precision*  g_e,
160                             Precision*  m_e);
161
162 __device__
163 void do_calculate_c_c_t_e(  MeshCFDType meshCFDType,
164                             MatrixType  matrixType,
165                             size_t      iNumberDimensions,
166                             size_t      iNumberNodes_u,
167                             Precision*  nodesCoordinates,
```

```
                                Precision*  u_e,
                                Precision*  v_e,
                                Precision*  w_e,
                                Precision   d_XSI,
                                Precision   d_ETA,
                                Precision   d_ZETA,
                                Precision*  Nu,
                                Precision*  NuDerivatives,
                                Precision*  Ju,
                                Precision*  JuInverse,
                                Precision*  NuDerivativesTransformed,
                                Precision*  B,
                                Precision*  Nu_DOF,
                                Precision*  Nu_DOF_T,
                                Precision*  e_dot_u,
                                Precision*  Nu_DOF_T_e_dot_u,
                                Precision*  NuDerivatives_x_DOF,
                                Precision*  NuDerivatives_y_DOF,
                                Precision*  NuDerivatives_x_DOF_NuDerivatives_y_DOF,
                                Precision*  Nu_DOF_T_NuDerivatives_x_DOF_NuDerivatives_y_DOF,
                                Precision*  gradu,
                                Precision*  Nu_DOF_T_gradu,
                                Precision*  Nu_DOF_T_gradu_Nu_DOF,
                                Precision   JuDeterminant,
                                Precision*  c_e_int,
                                Precision*  c_t_e_int);

////////////////////////////////////////////////////////////////////////////////

__device__
void do_calculate_k_e(  size_t      iNumberDimensions,
                        size_t      iNumberNodes_u,
                        Precision   RHO,
                        Precision   MI,
                        Precision*  NuDerivativesTransformed,
                        Precision*  Bu,
                        Precision*  Bu_T,
                        Precision*  C,
                        Precision*  Bu_T_C,
                        Precision*  Bu_T_C_Bu,
                        Precision   JuDeterminant,
                        Precision*  k_e);

__device__
void do_calculate_g_e(  size_t      iNumberDimensions,
                        size_t      iNumberNodes_u,
                        size_t      iNumberNodes_p,
                        Precision*  Np,
                        Precision*  NuDerivatives,
                        Precision*  Ju,
                        Precision*  NuDerivativesTransformed,
                        Precision*  div_Nu,
                        Precision*  div_Nu_T,
                        Precision*  div_Nu_T_Np,
                        Precision   JuDeterminant,
                        Precision*  g_e);

__device__
void do_calculate_m_e(  size_t      iNumberDimensions,
                        size_t      iNumberNodes_u,
                        Precision   RHO,
                        Precision*  Nu,
                        Precision*  Nu_DOF,
                        Precision*  Nu_DOF_T,
                        Precision*  Nu_DOF_T_Nu_DOF,
                        Precision   JuDeterminant,
                        Precision*  m_e);

__device__
void do_calculate_c_e(  size_t      iNumberDimensions,
                        size_t      iNumberNodes_u,
                        Precision*  u_e,
                        Precision*  v_e,
                        Precision*  w_e,
                        Precision*  Nu,
                        Precision*  NuDerivativesTransformed,
```

```
                        Precision*  B,
                        Precision*  Nu_DOF,
                        Precision*  Nu_DOF_T,
                        Precision*  e_dot_u,
                        Precision*  Nu_DOF_T_e_dot_u,
                        Precision   JuDeterminant,
                        Precision*  c_e_int);

__device__
void do_calculate_c_t_e(size_t      iNumberDimensions,
                        size_t      iNumberNodes_u,
                        Precision*  u_e,
                        Precision*  v_e,
                        Precision*  w_e,
                        Precision*  Nu,
                        Precision*  NuDerivativesTransformed,
                        Precision*  B,
                        Precision*  Nu_DOF,
                        Precision*  Nu_DOF_T,
                        Precision*  NuDerivatives_x_DOF,
                        Precision*  NuDerivatives_y_DOF,
                        Precision*  NuDerivatives_x_DOF_NuDerivatives_y_DOF,
                        Precision*  Nu_DOF_T_NuDerivatives_x_DOF_NuDerivatives_y_DOF,
                        Precision*  gradu,
                        Precision*  Nu_DOF_T_gradu,
                        Precision*  Nu_DOF_T_gradu_Nu_DOF,
                        Precision   JuDeterminant,
                        Precision*  c_t_e_int);

__device__
void calculate_Nu(  MeshCFDType meshCFDType,
                    Precision   d_XSI,
                    Precision   d_ETA,
                    Precision   d_ZETA,
                    Precision*   Nu);

__device__
void calculate_Np(  MeshCFDType meshCFDType,
                    Precision   d_XSI,
                    Precision   d_ETA,
                    Precision   d_ZETA,
                    Precision*  Np);

__device__
void calculate_NuDerivatives(   MeshCFDType meshCFDType,
                                Precision   d_XSI,
                                Precision   d_ETA,
                                Precision   d_ZETA,
                                Precision*  NuDerivatives);

__device__
void calculate_divNu(   size_t      iNumberDimensions,
                        size_t      iNumberNodes_u,
                        Precision*  NuDerivatives,
                        Precision*  divNu);

////////////////////////////////////////////////////////////////////////////////

__device__
void calculate_C(   size_t iNumberDimensions,
                    Precision RHO,
                    Precision MI,
                    Precision* C);

////////////////////////////////////////////////////////////////////////////////

__device__
void calculate_e_dot_u( size_t      iNumberDimensions,
                        size_t      iNumberNodes_u,
                        Precision*  u_e,
                        Precision*  v_e,
                        Precision*  w_e,
                        Precision*  Nu,
                        Precision*  e_dot_u);
```

```
320 __device__
321 void calculate_gradu(   size_t      iNumberDimensions,
322                         size_t      iNumberNodes_u,
323                         Precision*  u_e,
324                         Precision*  v_e,
325                         Precision*  w_e,
326                         Precision*  NuDerivativesTransformed,
327                         Precision*  gradu);
328
329 //-----------------------------------------------------------------------------
330 // Device Function
331 //-----------------------------------------------------------------------------
332
333 //------------------------------ calculate_k_g_m_e ----------------------------
334 __device__
335 void calculate_k_g_m_e( MeshCFDType     meshCFDType,
336                         size_t          iNumberDimensions,
337                         Precision*      nodesCoordinates,
338                         Precision       RHO,
339                         Precision       MI,
340                         unsigned int*   integrationOrder_XSI,
341                         unsigned int*   integrationOrder_ETA,
342                         unsigned int*   integrationOrder_ZETA,
343                         Precision*      k_e,
344                         Precision*      g_e,
345                         Precision*      m_e)
346 {
347     if (meshCFDType == MCFDT_P2P1)
348     {
349         if (iNumberDimensions == 2)
350         {
351             const size_t NBR_DIMENSIONS = 2;
352             const size_t NBR_NODES_U    = 6;
353             const size_t NBR_NODES_P    = 3;
354
355             Precision Nu[1 * NBR_NODES_U]                                      = { 0. };
356             Precision Np[1 * NBR_NODES_P]                                      = { 0. };
357             Precision Nu_DOF[NBR_DIMENSIONS * NBR_DIMENSIONS * NBR_NODES_U]    = { 0. };
358             Precision Nu_DOF_T[NBR_DIMENSIONS * NBR_NODES_U * NBR_DIMENSIONS]  = { 0. };
359             Precision Nu_DOF_T_Nu_DOF[NBR_DIMENSIONS * NBR_NODES_U *
360                 NBR_DIMENSIONS * NBR_NODES_U] = { 0. };
361             Precision NuDerivatives[NBR_DIMENSIONS * NBR_NODES_U]              = { 0. };
362             Precision div_Nu[1 * NBR_DIMENSIONS * NBR_NODES_U]                 = { 0. };
363             Precision div_Nu_T[NBR_DIMENSIONS * NBR_NODES_U * 1]               = { 0. };
364             Precision div_Nu_T_Np[NBR_DIMENSIONS * NBR_NODES_U * NBR_NODES_P]  = { 0. };
365
366             Precision Ju[NBR_DIMENSIONS * NBR_DIMENSIONS]                      = { 0. };
367             Precision JuDeterminant                                            = 0.;
368             Precision JuInverse[NBR_DIMENSIONS * NBR_DIMENSIONS]               = { 0. };
369             Precision NuDerivativesTransformed[NBR_DIMENSIONS * NBR_NODES_U]   = { 0. };
370
371             Precision Bu[3 * NBR_DIMENSIONS * NBR_NODES_U]                     = { 0. };
372             Precision Bu_T[NBR_DIMENSIONS * NBR_NODES_U * 3]                   = { 0. };
373             Precision C[3 * 3]                                                 = { 0. };
374             Precision Bu_T_C[NBR_DIMENSIONS * NBR_NODES_U * 3]                 = { 0. };
375             Precision Bu_T_C_Bu[NBR_DIMENSIONS * NBR_NODES_U * NBR_DIMENSIONS *
376                 NBR_NODES_U] = { 0. };
377
378             Precision k_e_int[NBR_DIMENSIONS * NBR_NODES_U * NBR_DIMENSIONS *
379                 NBR_NODES_U] = { 0. };
380             Precision g_e_int[NBR_DIMENSIONS * NBR_NODES_U * NBR_NODES_P]      = { 0. };
381             Precision g_e_int_T[NBR_NODES_P * NBR_DIMENSIONS * NBR_NODES_U]    = { 0. };
382             Precision m_e_int[NBR_DIMENSIONS * NBR_NODES_U * NBR_DIMENSIONS *
383                 NBR_NODES_U] = { 0. };
384
385             Precision k_e_int_Temp[NBR_DIMENSIONS * NBR_NODES_U * NBR_DIMENSIONS *
386                 NBR_NODES_U] = { 0. };
387             Precision g_e_int_Temp[NBR_DIMENSIONS * NBR_NODES_U * NBR_NODES_P]  = { 0. };
388             Precision m_e_int_Temp[NBR_DIMENSIONS * NBR_NODES_U * NBR_DIMENSIONS *
389                 NBR_NODES_U] = { 0. };
390
391             integrateElement_k_g_m( meshCFDType,                MT_k_e,
392                                     NBR_DIMENSIONS,             NBR_NODES_U,
393                                     NBR_NODES_P,                nodesCoordinates,
394                                     RHO,                        MI,
395                                     integrationOrder_XSI[0],    integrationOrder_ETA[0],
```

```
                                    integrationOrder_ZETA[0],   Nu,
                                    Np,                         Nu_DOF,
                                    Nu_DOF_T,                   Nu_DOF_T_Nu_DOF,
                                    NuDerivatives,              div_Nu,
                                    div_Nu_T,                   div_Nu_T_Np,
                                    Ju,                         JuInverse,
                                    NuDerivativesTransformed,   Bu,
                                    Bu_T,                       C,
                                    Bu_T_C,                     Bu_T_C_Bu,
                                    JuDeterminant,              k_e_int,
                                    k_e_int_Temp,               g_e_int,
                                    g_e_int_Temp,               m_e_int,
                                    m_e_int_Temp);
            block(  0,                              0,
                    NBR_DIMENSIONS * NBR_NODES_U,   NBR_DIMENSIONS * NBR_NODES_U,
                    k_e_int,                        k_e,
                    NBR_DIMENSIONS * NBR_NODES_U + NBR_NODES_P);

            integrateElement_k_g_m( meshCFDType,                MT_g_e,
                                    NBR_DIMENSIONS,             NBR_NODES_U,
                                    NBR_NODES_P,                nodesCoordinates,
                                    RHO,                        MI,
                                    integrationOrder_XSI[1],    integrationOrder_ETA[1],
                                    integrationOrder_ZETA[1],   Nu,
                                    Np,                         Nu_DOF,
                                    Nu_DOF_T,                   Nu_DOF_T_Nu_DOF,
                                    NuDerivatives,              div_Nu,
                                    div_Nu_T,                   div_Nu_T_Np,
                                    Ju,                         JuInverse,
                                    NuDerivativesTransformed,   Bu,
                                    Bu_T,                       C,
                                    Bu_T_C,                     Bu_T_C_Bu,
                                    JuDeterminant,              k_e_int,
                                    k_e_int_Temp,               g_e_int,
                                    g_e_int_Temp,               m_e_int,
                                    m_e_int_Temp);
            transpose(NBR_DIMENSIONS * NBR_NODES_U, NBR_NODES_P, g_e_int, g_e_int_T);
            block(  0,                              NBR_DIMENSIONS * NBR_NODES_U,
                    NBR_DIMENSIONS * NBR_NODES_U,   NBR_NODES_P,
                    g_e_int,                        g_e,
                    NBR_DIMENSIONS * NBR_NODES_U + NBR_NODES_P);
            block(  NBR_DIMENSIONS * NBR_NODES_U,   0,
                    NBR_NODES_P,                    NBR_DIMENSIONS * NBR_NODES_U,
                    g_e_int_T,                      g_e,
                    NBR_DIMENSIONS * NBR_NODES_U + NBR_NODES_P);

            integrateElement_k_g_m( meshCFDType,                MT_m_e,
                                    NBR_DIMENSIONS,             NBR_NODES_U,
                                    NBR_NODES_P,                nodesCoordinates,
                                    RHO,                        MI,
                                    integrationOrder_XSI[2],    integrationOrder_ETA[2],
                                    integrationOrder_ZETA[2],   Nu,
                                    Np,                         Nu_DOF,
                                    Nu_DOF_T,                   Nu_DOF_T_Nu_DOF,
                                    NuDerivatives,              div_Nu,
                                    div_Nu_T,                   div_Nu_T_Np,
                                    Ju,                         JuInverse,
                                    NuDerivativesTransformed,   Bu,
                                    Bu_T,                       C,
                                    Bu_T_C,                     Bu_T_C_Bu,
                                    JuDeterminant,              k_e_int,
                                    k_e_int_Temp,               g_e_int,
                                    g_e_int_Temp,               m_e_int,
                                    m_e_int_Temp);
            block(  0,                              0,
                    NBR_DIMENSIONS * NBR_NODES_U,   NBR_DIMENSIONS * NBR_NODES_U,
                    m_e_int,                        m_e,
                    NBR_DIMENSIONS * NBR_NODES_U + NBR_NODES_P);
        }
    }
}

//----------------------------- calculate_k_g_m_e ----------------------------
__device__
void calculate_c_c_t_e( MeshCFDType     meshCFDType,
                        size_t          iNumberDimensions,
```

```
472                         Precision*      nodesCoordinates,
473                         Precision*      u_e,
474                         Precision*      v_e,
475                         Precision*      w_e,
476                         unsigned int*   integrationOrder_XSI,
477                         unsigned int*   integrationOrder_ETA,
478                         unsigned int*   integrationOrder_ZETA,
479                         Precision*      c_e,
480                         Precision*      c_t_e)
481 {
482     if (meshCFDType == MCFDT_P2P1)
483     {
484         if (iNumberDimensions == 2)
485         {
486             const size_t NBR_DIMENSIONS = 2;
487             const size_t NBR_NODES_U    = 6;
488             const size_t NBR_NODES_P    = 3;
489 
490             Precision Nu[1 * NBR_NODES_U] = { 0. };
491             Precision Nu_DOF[NBR_DIMENSIONS * NBR_DIMENSIONS * NBR_NODES_U]     = { 0. };
492             Precision Nu_DOF_T[NBR_DIMENSIONS * NBR_NODES_U * NBR_DIMENSIONS]   = { 0. };
493             Precision NuDerivatives[NBR_DIMENSIONS * NBR_NODES_U]               = { 0. };
494 
495             Precision Ju[NBR_DIMENSIONS * NBR_DIMENSIONS]                       = { 0. };
496             Precision JuDeterminant                                             = 0.;
497             Precision JuInverse[NBR_DIMENSIONS * NBR_DIMENSIONS]                = { 0. };
498             Precision NuDerivativesTransformed[NBR_DIMENSIONS * NBR_NODES_U]    = { 0. };
499 
500             Precision B[NBR_DIMENSIONS * NBR_DIMENSIONS * NBR_DIMENSIONS *
501                 NBR_NODES_U] = { 0. };
502 
503             Precision e_dot_u[NBR_DIMENSIONS * NBR_DIMENSIONS * NBR_DIMENSIONS] = { 0. };
504             Precision Nu_DOF_T_e_dot_u[NBR_DIMENSIONS * NBR_NODES_U * NBR_DIMENSIONS *
505                 NBR_DIMENSIONS] = { 0. };
506 
507             Precision NuDerivatives_x_DOF[NBR_DIMENSIONS * NBR_DIMENSIONS *
508                 NBR_NODES_U] = { 0. };
509             Precision NuDerivatives_y_DOF[NBR_DIMENSIONS * NBR_DIMENSIONS *
510                 NBR_NODES_U] = { 0. };
511             Precision NuDerivatives_x_DOF_NuDerivatives_y_DOF[NBR_DIMENSIONS *
512                 NBR_DIMENSIONS * NBR_NODES_U] = { 0. };
513             Precision Nu_DOF_T_NuDerivatives_x_DOF_NuDerivatives_y_DOF[NBR_DIMENSIONS *
514                 NBR_NODES_U * NBR_DIMENSIONS * NBR_NODES_U] = { 0. };
515 
516             Precision gradu[NBR_DIMENSIONS * NBR_DIMENSIONS] = { 0. };
517             Precision Nu_DOF_T_gradu[NBR_DIMENSIONS * NBR_NODES_U *
518                 NBR_DIMENSIONS] = { 0. };
519             Precision Nu_DOF_T_gradu_Nu_DOF[NBR_DIMENSIONS * NBR_NODES_U *
520                 NBR_DIMENSIONS * NBR_NODES_U] = { 0. };
521 
522             Precision c_e_int[NBR_DIMENSIONS * NBR_NODES_U * NBR_DIMENSIONS *
523                 NBR_NODES_U] = { 0. };
524             Precision c_e_int_Temp[NBR_DIMENSIONS * NBR_NODES_U * NBR_DIMENSIONS *
525                 NBR_NODES_U] = { 0. };
526             Precision c_t_e_int[NBR_DIMENSIONS * NBR_NODES_U * NBR_DIMENSIONS *
527                 NBR_NODES_U] = { 0. };
528             Precision c_t_e_int_Temp[NBR_DIMENSIONS * NBR_NODES_U * NBR_DIMENSIONS *
529                 NBR_NODES_U] = { 0. };
530 
531             integrateElement_c_c_t( meshCFDType,                MT_c_e,
532                                     NBR_DIMENSIONS,             NBR_NODES_U,
533                                     nodesCoordinates,           u_e,
534                                     v_e,                        w_e,
535                                     integrationOrder_XSI[3],    integrationOrder_ETA[3],
536                                     integrationOrder_ZETA[3],   Nu,
537                                     NuDerivatives,              Ju,
538                                     JuInverse,                  NuDerivativesTransformed,
539                                     B,                          Nu_DOF,
540                                     Nu_DOF_T,                   e_dot_u,
541                                     Nu_DOF_T_e_dot_u,           NuDerivatives_x_DOF,
542                                     NuDerivatives_y_DOF,
543                                     NuDerivatives_x_DOF_NuDerivatives_y_DOF,
544                                     Nu_DOF_T_NuDerivatives_x_DOF_NuDerivatives_y_DOF,
545                                     gradu,                      Nu_DOF_T_gradu,
546                                     Nu_DOF_T_gradu_Nu_DOF,      JuDeterminant,
547                                     c_e_int,                    c_e_int_Temp,
```

```
                                    c_t_e_int,                  c_t_e_int_Temp);
            block(  0,                              0,
                    NBR_DIMENSIONS * NBR_NODES_U,   NBR_DIMENSIONS * NBR_NODES_U,
                    c_e_int,                        c_e,
                    NBR_DIMENSIONS * NBR_NODES_U + NBR_NODES_P);

            integrateElement_c_c_t( meshCFDType,                MT_c_t_e,
                                    NBR_DIMENSIONS,             NBR_NODES_U,
                                    nodesCoordinates,           u_e,
                                    v_e,                        w_e,
                                    integrationOrder_XSI[4],    integrationOrder_ETA[4],
                                    integrationOrder_ZETA[4],   Nu,
                                    NuDerivatives,              Ju,
                                    JuInverse,                  NuDerivativesTransformed,
                                    B,                          Nu_DOF,
                                    Nu_DOF_T,                   e_dot_u,
                                    Nu_DOF_T_e_dot_u,           NuDerivatives_x_DOF,
                                    NuDerivatives_y_DOF,
                                    NuDerivatives_x_DOF_NuDerivatives_y_DOF,
                                    Nu_DOF_T_NuDerivatives_x_DOF_NuDerivatives_y_DOF,
                                    gradu,                      Nu_DOF_T_gradu,
                                    Nu_DOF_T_gradu_Nu_DOF,      JuDeterminant,
                                    c_e_int,                    c_e_int_Temp,
                                    c_t_e_int,                  c_t_e_int_Temp);
            block(  0,                              0,
                    NBR_DIMENSIONS * NBR_NODES_U,   NBR_DIMENSIONS * NBR_NODES_U,
                    c_t_e_int,                      c_t_e,
                    NBR_DIMENSIONS * NBR_NODES_U + NBR_NODES_P);
        }
    }
}

////////////////////////////////////////////////////////////////////////////////

//----------------------------- integrateElement_k_g_m -----------------------
__device__
void integrateElement_k_g_m(    MeshCFDType meshCFDType,
                                MatrixType  matrixType,
                                size_t      iNumberDimensions,
                                size_t      iNumberNodes_u,
                                size_t      iNumberNodes_p,
                                Precision*  nodesCoordinates,
                                Precision   RHO,
                                Precision   MI,
                                size_t      iIntegrationOrder_XSI,
                                size_t      iIntegrationOrder_ETA,
                                size_t      iIntegrationOrder_ZETA,
                                Precision*  Nu,
                                Precision*  Np,
                                Precision*  Nu_DOF,
                                Precision*  Nu_DOF_T,
                                Precision*  Nu_DOF_T_Nu_DOF,
                                Precision*  NuDerivatives,
                                Precision*  div_Nu,
                                Precision*  div_Nu_T,
                                Precision*  div_Nu_T_Np,
                                Precision*  Ju,
                                Precision*  JuInverse,
                                Precision*  NuDerivativesTransformed,
                                Precision*  Bu,
                                Precision*  Bu_T,
                                Precision*  C,
                                Precision*  Bu_T_C,
                                Precision*  Bu_T_C_Bu,
                                Precision   JuDeterminant,
                                Precision*  k_e_int,
                                Precision*  k_e_int_Temp,
                                Precision*  g_e_int,
                                Precision*  g_e_int_Temp,
                                Precision*  m_e_int,
                                Precision*  m_e_int_Temp)
{
    IntegrationElementType integrationElementType;

    if (meshCFDType == MCFDT_P2P1)
    {
```

```
624         integrationElementType = TRIANGLE;
625     }
626     else if (meshCFDType == MCFDT_Q2Q1)
627     {
628         integrationElementType = QUADRILATERAL;
629     }
630 
631     Precision integrationRules_XSI[12 * 4];
632     Precision integrationRules_ETA[12 * 4];
633     Precision integrationRules_ZETA[12 * 4];
634 
635     if (iIntegrationOrder_XSI > 0)
636     {
637         getIntegrationRules(integrationElementType,
638                             iIntegrationOrder_XSI,
639                             integrationRules_XSI);
640     }
641 
642     if (iIntegrationOrder_ETA > 0)
643     {
644         getIntegrationRules(integrationElementType,
645                             iIntegrationOrder_ETA,
646                             integrationRules_ETA);
647     }
648 
649     if (iIntegrationOrder_ZETA > 0)
650     {
651         getIntegrationRules(integrationElementType,
652                             iIntegrationOrder_ZETA,
653                             integrationRules_ZETA);
654     }
655 
656     Precision d_XSI     = 0.;
657     Precision d_ETA     = 0.;
658     Precision d_ZETA    = 0.;
659 
660     Precision d_W   = 0.;
661     Precision d_W_i = 0.;
662     Precision d_W_j = 0.;
663     Precision d_W_k = 0.;
664 
665     //Calcula a integral 1D
666     for (size_t i = 0; i < iIntegrationOrder_XSI; ++i)
667     {
668         d_W_i = integrationRules_XSI[IDX2C(i, 0, 12)];
669 
670         if (integrationElementType == TRIANGLE)
671         {
672             d_XSI   = integrationRules_XSI[IDX2C(i, 1, 12)];
673             d_ETA   = integrationRules_XSI[IDX2C(i, 2, 12)];
674             d_ZETA  = integrationRules_XSI[IDX2C(i, 3, 12)];
675         }
676         else if (integrationElementType == QUADRILATERAL)
677         {
678             d_XSI = integrationRules_XSI[IDX2C(i, 1, 12)];
679         }
680 
681         d_W = d_W_i;
682         do_calculate_k_g_m_e(   meshCFDType,                matrixType,
683                                 iNumberDimensions,          iNumberNodes_u,
684                                 iNumberNodes_p,             nodesCoordinates,
685                                 RHO,                        MI,
686                                 d_XSI,                      d_ETA,
687                                 d_ZETA,                     Nu,
688                                 Np,                         Nu_DOF,
689                                 Nu_DOF_T,                   Nu_DOF_T_Nu_DOF,
690                                 NuDerivatives,              div_Nu,
691                                 div_Nu_T,                   div_Nu_T_Np,
692                                 Ju,                         JuInverse,
693                                 NuDerivativesTransformed,   Bu,
694                                 Bu_T,                       C,
695                                 Bu_T_C,                     Bu_T_C_Bu,
696                                 JuDeterminant,              k_e_int_Temp,
697                                 g_e_int_Temp,               m_e_int_Temp);
698 
699         if (matrixType == MT_k_e)
```

```
700         {
701             multiplyScalar( iNumberDimensions * iNumberNodes_u,
702                             iNumberDimensions * iNumberNodes_u,
703                             d_W,
704                             k_e_int_Temp,
705                             k_e_int_Temp);
706             sum(    iNumberDimensions * iNumberNodes_u,
707                     iNumberDimensions * iNumberNodes_u,
708                     k_e_int_Temp,
709                     k_e_int,
710                     k_e_int);
711         }
712         else if (matrixType == MT_g_e)
713         {
714             multiplyScalar( iNumberDimensions * iNumberNodes_u,     iNumberNodes_p,
715                             d_W,                                    g_e_int_Temp,
716                             g_e_int_Temp);
717             sum(iNumberDimensions * iNumberNodes_u,     iNumberNodes_p,
718                 g_e_int_Temp,                           g_e_int,
719                 g_e_int);
720         }
721         else if (matrixType == MT_m_e)
722         {
723             multiplyScalar( iNumberDimensions * iNumberNodes_u,
724                             iNumberDimensions * iNumberNodes_u,
725                             d_W,
726                             m_e_int_Temp,
727                             m_e_int_Temp);
728             sum(iNumberDimensions * iNumberNodes_u,
729                 iNumberDimensions * iNumberNodes_u,
730                 m_e_int_Temp,
731                 m_e_int,
732                 m_e_int);
733         }
734     }
735
736     if (meshCFDType == MCFDT_P2P1)
737     {
738         if (matrixType == MT_k_e)
739         {
740             multiplyScalar( iNumberDimensions * iNumberNodes_u,
741                             iNumberDimensions * iNumberNodes_u,
742                             .5,
743                             k_e_int,
744                             k_e_int);
745         }
746         else if (matrixType == MT_g_e)
747         {
748             multiplyScalar( iNumberDimensions * iNumberNodes_u,
749                             iNumberNodes_p,
750                             .5,
751                             g_e_int,
752                             g_e_int);
753         }
754         else if (matrixType == MT_m_e)
755         {
756             multiplyScalar( iNumberDimensions * iNumberNodes_u,
757                             iNumberDimensions * iNumberNodes_u,
758                             .5,
759                             m_e_int,
760                             m_e_int);
761         }
762     }
763 }
764
765 //----------------------------- integrateElement_c_c_t -----------------------
766 __device__
767 void integrateElement_c_c_t(MeshCFDType meshCFDType,
768                             MatrixType  matrixType,
769                             size_t      iNumberDimensions,
770                             size_t      iNumberNodes_u,
771                             Precision*  nodesCoordinates,
772                             Precision*  u_e,
773                             Precision*  v_e,
774                             Precision*  w_e,
775                             size_t      iIntegrationOrder_XSI,
```

```
776                                 size_t      iIntegrationOrder_ETA,
777                                 size_t      iIntegrationOrder_ZETA,
778                                 Precision*  Nu,
779                                 Precision*  NuDerivatives,
780                                 Precision*  Ju,
781                                 Precision*  JuInverse,
782                                 Precision*  NuDerivativesTransformed,
783                                 Precision*  B,
784                                 Precision*  Nu_DOF,
785                                 Precision*  Nu_DOF_T,
786                                 Precision*  e_dot_u,
787                                 Precision*  Nu_DOF_T_e_dot_u,
788                                 Precision*  NuDerivatives_x_DOF,
789                                 Precision*  NuDerivatives_y_DOF,
790                                 Precision*  NuDerivatives_x_DOF_NuDerivatives_y_DOF,
791                                 Precision*  Nu_DOF_T_NuDerivatives_x_DOF_NuDerivatives_y_DOF,
792                                 Precision*  gradu,
793                                 Precision*  Nu_DOF_T_gradu,
794                                 Precision*  Nu_DOF_T_gradu_Nu_DOF,
795                                 Precision   JuDeterminant,
796                                 Precision*  c_e_int,
797                                 Precision*  c_e_int_Temp,
798                                 Precision*  c_t_e_int,
799                                 Precision*  c_t_e_int_Temp)
800 {
801     IntegrationElementType integrationElementType;
802 
803     if (meshCFDType == MCFDT_P2P1)
804     {
805         integrationElementType = TRIANGLE;
806     }
807     else if (meshCFDType == MCFDT_Q2Q1)
808     {
809         integrationElementType = QUADRILATERAL;
810     }
811 
812     Precision integrationRules_XSI[12 * 4];
813     Precision integrationRules_ETA[12 * 4];
814     Precision integrationRules_ZETA[12 * 4];
815 
816     if (iIntegrationOrder_XSI > 0)
817     {
818         getIntegrationRules(integrationElementType,
819                             iIntegrationOrder_XSI,
820                             integrationRules_XSI);
821     }
822 
823     if (iIntegrationOrder_ETA > 0)
824     {
825         getIntegrationRules(integrationElementType,
826                             iIntegrationOrder_ETA,
827                             integrationRules_ETA);
828     }
829 
830     if (iIntegrationOrder_ZETA > 0)
831     {
832         getIntegrationRules(integrationElementType,
833                             iIntegrationOrder_ZETA,
834                             integrationRules_ZETA);
835     }
836 
837     Precision d_XSI = 0.;
838     Precision d_ETA = 0.;
839     Precision d_ZETA = 0.;
840 
841     Precision d_W = 0.;
842     Precision d_W_i = 0.;
843     Precision d_W_j = 0.;
844     Precision d_W_k = 0.;
845 
846     //Calcula a integral 1D
847     for (size_t i = 0; i < iIntegrationOrder_XSI; ++i)
848     {
849         d_W_i = integrationRules_XSI[IDX2C(i, 0, 12)];
850 
851         if (integrationElementType == TRIANGLE)
```

```
852         {
853             d_XSI = integrationRules_XSI[IDX2C(i, 1, 12)];
854             d_ETA = integrationRules_XSI[IDX2C(i, 2, 12)];
855             d_ZETA = integrationRules_XSI[IDX2C(i, 3, 12)];
856         }
857         else if (integrationElementType == QUADRILATERAL)
858         {
859             d_XSI = integrationRules_XSI[IDX2C(i, 1, 12)];
860         }
861 
862         d_W = d_W_i;
863 
864         do_calculate_c_c_t_e(   meshCFDType,            matrixType,
865                                 iNumberDimensions,      iNumberNodes_u,
866                                 nodesCoordinates,       u_e,
867                                 v_e,                    w_e,
868                                 d_XSI,                  d_ETA,
869                                 d_ZETA,                 Nu,
870                                 NuDerivatives,          Ju,
871                                 JuInverse,              NuDerivativesTransformed,
872                                 B,                      Nu_DOF,
873                                 Nu_DOF_T,               e_dot_u,
874                                 Nu_DOF_T_e_dot_u,       NuDerivatives_x_DOF,
875                                 NuDerivatives_y_DOF,
876                                 NuDerivatives_x_DOF_NuDerivatives_y_DOF,
877                                 Nu_DOF_T_NuDerivatives_x_DOF_NuDerivatives_y_DOF,
878                                 gradu,                  Nu_DOF_T_gradu,
879                                 Nu_DOF_T_gradu_Nu_DOF,  JuDeterminant,
880                                 c_e_int_Temp,           c_t_e_int_Temp);
881 
882         if (matrixType == MT_c_e)
883         {
884             multiplyScalar( iNumberDimensions * iNumberNodes_u,
885                             iNumberDimensions * iNumberNodes_u,
886                             d_W,
887                             c_e_int_Temp,
888                             c_e_int_Temp);
889             sum(iNumberDimensions * iNumberNodes_u,
890                 iNumberDimensions * iNumberNodes_u,
891                 c_e_int_Temp,
892                 c_e_int,
893                 c_e_int);
894         }
895         else if (matrixType == MT_c_t_e)
896         {
897             multiplyScalar( iNumberDimensions * iNumberNodes_u,
898                             iNumberDimensions * iNumberNodes_u,
899                             d_W,
900                             c_t_e_int_Temp,
901                             c_t_e_int_Temp);
902             sum(iNumberDimensions * iNumberNodes_u,
903                 iNumberDimensions * iNumberNodes_u,
904                 c_t_e_int_Temp,
905                 c_t_e_int,
906                 c_t_e_int);
907         }
908     }
909 
910     if (meshCFDType == MCFDT_P2P1)
911     {
912         if (matrixType == MT_c_e)
913         {
914             multiplyScalar( iNumberDimensions * iNumberNodes_u,
915                             iNumberDimensions * iNumberNodes_u,
916                             .5,
917                             c_e_int,
918                             c_e_int);
919         }
920         else if (matrixType == MT_c_t_e)
921         {
922             multiplyScalar( iNumberDimensions * iNumberNodes_u,
923                             iNumberDimensions * iNumberNodes_u,
924                             .5,
925                             c_t_e_int,
926                             c_t_e_int);
927         }
```

```
928     }
929 }
930
931 ////////////////////////////////////////////////////////////////////////////////
932
933 //----------------------------- calculate_k_e -------------------------------
934 __device__
935 void do_calculate_k_g_m_e(  MeshCFDType meshCFDType,
936                            MatrixType  matrixType,
937                            size_t      iNumberDimensions,
938                            size_t      iNumberNodes_u,
939                            size_t      iNumberNodes_p,
940                            Precision*  nodesCoordinates,
941                            Precision   RHO,
942                            Precision   MI,
943                            Precision   d_XSI,
944                            Precision   d_ETA,
945                            Precision   d_ZETA,
946                            Precision*  Nu,
947                            Precision*  Np,
948                            Precision*  Nu_DOF,
949                            Precision*  Nu_DOF_T,
950                            Precision*  Nu_DOF_T_Nu_DOF,
951                            Precision*  NuDerivatives,
952                            Precision*  div_Nu,
953                            Precision*  div_Nu_T,
954                            Precision*  div_Nu_T_Np,
955                            Precision*  Ju,
956                            Precision*  JuInverse,
957                            Precision*  NuDerivativesTransformed,
958                            Precision*  Bu,
959                            Precision*  Bu_T,
960                            Precision*  C,
961                            Precision*  Bu_T_C,
962                            Precision*  Bu_T_C_Bu,
963                            Precision   JuDeterminant,
964                            Precision*  k_e_int,
965                            Precision*  g_e_int,
966                            Precision*  m_e_int)
967 {
968     calculate_Nu(meshCFDType, d_XSI, d_ETA, d_ZETA, Nu);
969     calculate_Np(meshCFDType, d_XSI, d_ETA, d_ZETA, Np);
970     calculate_NuDerivatives(meshCFDType, d_XSI, d_ETA, d_ZETA, NuDerivatives);
971     calculate_J(iNumberDimensions, iNumberNodes_u, nodesCoordinates, NuDerivatives, Ju);
972     JuDeterminant = determinant(iNumberDimensions, iNumberDimensions, Ju);
973     inverse(iNumberDimensions, JuDeterminant, Ju, JuInverse);
974     calculate_NDerivativesTransformed(  iNumberDimensions,  iNumberNodes_u,
975                                         Ju,                 JuInverse,
976                                         NuDerivatives,      NuDerivativesTransformed);
977
978     if (matrixType == MT_k_e)
979     {
980         do_calculate_k_e(   iNumberDimensions,          iNumberNodes_u,
981                             RHO,                        MI,
982                             NuDerivativesTransformed,   Bu,
983                             Bu_T,                       C,
984                             Bu_T_C,                     Bu_T_C_Bu,
985                             JuDeterminant,              k_e_int);
986     }
987     else if (matrixType == MT_g_e)
988     {
989         do_calculate_g_e(   iNumberDimensions,          iNumberNodes_u,
990                             iNumberNodes_p,             Np,
991                             NuDerivatives,              Ju,
992                             NuDerivativesTransformed,   div_Nu,
993                             div_Nu_T,                   div_Nu_T_Np,
994                             JuDeterminant,              g_e_int);
995     }
996     else if (matrixType == MT_m_e)
997     {
998         do_calculate_m_e(   iNumberDimensions,  iNumberNodes_u,
999                             RHO,                Nu,
1000                            Nu_DOF,             Nu_DOF_T,
1001                            Nu_DOF_T_Nu_DOF,    JuDeterminant,
1002                            m_e_int);
1003    }
```

```
1004 }
1005
1006 //----------------------------- calculate_m_e -------------------------------
1007 __device__
1008 void do_calculate_c_c_t_e(  MeshCFDType meshCFDType,
1009                             MatrixType  matrixType,
1010                             size_t      iNumberDimensions,
1011                             size_t      iNumberNodes_u,
1012                             Precision*  nodesCoordinates,
1013                             Precision*  u_e,
1014                             Precision*  v_e,
1015                             Precision*  w_e,
1016                             Precision   d_XSI,
1017                             Precision   d_ETA,
1018                             Precision   d_ZETA,
1019                             Precision*  Nu,
1020                             Precision*  NuDerivatives,
1021                             Precision*  Ju,
1022                             Precision*  JuInverse,
1023                             Precision*  NuDerivativesTransformed,
1024                             Precision*  B,
1025                             Precision*  Nu_DOF,
1026                             Precision*  Nu_DOF_T,
1027                             Precision*  e_dot_u,
1028                             Precision*  Nu_DOF_T_e_dot_u,
1029                             Precision*  NuDerivatives_x_DOF,
1030                             Precision*  NuDerivatives_y_DOF,
1031                             Precision*  NuDerivatives_x_DOF_NuDerivatives_y_DOF,
1032                             Precision*  Nu_DOF_T_NuDerivatives_x_DOF_NuDerivatives_y_DOF,
1033                             Precision*  gradu,
1034                             Precision*  Nu_DOF_T_gradu,
1035                             Precision*  Nu_DOF_T_gradu_Nu_DOF,
1036                             Precision   JuDeterminant,
1037                             Precision*  c_e_int,
1038                             Precision*  c_t_e_int)
1039 {
1040     calculate_Nu(meshCFDType, d_XSI, d_ETA, d_ZETA, Nu);
1041     calculate_NuDerivatives(meshCFDType, d_XSI, d_ETA, d_ZETA, NuDerivatives);
1042     calculate_J(iNumberDimensions, iNumberNodes_u, nodesCoordinates, NuDerivatives, Ju);
1043     JuDeterminant = determinant(iNumberDimensions, iNumberDimensions, Ju);
1044     inverse(iNumberDimensions, JuDeterminant, Ju, JuInverse);
1045     calculate_NDerivativesTransformed(  iNumberDimensions,  iNumberNodes_u,
1046                                         Ju,                 JuInverse,
1047                                         NuDerivatives,      NuDerivativesTransformed);
1048
1049     if (matrixType == MT_c_e)
1050     {
1051         do_calculate_c_e(   iNumberDimensions,          iNumberNodes_u,
1052                             u_e,                        v_e,
1053                             w_e,                        Nu,
1054                             NuDerivativesTransformed,   B,
1055                             Nu_DOF,                     Nu_DOF_T,
1056                             e_dot_u,                    Nu_DOF_T_e_dot_u,
1057                             JuDeterminant,              c_e_int);
1058     }
1059     else if (matrixType == MT_c_t_e)
1060     {
1061         do_calculate_c_t_e( iNumberDimensions,          iNumberNodes_u,
1062                             u_e,                        v_e,
1063                             w_e,                        Nu,
1064                             NuDerivativesTransformed,   B,
1065                             Nu_DOF, Nu_DOF_T,
1066                             NuDerivatives_x_DOF,        NuDerivatives_y_DOF,
1067                             NuDerivatives_x_DOF_NuDerivatives_y_DOF,
1068                             Nu_DOF_T_NuDerivatives_x_DOF_NuDerivatives_y_DOF,
1069                             gradu,                      Nu_DOF_T_gradu,
1070                             Nu_DOF_T_gradu_Nu_DOF,      JuDeterminant,
1071                             c_t_e_int);
1072     }
1073 }
1074
1075 ////////////////////////////////////////////////////////////////////////////////
1076
1077 //----------------------------- calculate_k_e -------------------------------
1078 __device__
1079 void do_calculate_k_e(  size_t      iNumberDimensions,
```

```
1080                        size_t      iNumberNodes_u,
1081                        Precision   RHO,
1082                        Precision   MI,
1083                        Precision*  NuDerivativesTransformed,
1084                        Precision*  Bu,
1085                        Precision*  Bu_T,
1086                        Precision*  C,
1087                        Precision*  Bu_T_C,
1088                        Precision*  Bu_T_C_Bu,
1089                        Precision   JuDeterminant,
1090                        Precision*  k_e_int)
1091 {
1092     calculate_B(iNumberDimensions, iNumberNodes_u, NuDerivativesTransformed, Bu);
1093     transpose(3, iNumberDimensions * iNumberNodes_u, Bu, Bu_T);
1094     calculate_C(iNumberDimensions, RHO, MI, C);
1095     multiply(iNumberDimensions * iNumberNodes_u, 3, 3, Bu_T, C, Bu_T_C);
1096     multiply(   iNumberDimensions * iNumberNodes_u, iNumberDimensions * iNumberNodes_u,
1097                 3,                                  Bu_T_C,
1098                 Bu,                                 Bu_T_C_Bu);
1099     multiplyScalar( iNumberDimensions * iNumberNodes_u,
1100                     iNumberDimensions * iNumberNodes_u,
1101                     JuDeterminant,
1102                     Bu_T_C_Bu,
1103                     k_e_int);
1104 }
1105
1106 //------------------------------ calculate_g_e --------------------------------
1107 __device__
1108 void do_calculate_g_e(  size_t      iNumberDimensions,
1109                        size_t      iNumberNodes_u,
1110                        size_t      iNumberNodes_p,
1111                        Precision*  Np,
1112                        Precision*  NuDerivatives,
1113                        Precision*  Ju,
1114                        Precision*  NuDerivativesTransformed,
1115                        Precision*  div_Nu,
1116                        Precision*  div_Nu_T,
1117                        Precision*  div_Nu_T_Np,
1118                        Precision   JuDeterminant,
1119                        Precision*  g_e_int)
1120 {
1121     calculate_divNu(iNumberDimensions, iNumberNodes_u, NuDerivativesTransformed, div_Nu);
1122     transpose(1, iNumberDimensions * iNumberNodes_u, div_Nu, div_Nu_T);
1123     multiply(   iNumberDimensions * iNumberNodes_u,
1124                 iNumberNodes_p,
1125                 1,
1126                 div_Nu_T,
1127                 Np,
1128                 div_Nu_T_Np);
1129     multiplyScalar( iNumberDimensions * iNumberNodes_u,
1130                     iNumberNodes_p,
1131                     JuDeterminant,
1132                     div_Nu_T_Np,
1133                     g_e_int);
1134     multiplyScalar( iNumberDimensions * iNumberNodes_u,
1135                     iNumberNodes_p,
1136                     -1.,
1137                     g_e_int,
1138                     g_e_int);
1139 }
1140
1141 //------------------------------ calculate_m_e --------------------------------
1142 __device__
1143 void do_calculate_m_e(  size_t      iNumberDimensions,
1144                        size_t      iNumberNodes_u,
1145                        Precision   RHO,
1146                        Precision*  Nu,
1147                        Precision*  Nu_DOF,
1148                        Precision*  Nu_DOF_T,
1149                        Precision*  Nu_DOF_T_Nu_DOF,
1150                        Precision   JuDeterminant,
1151                        Precision*  m_e_int)
1152 {
1153     for (size_t i = 0; i < iNumberNodes_u; ++i)
1154     {
1155         Nu_DOF[IDX2C(0, 2 * i, iNumberDimensions)]      = Nu[IDX2C(0, i, 1)];
```

```
1156        Nu_DOF[IDX2C(1, 2 * i + 1, iNumberDimensions)]  = Nu[IDX2C(0, i, 1)];
1157    }
1158
1159    transpose(iNumberDimensions, iNumberDimensions* iNumberNodes_u, Nu_DOF, Nu_DOF_T);
1160    multiply(   iNumberDimensions * iNumberNodes_u,
1161                iNumberDimensions * iNumberNodes_u,
1162                iNumberDimensions,
1163                Nu_DOF_T,
1164                Nu_DOF,
1165                Nu_DOF_T_Nu_DOF);
1166    multiplyScalar( iNumberDimensions * iNumberNodes_u,
1167                    iNumberDimensions * iNumberNodes_u,
1168                    RHO,
1169                    Nu_DOF_T_Nu_DOF,
1170                    m_e_int);
1171    multiplyScalar( iNumberDimensions * iNumberNodes_u,
1172                    iNumberDimensions * iNumberNodes_u,
1173                    JuDeterminant,
1174                    m_e_int,
1175                    m_e_int);
1176 }
1177
1178 ////////////////////////////////////////////////////////////////////////////////
1179
1180 //----------------------------- calculate_g_e -------------------------------
1181 __device__
1182 void do_calculate_c_e(  size_t      iNumberDimensions,
1183                        size_t      iNumberNodes_u,
1184                        Precision*  u_e,
1185                        Precision*  v_e,
1186                        Precision*  w_e,
1187                        Precision*  Nu,
1188                        Precision*  NuDerivativesTransformed,
1189                        Precision*  B,
1190                        Precision*  Nu_DOF,
1191                        Precision*  Nu_DOF_T,
1192                        Precision*  e_dot_u,
1193                        Precision*  Nu_DOF_T_e_dot_u,
1194                        Precision   JuDeterminant,
1195                        Precision*  c_e_int)
1196 {
1197    for (size_t i = 0; i < iNumberNodes_u; ++i)
1198    {
1199        Nu_DOF[IDX2C(0, 2 * i, iNumberDimensions)]      = Nu[IDX2C(0, i, 1)];
1200        Nu_DOF[IDX2C(1, 2 * i + 1, iNumberDimensions)]  = Nu[IDX2C(0, i, 1)];
1201    }
1202
1203    if (iNumberDimensions == 2)
1204    {
1205        for (size_t i = 0; i < iNumberDimensions; ++i)
1206        {
1207            for (size_t j = 0; j < iNumberNodes_u; ++j)
1208            {
1209                B[IDX2C(2 * i, 2 * j + i, 4)] =
1210                    NuDerivativesTransformed[IDX2C(0, j, iNumberDimensions)];
1211                B[IDX2C(2 * i + 1, 2 * j + i, 4)] =
1212                    NuDerivativesTransformed[IDX2C(1, j, iNumberDimensions)];
1213            }
1214        }
1215    }
1216
1217    transpose(iNumberDimensions, iNumberDimensions * iNumberNodes_u, Nu_DOF, Nu_DOF_T);
1218    calculate_e_dot_u(iNumberDimensions, iNumberNodes_u, u_e, v_e, w_e, Nu, e_dot_u);
1219    multiply(   iNumberDimensions * iNumberNodes_u,
1220                iNumberDimensions * iNumberDimensions,
1221                iNumberDimensions,
1222                Nu_DOF_T,
1223                e_dot_u,
1224                Nu_DOF_T_e_dot_u);
1225    multiply(   iNumberDimensions * iNumberNodes_u,
1226                iNumberDimensions * iNumberNodes_u,
1227                iNumberDimensions * iNumberDimensions,
1228                Nu_DOF_T_e_dot_u,
1229                B,
1230                c_e_int);
1231    multiplyScalar( iNumberDimensions * iNumberNodes_u,
```

```
1232                    iNumberDimensions * iNumberNodes_u,
1233                    JuDeterminant,
1234                    c_e_int,
1235                    c_e_int);
1236 }
1237
1238 //----------------------------- calculate_c_t_e -----------------------------
1239 __device__
1240 void do_calculate_c_t_e(size_t      iNumberDimensions,
1241                         size_t      iNumberNodes_u,
1242                         Precision*  u_e,
1243                         Precision*  v_e,
1244                         Precision*  w_e,
1245                         Precision*  Nu,
1246                         Precision*  NuDerivativesTransformed,
1247                         Precision*  B,
1248                         Precision*  Nu_DOF,
1249                         Precision*  Nu_DOF_T,
1250                         Precision*  NuDerivatives_x_DOF,
1251                         Precision*  NuDerivatives_y_DOF,
1252                         Precision*  NuDerivatives_x_DOF_NuDerivatives_y_DOF,
1253                         Precision*  Nu_DOF_T_NuDerivatives_x_DOF_NuDerivatives_y_DOF,
1254                         Precision*  gradu,
1255                         Precision*  Nu_DOF_T_gradu,
1256                         Precision*  Nu_DOF_T_gradu_Nu_DOF,
1257                         Precision   JuDeterminant,
1258                         Precision*  c_t_e_int)
1259 {
1260     if (iNumberDimensions == 2)
1261     {
1262         for (size_t i = 0; i < iNumberNodes_u; ++i)
1263         {
1264             Nu_DOF[IDX2C(0, 2 * i, iNumberDimensions)]      = Nu[IDX2C(0, i, 1)];
1265             Nu_DOF[IDX2C(1, 2 * i + 1, iNumberDimensions)]  = Nu[IDX2C(0, i, 1)];
1266
1267             NuDerivatives_x_DOF[IDX2C(0, 2 * i, iNumberDimensions)]
1268                 = NuDerivativesTransformed[IDX2C(0, i, iNumberDimensions)];
1269             NuDerivatives_x_DOF[IDX2C(1, 2 * i + 1, iNumberDimensions)] =
1270                 NuDerivativesTransformed[IDX2C(0, i, iNumberDimensions)];
1271
1272             NuDerivatives_y_DOF[IDX2C(0, 2 * i, iNumberDimensions)] =
1273                 NuDerivativesTransformed[IDX2C(1, i, iNumberDimensions)];
1274             NuDerivatives_y_DOF[IDX2C(1, 2 * i + 1, iNumberDimensions)] =
1275                 NuDerivativesTransformed[IDX2C(1, i, iNumberDimensions)];
1276         }
1277     }
1278
1279     Precision u[1];
1280     Precision v[1];
1281
1282     multiply(1, 1, iNumberNodes_u, Nu, u_e, u);
1283     multiply(1, 1, iNumberNodes_u, Nu, v_e, v);
1284
1285     transpose(iNumberDimensions, iNumberDimensions * iNumberNodes_u, Nu_DOF, Nu_DOF_T);
1286     multiplyScalar( iNumberDimensions,
1287                     iNumberDimensions * iNumberNodes_u,
1288                     u[0],
1289                     NuDerivatives_x_DOF,
1290                     NuDerivatives_x_DOF);
1291     multiplyScalar( iNumberDimensions,
1292                     iNumberDimensions * iNumberNodes_u,
1293                     v[0],
1294                     NuDerivatives_y_DOF,
1295                     NuDerivatives_y_DOF);
1296     sum(iNumberDimensions,      iNumberDimensions * iNumberNodes_u,
1297         NuDerivatives_x_DOF,    NuDerivatives_y_DOF,
1298         NuDerivatives_x_DOF_NuDerivatives_y_DOF);
1299     multiply(   iNumberDimensions * iNumberNodes_u,
1300                 iNumberDimensions * iNumberNodes_u,
1301                 iNumberDimensions,
1302                 Nu_DOF_T,
1303                 NuDerivatives_x_DOF_NuDerivatives_y_DOF,
1304                 Nu_DOF_T_NuDerivatives_x_DOF_NuDerivatives_y_DOF);
1305
1306     calculate_gradu(iNumberDimensions,  iNumberNodes_u,
1307                     u_e,                v_e,
```

```
                    w_e,                NuDerivativesTransformed,
                    gradu);
    multiply(   iNumberDimensions * iNumberNodes_u,     iNumberDimensions,
                iNumberDimensions,                      Nu_DOF_T,
                gradu,                                  Nu_DOF_T_gradu);
    multiply(   iNumberDimensions * iNumberNodes_u,
                iNumberDimensions * iNumberNodes_u,
                iNumberDimensions,
                Nu_DOF_T_gradu,
                Nu_DOF,
                Nu_DOF_T_gradu_Nu_DOF);

    sum(iNumberDimensions * iNumberNodes_u,
        iNumberDimensions * iNumberNodes_u,
        Nu_DOF_T_NuDerivatives_x_DOF_NuDerivatives_y_DOF,
        Nu_DOF_T_gradu_Nu_DOF,
        c_t_e_int);

    multiplyScalar( iNumberDimensions * iNumberNodes_u,
                    iNumberDimensions * iNumberNodes_u,
                    JuDeterminant,
                    c_t_e_int,
                    c_t_e_int);
}

////////////////////////////////////////////////////////////////////////////////

//------------------------------ calculate_Nu ---------------------------------
__device__
void calculate_Nu(  MeshCFDType meshCFDType,
                    Precision   d_XSI,
                    Precision   d_ETA,
                    Precision   d_ZETA,
                    Precision*  Nu)
{
    if (meshCFDType == MCFDT_P2P1)
    {
        Precision d_ZETA = 1. - d_XSI - d_ETA;

        Nu[IDX2C(0, 0, 1)] = 2. * d_ZETA * (d_ZETA - .5);
        Nu[IDX2C(0, 1, 1)] = 2. * d_XSI * (d_XSI - .5);
        Nu[IDX2C(0, 2, 1)] = 2. * d_ETA * (d_ETA - .5);
        Nu[IDX2C(0, 3, 1)] = 4. * d_ZETA * d_XSI;
        Nu[IDX2C(0, 4, 1)] = 4. * d_XSI * d_ETA;
        Nu[IDX2C(0, 5, 1)] = 4. * d_ETA * d_ZETA;
    }
    else if (meshCFDType == MCFDT_Q2Q1)
    {
        Nu[IDX2C(0, 0, 1)] = .25 * (d_XSI * d_ETA) * (d_XSI - 1.) * (d_ETA - 1.);
        Nu[IDX2C(0, 1, 1)] = .25 * (d_XSI * d_ETA) * (d_XSI + 1.) * (d_ETA - 1.);
        Nu[IDX2C(0, 2, 1)] = .25 * (d_XSI * d_ETA) * (d_XSI + 1.) * (d_ETA + 1.);
        Nu[IDX2C(0, 3, 1)] = .25 * (d_XSI * d_ETA) * (d_XSI - 1.) * (d_ETA + 1.);
        Nu[IDX2C(0, 4, 1)] = -.5 * d_ETA * (d_ETA - 1.) * (powf(d_XSI, 2.) - 1.);
        Nu[IDX2C(0, 5, 1)] = -.5 * d_XSI * (d_XSI + 1.) * (powf(d_ETA, 2.) - 1.);
        Nu[IDX2C(0, 6, 1)] = -.5 * d_ETA * (d_ETA + 1.) * (powf(d_XSI, 2.) - 1.);
        Nu[IDX2C(0, 7, 1)] = -.5 * d_XSI * (d_XSI - 1.) * (powf(d_ETA, 2.) - 1.);
        Nu[IDX2C(0, 8, 1)] = (powf(d_XSI, 2.) - 1.) * (powf(d_ETA, 2.) - 1.);
    }
}

//------------------------------ calculate_Np ---------------------------------
__device__
void calculate_Np(  MeshCFDType meshCFDType,
                    Precision   d_XSI,
                    Precision   d_ETA,
                    Precision   d_ZETA,
                    Precision*  Np)
{
    if (meshCFDType == MCFDT_P2P1)
    {
        Np[IDX2C(0, 0, 1)] = 1. - d_XSI - d_ETA;
        Np[IDX2C(0, 1, 1)] = d_XSI;
        Np[IDX2C(0, 2, 1)] = d_ETA;
    }
    else if (meshCFDType == MCFDT_Q2Q1)
    {
```

```
1384        Np[IDX2C(0, 0, 1)] = .25 * (1 - d_XSI) * (1 - d_ETA);
1385        Np[IDX2C(0, 1, 1)] = .25 * (1 + d_XSI) * (1 - d_ETA);
1386        Np[IDX2C(0, 2, 1)] = .25 * (1 + d_XSI) * (1 + d_ETA);
1387        Np[IDX2C(0, 3, 1)] = .25 * (1 - d_XSI) * (1 + d_ETA);
1388    }
1389 }
1390
1391 //----------------------------- calculate_NDerivatives -----------------------
1392 __device__
1393 void calculate_NuDerivatives(   MeshCFDType meshCFDType,
1394                                 Precision   d_XSI,
1395                                 Precision   d_ETA,
1396                                 Precision   d_ZETA,
1397                                 Precision*  NuDerivatives)
1398 {
1399     if (meshCFDType == MCFDT_P2P1)
1400     {
1401         Precision d_dZETA_dXSI  = -1.;
1402         Precision d_dXSI_dXSI   = 1.;
1403         Precision d_dETA_dXSI   = 0.;
1404
1405         Precision d_dZETA_dETA  = -1.;
1406         Precision d_dXSI_dETA   = 0.;
1407         Precision d_dETA_dETA   = 1.;
1408
1409         Precision d_ZETA = 1. - d_XSI - d_ETA;
1410
1411         NuDerivatives[IDX2C(0, 0, 2)] = (4. * d_ZETA - 1.) * d_dZETA_dXSI;
1412         NuDerivatives[IDX2C(0, 1, 2)] = (4. * d_XSI - 1.) * d_dXSI_dXSI;
1413         NuDerivatives[IDX2C(0, 2, 2)] = (4. * d_ETA - 1.) * d_dETA_dXSI;
1414         NuDerivatives[IDX2C(0, 3, 2)] = 4. * d_XSI * d_dZETA_dXSI +
1415             4. * d_ZETA * d_dXSI_dXSI;
1416         NuDerivatives[IDX2C(0, 4, 2)] = 4. * d_ETA * d_dXSI_dXSI +
1417             4. * d_XSI * d_dETA_dXSI;
1418         NuDerivatives[IDX2C(0, 5, 2)] = 4. * d_ETA * d_dZETA_dXSI +
1419             4 * d_ZETA * d_dETA_dXSI;
1420
1421         NuDerivatives[IDX2C(1, 0, 2)] = (4. * d_ZETA - 1.) * d_dZETA_dETA;
1422         NuDerivatives[IDX2C(1, 1, 2)] = (4. * d_XSI - 1.) * d_dXSI_dETA;
1423         NuDerivatives[IDX2C(1, 2, 2)] = (4. * d_ETA - 1.) * d_dETA_dETA;
1424         NuDerivatives[IDX2C(1, 3, 2)] = 4. * d_XSI * d_dZETA_dETA +
1425             4. * d_ZETA * d_dXSI_dETA;
1426         NuDerivatives[IDX2C(1, 4, 2)] = 4. * d_ETA * d_dXSI_dETA +
1427             4. * d_XSI * d_dETA_dETA;
1428         NuDerivatives[IDX2C(1, 5, 2)] = 4. * d_ETA * d_dZETA_dETA +
1429             4 * d_ZETA * d_dETA_dETA;
1430     }
1431     else if (meshCFDType == MCFDT_Q2Q1)
1432     {
1433         NuDerivatives[IDX2C(0, 0, 2)] = .25 * (2. * d_XSI * powf(d_ETA, 2.) -
1434             2. * d_XSI * d_ETA - powf(d_ETA, 2.) + d_ETA);
1435         NuDerivatives[IDX2C(0, 1, 2)] = .25 * (2. * d_XSI * powf(d_ETA, 2.) -
1436             2. * d_XSI * d_ETA + powf(d_ETA, 2.) - d_ETA);
1437         NuDerivatives[IDX2C(0, 2, 2)] = .25 * (2. * d_XSI * powf(d_ETA, 2.) +
1438             2. * d_XSI * d_ETA + powf(d_ETA, 2.) + d_ETA);
1439         NuDerivatives[IDX2C(0, 3, 2)] = .25 * (2. * d_XSI * powf(d_ETA, 2.) +
1440             2. * d_XSI * d_ETA - powf(d_ETA, 2.) - d_ETA);
1441         NuDerivatives[IDX2C(0, 4, 2)] = -.5 * (2 * d_XSI * powf(d_ETA, 2.) -
1442             2 * d_XSI * d_ETA);
1443         NuDerivatives[IDX2C(0, 5, 2)] = -.5 * (2 * d_XSI * powf(d_ETA, 2.) -
1444             2 * d_XSI + powf(d_ETA, 2.) - 1.);
1445         NuDerivatives[IDX2C(0, 6, 2)] = -.5 * (2 * d_XSI * powf(d_ETA, 2.) +
1446             2 * d_XSI * d_ETA);
1447         NuDerivatives[IDX2C(0, 7, 2)] = -.5 * (2 * d_XSI * powf(d_ETA, 2.) -
1448             2 * d_XSI - powf(d_ETA, 2.) + 1.);
1449         NuDerivatives[IDX2C(0, 8, 2)] = 2 * d_XSI * powf(d_ETA, 2.) - 2 * d_XSI;
1450
1451         NuDerivatives[IDX2C(1, 0, 2)] = .25 * (2. * powf(d_XSI, 2.) *
1452             d_ETA - powf(d_XSI, 2.) - 2. * d_XSI * d_ETA + d_XSI);
1453         NuDerivatives[IDX2C(1, 1, 2)] = .25 * (2. * powf(d_XSI, 2.) *
1454             d_ETA - powf(d_XSI, 2.) + 2. * d_XSI * d_ETA - d_XSI);
1455         NuDerivatives[IDX2C(1, 2, 2)] = .25 * (2. * powf(d_XSI, 2.) *
1456             d_ETA + powf(d_XSI, 2.) + 2. * d_XSI * d_ETA + d_XSI);
1457         NuDerivatives[IDX2C(1, 3, 2)] = .25 * (2. * powf(d_XSI, 2.) *
1458             d_ETA + powf(d_XSI, 2.) - 2. * d_XSI * d_ETA - d_XSI);
1459         NuDerivatives[IDX2C(1, 4, 2)] = -.5 * (2 * powf(d_XSI, 2.) *
```

```
1460            d_ETA - 2 * d_ETA - powf(d_XSI, 2.) + 1.);
1461        NuDerivatives[IDX2C(1, 5, 2)] = -.5 * (2 * powf(d_XSI, 2.) *
1462            d_ETA + 2 * d_XSI * d_ETA);
1463        NuDerivatives[IDX2C(1, 6, 2)] = -.5 * (2 * powf(d_XSI, 2.) *
1464            d_ETA - 2 * d_ETA + powf(d_XSI, 2.) - 1.);
1465        NuDerivatives[IDX2C(1, 7, 2)] = -.5 * (2 * powf(d_XSI, 2.) *
1466            d_ETA - 2 * d_XSI * d_ETA);
1467        NuDerivatives[IDX2C(1, 8, 2)] = 2 * powf(d_XSI, 2.) * d_ETA - 2 * d_ETA;
1468    }
1469 }
1470
1471 ////////////////////////////////////////////////////////////////////////////////
1472
1473 //----------------------------- calculate_divNu -----------------------------
1474 __device__
1475 void calculate_divNu(   size_t      iNumberDimensions,
1476                        size_t      iNumberNodes_u,
1477                        Precision*  NuDerivatives,
1478                        Precision*  divNu)
1479 {
1480     for (size_t j = 0; j < iNumberNodes_u; ++j)
1481     {
1482         for (size_t i = 0; i < iNumberDimensions; ++i)
1483         {
1484             divNu[IDX2C(0, iNumberDimensions * j + i, 1)] =
1485                 NuDerivatives[IDX2C(i, j, iNumberDimensions)];
1486         }
1487     }
1488 }
1489
1490 ////////////////////////////////////////////////////////////////////////////////
1491
1492 //----------------------------- calculate_C ---------------------------------
1493 __device__
1494 void calculate_C(   size_t      iNumberDimensions,
1495                    Precision   RHO,
1496                    Precision   MI,
1497                    Precision*  C)
1498 {
1499     Precision d_NI = MI / RHO;
1500
1501     if (iNumberDimensions == 2)
1502     {
1503         C[IDX2C(0, 0, 3)] = 2. * d_NI;
1504         C[IDX2C(0, 1, 3)] = 0. * d_NI;
1505         C[IDX2C(0, 2, 3)] = 0. * d_NI;
1506
1507         C[IDX2C(1, 0, 3)] = 0. * d_NI;
1508         C[IDX2C(1, 1, 3)] = 2. * d_NI;
1509         C[IDX2C(1, 2, 3)] = 0. * d_NI;
1510
1511         C[IDX2C(2, 0, 3)] = 0. * d_NI;
1512         C[IDX2C(2, 1, 3)] = 0. * d_NI;
1513         C[IDX2C(2, 2, 3)] = 1. * d_NI;
1514     }
1515 }
1516
1517 ////////////////////////////////////////////////////////////////////////////////
1518
1519 //----------------------------- calculate_e_dot_u ---------------------------
1520 __device__
1521 void calculate_e_dot_u( size_t      iNumberDimensions,
1522                        size_t      iNumberNodes_u,
1523                        Precision*  u_e,
1524                        Precision*  v_e,
1525                        Precision*  w_e,
1526                        Precision*  Nu,
1527                        Precision*  e_dot_u)
1528 {
1529     if (iNumberDimensions == 2)
1530     {
1531         Precision u[1];
1532         Precision v[1];
1533
1534         multiply(1, 1, iNumberNodes_u, Nu, u_e, u);
1535         multiply(1, 1, iNumberNodes_u, Nu, v_e, v);
```

```
        e_dot_u[IDX2C(0, 0, iNumberDimensions)] = u[0];
        e_dot_u[IDX2C(0, 1, iNumberDimensions)] = v[0];

        e_dot_u[IDX2C(1, 2, iNumberDimensions)] = u[0];
        e_dot_u[IDX2C(1, 3, iNumberDimensions)] = v[0];
    }
    else if (iNumberDimensions == 3)
    {
        Precision u[1];
        Precision v[1];
        Precision w[1];

        multiply(1, 1, iNumberNodes_u, Nu, u_e, u);
        multiply(1, 1, iNumberNodes_u, Nu, v_e, v);
        multiply(1, 1, iNumberNodes_u, Nu, w_e, w);

        e_dot_u[IDX2C(0, 0, iNumberDimensions)] = u[0];
        e_dot_u[IDX2C(0, 1, iNumberDimensions)] = v[0];
        e_dot_u[IDX2C(0, 2, iNumberDimensions)] = w[0];

        e_dot_u[IDX2C(1, 3, iNumberDimensions)] = u[0];
        e_dot_u[IDX2C(1, 4, iNumberDimensions)] = v[0];
        e_dot_u[IDX2C(1, 5, iNumberDimensions)] = w[0];
    }
}

//------------------------------ calculate_gradu ------------------------------
__device__
void calculate_gradu(   size_t      iNumberDimensions,
                        size_t      iNumberNodes_u,
                        Precision*  u_e,
                        Precision*  v_e,
                        Precision*  w_e,
                        Precision*  NuDerivativesTransformed,
                        Precision*  gradu)
{
    if (iNumberDimensions == 2)
    {
        Precision NuDerivatives_u_e[2 * 1];
        Precision NuDerivatives_v_e[2 * 1];

        multiply(   iNumberDimensions,  1,
                    iNumberNodes_u,     NuDerivativesTransformed,
                    u_e,                NuDerivatives_u_e);
        multiply(   iNumberDimensions,  1,
                    iNumberNodes_u,     NuDerivativesTransformed,
                    v_e,                NuDerivatives_v_e);

        gradu[IDX2C(0, 0, iNumberDimensions)] =
            NuDerivatives_u_e[IDX2C(0, 0, iNumberDimensions)];
        gradu[IDX2C(0, 1, iNumberDimensions)] =
            NuDerivatives_u_e[IDX2C(1, 0, iNumberDimensions)];

        gradu[IDX2C(1, 0, iNumberDimensions)] =
            NuDerivatives_v_e[IDX2C(0, 0, iNumberDimensions)];
        gradu[IDX2C(1, 1, iNumberDimensions)] =
            NuDerivatives_v_e[IDX2C(1, 0, iNumberDimensions)];
    }
}

#endif
```

- *CUDA_Mesh.cuh*

```
/*****************************************************************************/
//                                HPC - LMC                                  //
//---------------------------------------------------------------------------//
/*!
 * \brief   Classe CUDA_Mesh
 *
 * Esta classe é responsável pelas operações com os elementos
 *
```

```
 * \author  Luiz Felipe Couto
 */
////////////////////////////////////////////////////////////////////////////////

#ifndef HPC_LMC_FEM_CUDA_FEM_CUDA_MESH_H
#define HPC_LMC_FEM_CUDA_FEM_CUDA_MESH_H

//------------------------------------------------------------------------------
// Include and Forward declaration
//------------------------------------------------------------------------------
#include "../../core/cuda/CUDA_Macros.cuh"
#include "../../core/cuda/CUDA_MatrixOperation.cuh"

//------------------------------------------------------------------------------
// Enumeration
//------------------------------------------------------------------------------
enum MeshCFDType
{
    MCFDT_P2P1,
    MCFDT_Q2Q1
};

//! Enumeration que define o tipo de matriz.
enum MatrixType
{
    //Local
    MT_k_e,     //!< Matriz k local - Rigidez, viscosa

    ////////////////////////////// CFD ////////////////////////////////////
    MT_m_e,     //!< Matriz m local -- Massa
    MT_g_e,     //!< Matriz g local -- Operador Gradiente
    MT_c_e,     //!< Matriz c local -- Convectiva
    MT_c_t_e,   //!< Matriz c_t local -- Tangente
    ////////////////////////////////////////////////////////////////////////

    MT_f_t_e,   //!< Matriz f_t local -- Forças de superfície
    MT_f_b_e,   //!< Matriz f_b local -- Forças de volume
    MT_F_e,     //!< Matriz F_e local -- Forças nodais
    MT_f_e,     //!< Matriz F_e local -- Forças totais

    //Global
    MT_K,       //!< Matriz K -- Rigidez, viscosa

    ////////////////////////////// CFD ////////////////////////////////////
    MT_M,       //!< Matrix M global -- Massa
    MT_G,       //!< Matriz G global -- Operador Gradiente
    MT_C,       //!< Matriz C global -- Convectiva
    MT_C_T,     //!< Matriz C_T global -- Tangente
    ////////////////////////////////////////////////////////////////////////

    MT_f_t,     //!< Matriz f_t -- Forças de superfície
    MT_f_b,     //!< Matriz f_b -- Forças de volume
    MT_F,       //!< Matriz F -- Forças nodais
    MT_f        //!< Matriz f -- Forças
};

//------------------------------------------------------------------------------
// Prototype definition
//------------------------------------------------------------------------------
__device__
void calculate_J(   size_t      iNumberDimensions,
                    size_t      iNumberNodes,
                    Precision*  nodesCoordinates,
                    Precision*  nodalData,
                    Precision*  NDerivatives,
                    Precision*  J);

__device__
void calculate_NDerivativesTransformed( size_t      iNumberDimensions,
                                        size_t      iNumberNodes,
                                        Precision*  J,
                                        Precision*  JInverse,
                                        Precision*  NDerivatives,
                                        Precision*  NDerivativesTransformed);

__device__
```

```
void calculate_B(   size_t      iNumberDimensions,
                    size_t      iNumberNodes,
                    Precision*  NDerivativesTransformed,
                    Precision*  B);

//-----------------------------------------------------------------------------
// Device Function
//-----------------------------------------------------------------------------

//------------------------------ calculate_J ---------------------------------
__device__
void  calculate_J(  size_t      iNumberDimensions,
                    size_t      iNumberNodes,
                    Precision*  nodesCoordinates,
                    Precision*  NDerivatives,
                    Precision*  J)
{
    multiply(   iNumberDimensions,  iNumberDimensions,
                iNumberNodes,       NDerivatives,
                nodesCoordinates,   J);
}

////////////////////////////////////////////////////////////////////////////////

//------------------------------ calculate_NDerivativesTransformed ------------
__device__
void calculate_NDerivativesTransformed( size_t      iNumberDimensions,
                                        size_t      iNumberNodes,
                                        Precision*  J,
                                        Precision*  JInverse,
                                        Precision*  NDerivatives,
                                        Precision*  NDerivativesTransformed)
{
    multiply(   iNumberDimensions,  iNumberNodes,
                iNumberDimensions,  JInverse,
                NDerivatives,       NDerivativesTransformed);
}

////////////////////////////////////////////////////////////////////////////////

//------------------------------ calculate_B ---------------------------------
__device__
void calculate_B(   size_t      iNumberDimensions,
                    size_t      iNumberNodes,
                    Precision*  NDerivativesTransformed,
                    Precision* B)
{
    //Para análise 2D -- B[3, #DOF], 3D -- B[6, #DOF]
    if (iNumberDimensions == 2)
    {
        for (size_t i = 0; i < iNumberNodes; ++i)
        {
            B[IDX2C(0, 2 * i, 3)] =
                NDerivativesTransformed[IDX2C(0, i, iNumberDimensions)];
            B[IDX2C(1, 2 * i + 1, 3)] =
                NDerivativesTransformed[IDX2C(1, i, iNumberDimensions)];
            B[IDX2C(2, 2 * i, 3)] =
                NDerivativesTransformed[IDX2C(1, i, iNumberDimensions)];
            B[IDX2C(2, 2 * i + 1, 3)] =
                NDerivativesTransformed[IDX2C(0, i, iNumberDimensions)];
        }
    }
}

#endif
```

- *CUDA_Main.cu*

```
/*****************************************************************************/
//                                HPC - LMC                                  //
//---------------------------------------------------------------------------//
/*!
 * \brief   CUDA_Main
```

```
6   *
7   * Arquivo responsável pela definição das funções principais de assembly no
8   * device
9   *
10  * \author  Luiz Felipe Couto
11  */
12 ////////////////////////////////////////////////////////////////////////////////
13
14 //-----------------------------------------------------------------------------
15 // Include and Forward declaration
16 //-----------------------------------------------------------------------------
17 #include "CUDA_MeshCFD.cuh"
18
19 //-----------------------------------------------------------------------------
20 // Define declaration
21 //-----------------------------------------------------------------------------
22 //!< Tamanho do bloco de threads a serem executados em paralelo
23 //!< Está relacionado com o tamanho da variável shared utilizada
24 #define BLOCK_SIZE 128
25
26 //-----------------------------------------------------------------------------
27 // Struct definition
28 //-----------------------------------------------------------------------------
29 //! Estrutura de dados do vetor de dados nodais
30 struct NodalData
31 {
32     Precision       x;              //!< Coordenada x do nó do elemento finito
33     Precision       y;              //!< Coordenada y do nó do elemento finito
34
35     //!< GL Global da velocidade na direção x do nó do elemento finito
36     unsigned int    DOFGlobal_u;
37     //!< GL Local da velocidade na direção y do nó do elemento finito
38     unsigned int    DOFLocal_u;
39     //!< Velocidade na direção x do nó do elemento finito (C e C_t)
40     Precision       u;
41
42     //!< GL Global da velocidade na direção y do nó do elemento finito
43     unsigned int    DOFGlobal_v;
44     //!< GL Local da velocidade na direção y do nó do elemento finito
45     unsigned int    DOFLocal_v;
46     //!< Velocidade na direção y do nó do elemento finito (C e C_t)
47     Precision       v;
48
49     //!< GL Global da pressão do nó do elemento finito
50     unsigned int    DOFGlobal_p;
51     //!< GL Local da pressão do nó do elemento finito
52     unsigned int    DOFLocal_p;
53 };
54
55 //! Estrutura i, j e valor a ser armazenado na matriz esparsa global
56 struct Triplets
57 {
58     unsigned int    i;                      //!< Linha i da matriz esparsa global
59     unsigned int    j;                      //!< Linha j da matriz esparsa global
60
61     //!< Valor a ser armazenado na posição i, j na matriz esparsa global
62     Precision       value;
63 };
64
65 //-----------------------------------------------------------------------------
66 // Global variables in device
67 //-----------------------------------------------------------------------------
68 NodalData*      dev_nodalData;              //!< Vetor de dados nodais
69
70 //!< Vetor de velocidades nos nós do elemento finito
71 Precision*      dev_u;
72
73 unsigned int*   dev_integrationOrder_XSI;   //!< Pontos de integração na direção XSI
74 unsigned int*   dev_integrationOrder_ETA;   //!< Pontos de integração na direção ETA
75 unsigned int*   dev_integrationOrder_ZETA;  //!< Pontos de integração na direção ZETA
76
77 Triplets*       dev_K;                      //!< Matriz de rigidez
78 Triplets*       dev_G;                      //!< Matriz operador gradiente
79 Triplets*       dev_M;                      //!< Matriz de massa
80
81 Triplets*       dev_C;                      //!< Matriz convectiva
```

```
82  Triplets*       dev_C_t;                    //!< Matriz convectiva tangente
83
84  //-----------------------------------------------------------------------------
85  // Mapped global variables in device
86  //-----------------------------------------------------------------------------
87  __device__ NodalData*       d_nodalData;
88
89  __device__ Precision*       d_u;
90
91  __device__ unsigned int*    d_integrationOrder_XSI;
92  __device__ unsigned int*    d_integrationOrder_ETA;
93  __device__ unsigned int*    d_integrationOrder_ZETA;
94
95  __device__ Triplets*        d_K;
96  __device__ Triplets*        d_G;
97  __device__ Triplets*        d_M;
98
99  __device__ Triplets*        d_C;
100 __device__ Triplets*        d_C_t;
101
102 //-----------------------------------------------------------------------------
103 // Kernel implementation
104 //-----------------------------------------------------------------------------
105
106 //------------------------------ do_assembly_K_G_M ----------------------------
107 //! Realiza o assembly das matrizes K, G e M
108 /*!
109  * \param meshCFDType Tipo de elemento finito: P2P1 ou Q2Q1
110  * \param inumberDimensions Número de dimensões do problema analisado
111  * \param iNumberNodes_u Número de nós para velocidade
112  * \param iNumberNodes_p Número de nós para pressão
113  * \param N Número de elementos finitos do problema
114  * \param RHO Massa específica do fluido
115  * \param MI Coeficiente de viscosidade dinâmica do fluido
116  */
117 __global__
118 void do_assembly_K_G_M( MeshCFDType meshCFDType,
119                         size_t      iNumberDimensions,
120                         size_t      iNumberNodes_u,
121                         size_t      iNumberNodes_p,
122                         size_t      N,
123                         Precision   RHO,
124                         Precision   MI)
125 {
126     ////////////////////////////////////////////////////////////////////////
127     //     Variável do tipo shared a ser compartilhada com as threads do     //
128     //     bloco                                                             //
129     ////////////////////////////////////////////////////////////////////////
130     extern __shared__ NodalData s_nodalData[];
131
132     ////////////////////////////////////////////////////////////////////////
133     //     Armazena os valores dos dados nodais global na variável           //
134     //     compartilhada                                                     //
135     ////////////////////////////////////////////////////////////////////////
136     size_t iContNodalData = 0;
137
138     for (   size_t  i       = blockDim.x * blockIdx.x * iNumberNodes_u,
139     iLen    = blockDim.x * blockIdx.x * iNumberNodes_u + blockDim.x * iNumberNodes_u;
140     i < iLen;
141     ++i)
142     {
143         if (i < N * iNumberNodes_u)
144         {
145             s_nodalData[iContNodalData].x           = d_nodalData[i].x;
146             s_nodalData[iContNodalData].y           = d_nodalData[i].y;
147
148             s_nodalData[iContNodalData].DOFGlobal_u = d_nodalData[i].DOFGlobal_u;
149             s_nodalData[iContNodalData].DOFLocal_u  = d_nodalData[i].DOFLocal_u;
150             s_nodalData[iContNodalData].u           = d_nodalData[i].u;
151
152             s_nodalData[iContNodalData].DOFGlobal_v = d_nodalData[i].DOFGlobal_v;
153             s_nodalData[iContNodalData].DOFLocal_v  = d_nodalData[i].DOFLocal_v;
154             s_nodalData[iContNodalData].v           = d_nodalData[i].v;
155
156             s_nodalData[iContNodalData].DOFGlobal_p = d_nodalData[i].DOFGlobal_p;
157             s_nodalData[iContNodalData].DOFLocal_p  = d_nodalData[i].DOFLocal_p;
```

```
158         }
159
160         ++iContNodalData;
161     }
162
163     ////////////////////////////////////////////////////////////////////////
164     //     Sincroniza as threads do bloco para início do assembly         //
165     ////////////////////////////////////////////////////////////////////////
166     __syncthreads();
167
168     ////////////////////////////////////////////////////////////////////////
169     //     Início do processo de cálculo e assembly                       //
170     ////////////////////////////////////////////////////////////////////////
171     size_t tIdLocal     = threadIdx.x;
172     size_t tIdGlobal    = blockDim.x * blockIdx.x + threadIdx.x;
173
174     if (tIdGlobal < N)
175     {
176         Precision nodesCoordinates[6 * 2]   = { 0. };
177         unsigned int DOFMap[6 * 6]          = { 0 };
178
179         Precision k_e[(2 * 6 + 3) * (2 * 6 + 3)] = { 0. };
180         Precision g_e[(2 * 6 + 3) * (2 * 6 + 3)] = { 0. };
181         Precision m_e[(2 * 6 + 3) * (2 * 6 + 3)] = { 0. };
182
183         ////////////////////////////////////////////////////////////////////
184         //     Armazena os graus de liberdade global e local do elemento     //
185         ////////////////////////////////////////////////////////////////////
186         for (size_t i = 0; i < iNumberNodes_u; ++i)
187         {
188             nodesCoordinates[IDX2C(i, 0, 6)] =
189                 s_nodalData[tIdLocal * iNumberNodes_u + i].x;
190             nodesCoordinates[IDX2C(i, 1, 6)] =
191                 s_nodalData[tIdLocal * iNumberNodes_u + i].y;
192
193             if (iNumberDimensions == 2)
194             {
195                 DOFMap[IDX2C(i, 0, iNumberNodes_u)] =
196                     s_nodalData[tIdLocal * iNumberNodes_u + i].DOFGlobal_u;
197                 DOFMap[IDX2C(i, 1, iNumberNodes_u)] =
198                     s_nodalData[tIdLocal * iNumberNodes_u + i].DOFLocal_u;
199
200                 DOFMap[IDX2C(i, 2, iNumberNodes_u)] =
201                     s_nodalData[tIdLocal * iNumberNodes_u + i].DOFGlobal_v;
202                 DOFMap[IDX2C(i, 3, iNumberNodes_u)] =
203                     s_nodalData[tIdLocal * iNumberNodes_u + i].DOFLocal_v;
204
205                 DOFMap[IDX2C(i, 4, iNumberNodes_u)] =
206                     s_nodalData[tIdLocal * iNumberNodes_u + i].DOFGlobal_p;
207                 DOFMap[IDX2C(i, 5, iNumberNodes_u)] =
208                     s_nodalData[tIdLocal * iNumberNodes_u + i].DOFLocal_p;
209             }
210             else if (iNumberDimensions == 3)
211             {
212             }
213         }
214
215         ////////////////////////////////////////////////////////////////////
216         //     Calcula a contribuição de cada elemento das matrizes M, K e G //
217         ////////////////////////////////////////////////////////////////////
218         calculate_k_g_m_e(  meshCFDType,            iNumberDimensions,
219                             nodesCoordinates,       RHO,
220                             MI,                     d_integrationOrder_XSI,
221                             d_integrationOrder_ETA, d_integrationOrder_ZETA,
222                             k_e,                    g_e,
223                             m_e);
224
225         ////////////////////////////////////////////////////////////////////
226         //     Realiza o assembly das matrizes locais nas matrizes globais   //
227         ////////////////////////////////////////////////////////////////////
228         size_t iDOFLocalNumber_i = 0;
229         size_t iDOFLocalNumber_j = 0;
230
231         size_t iDOFGlobalNumber_i = 0;
232         size_t iDOFGlobalNumber_j = 0;
233
```

```
234         size_t iLine_K_G_M = 0;
235
236         for (size_t i = 0; i < iNumberNodes_u; ++i)
237         {
238             for (size_t j = 0; j < iNumberDimensions + 1; ++j)
239             {
240                 iDOFGlobalNumber_i  = DOFMap[IDX2C(i, (2 * j), iNumberNodes_u)];
241                 iDOFLocalNumber_i   = DOFMap[IDX2C(i, (2 * j + 1), iNumberNodes_u)];
242
243                 for (size_t k = 0; k < iNumberNodes_u; ++k)
244                 {
245                     for (size_t l = 0; l < iNumberDimensions + 1; ++l)
246                     {
247                         iDOFGlobalNumber_j  = DOFMap[IDX2C(k, 2 * l, iNumberNodes_u)];
248                         iDOFLocalNumber_j   = DOFMap[IDX2C(k, 2 * l + 1, iNumberNodes_u)];
249
250                         if (iDOFLocalNumber_i > 0 && iDOFLocalNumber_j > 0)
251                         {
252                             d_K[tIdGlobal * (iNumberDimensions * iNumberNodes_u +
253                                 iNumberNodes_p) * (iNumberDimensions * iNumberNodes_u +
254                                 iNumberNodes_p) + iLine_K_G_M].i = iDOFGlobalNumber_i - 1;
255                             d_K[tIdGlobal * (iNumberDimensions * iNumberNodes_u +
256                                 iNumberNodes_p) * (iNumberDimensions * iNumberNodes_u +
257                                 iNumberNodes_p) + iLine_K_G_M].j = iDOFGlobalNumber_j - 1;
258                             d_K[tIdGlobal * (iNumberDimensions * iNumberNodes_u +
259                                 iNumberNodes_p) * (iNumberDimensions * iNumberNodes_u +
260                                 iNumberNodes_p) + iLine_K_G_M].value =
261                                     k_e[IDX2C(iDOFLocalNumber_i - 1,
262                                         iDOFLocalNumber_j - 1,
263                                         15)];
264
265                             d_G[tIdGlobal * (iNumberDimensions * iNumberNodes_u +
266                                 iNumberNodes_p) * (iNumberDimensions * iNumberNodes_u +
267                                 iNumberNodes_p) + iLine_K_G_M].i = iDOFGlobalNumber_i - 1;
268                             d_G[tIdGlobal * (iNumberDimensions * iNumberNodes_u +
269                                 iNumberNodes_p) * (iNumberDimensions * iNumberNodes_u +
270                                 iNumberNodes_p) + iLine_K_G_M].j = iDOFGlobalNumber_j - 1;
271                             d_G[tIdGlobal * (iNumberDimensions * iNumberNodes_u +
272                                 iNumberNodes_p) * (iNumberDimensions * iNumberNodes_u +
273                                 iNumberNodes_p) + iLine_K_G_M].value =
274                                     g_e[IDX2C(iDOFLocalNumber_i - 1,
275                                         iDOFLocalNumber_j - 1,
276                                         15)];
277
278                             d_M[tIdGlobal * (iNumberDimensions * iNumberNodes_u +
279                                 iNumberNodes_p) * (iNumberDimensions * iNumberNodes_u +
280                                 iNumberNodes_p) + iLine_K_G_M].i = iDOFGlobalNumber_i - 1;
281                             d_M[tIdGlobal * (iNumberDimensions * iNumberNodes_u +
282                                 iNumberNodes_p) * (iNumberDimensions * iNumberNodes_u +
283                                 iNumberNodes_p) + iLine_K_G_M].j = iDOFGlobalNumber_j - 1;
284                             d_M[tIdGlobal * (iNumberDimensions * iNumberNodes_u +
285                                 iNumberNodes_p) * (iNumberDimensions * iNumberNodes_u +
286                                 iNumberNodes_p) + iLine_K_G_M].value =
287                                     m_e[IDX2C(iDOFLocalNumber_i - 1,
288                                         iDOFLocalNumber_j - 1,
289                                         15)];
290
291                             ++iLine_K_G_M;
292                         }
293                     }
294                 }
295             }
296         }
297     }
298 }
299
300 //------------------------------ do_assembly_C_C_t ----------------------------
301 //! Realiza o assembly das matrizes C e C_t
302 /*!
303  * \param meshCFDType Tipo de elemento finito: P2P1 ou Q2Q1
304  * \param inumberDimensions Número de dimensões do problema analisado
305  * \param iNumberNodes_u Número de nós para velocidade
306  * \param iNumberNodes_p Número de nós para pressão
307  * \param N Número de elementos finitos do problema
308  */
309 __global__
```

```
310 void do_assembly_C_C_t( MeshCFDType meshCFDType,
311                         size_t      iNumberDimensions,
312                         size_t      iNumberNodes_u,
313                         size_t      iNumberNodes_p,
314                         size_t      N)
315 {
316     ////////////////////////////////////////////////////////////////////////
317     //      Variável do tipo shared a ser compartilhada com as threads do     //
318     //      bloco                                                             //
319     ////////////////////////////////////////////////////////////////////////
320     extern __shared__ NodalData s_nodalData[];
321
322     ////////////////////////////////////////////////////////////////////////
323     //      Armazena os valores dos dados nodais global na variável           //
324     //      compartilhada                                                     //
325     ////////////////////////////////////////////////////////////////////////
326     size_t iContNodalData = 0;
327
328     for (   size_t  i = blockDim.x * blockIdx.x * iNumberNodes_u,
329     iLen = blockDim.x * blockIdx.x * iNumberNodes_u + blockDim.x * iNumberNodes_u;
330     i < iLen;
331     ++i)
332     {
333         if (i < N * iNumberNodes_u)
334         {
335             s_nodalData[iContNodalData].x = d_nodalData[i].x;
336             s_nodalData[iContNodalData].y = d_nodalData[i].y;
337
338             s_nodalData[iContNodalData].DOFGlobal_u = d_nodalData[i].DOFGlobal_u;
339             s_nodalData[iContNodalData].DOFLocal_u  = d_nodalData[i].DOFLocal_u;
340             s_nodalData[iContNodalData].u           = d_nodalData[i].u;
341
342             s_nodalData[iContNodalData].DOFGlobal_v = d_nodalData[i].DOFGlobal_v;
343             s_nodalData[iContNodalData].DOFLocal_v  = d_nodalData[i].DOFLocal_v;
344             s_nodalData[iContNodalData].v           = d_nodalData[i].v;
345
346             s_nodalData[iContNodalData].DOFGlobal_p = d_nodalData[i].DOFGlobal_p;
347             s_nodalData[iContNodalData].DOFLocal_p  = d_nodalData[i].DOFLocal_p;
348         }
349
350         ++iContNodalData;
351     }
352
353     ////////////////////////////////////////////////////////////////////////
354     //      Sincroniza as threads do bloco para início do assembly            //
355     ////////////////////////////////////////////////////////////////////////
356     __syncthreads();
357
358     ////////////////////////////////////////////////////////////////////////
359     //      Início do processo de cálculo e assembly                          //
360     ////////////////////////////////////////////////////////////////////////
361     size_t tIdLocal     = threadIdx.x;
362     size_t tIdGlobal    = blockDim.x * blockIdx.x + threadIdx.x;
363
364     if (tIdGlobal < N)
365     {
366         Precision nodesCoordinates[6 * 2] = { 0. };
367         Precision u_e[6 * 1] = { 0. };
368         Precision v_e[6 * 1] = { 0. };
369         Precision w_e[6 * 1] = { 0. };
370         unsigned int DOFMap[6 * 6] = { 0 };
371
372         Precision c_e[(2 * 6 + 3) * (2 * 6 + 3)] = { 0. };
373         Precision c_t_e[(2 * 6 + 3) * (2 * 6 + 3)] = { 0. };
374
375         ////////////////////////////////////////////////////////////////////
376         //      Armazena os graus de liberdade global, local e velocidade     //
377         //      dos nós do elemento                                           //
378         ////////////////////////////////////////////////////////////////////
379         for (size_t i = 0; i < 6; ++i)
380         {
381             nodesCoordinates[IDX2C(i, 0, 6)] =
382                 s_nodalData[tIdLocal * iNumberNodes_u + i].x;
383             nodesCoordinates[IDX2C(i, 1, 6)] =
384                 s_nodalData[tIdLocal * iNumberNodes_u + i].y;
385
```

```
386             if (iNumberDimensions == 2)
387             {
388                 DOFMap[IDX2C(i, 0, iNumberNodes_u)] =
389                     s_nodalData[tIdLocal * iNumberNodes_u + i].DOFGlobal_u;
390                 DOFMap[IDX2C(i, 1, iNumberNodes_u)] =
391                     s_nodalData[tIdLocal * iNumberNodes_u + i].DOFLocal_u;
392                 u_e[IDX2C(i, 0, iNumberNodes_u)] =
393                 s_nodalData[tIdLocal * iNumberNodes_u + i].u;
394
395                 DOFMap[IDX2C(i, 2, iNumberNodes_u)] =
396                     s_nodalData[tIdLocal * iNumberNodes_u + i].DOFGlobal_v;
397                 DOFMap[IDX2C(i, 3, iNumberNodes_u)] =
398                     s_nodalData[tIdLocal * iNumberNodes_u + i].DOFLocal_v;
399                 v_e[IDX2C(i, 0, iNumberNodes_u)] =
400                 s_nodalData[tIdLocal * iNumberNodes_u + i].v;
401
402                 DOFMap[IDX2C(i, 4, iNumberNodes_u)] =
403                     s_nodalData[tIdLocal * iNumberNodes_u + i].DOFGlobal_p;
404                 DOFMap[IDX2C(i, 5, iNumberNodes_u)] =
405                     s_nodalData[tIdLocal * iNumberNodes_u + i].DOFLocal_p;
406             }
407             else if (iNumberDimensions == 3)
408             {
409             }
410         }
411
412         ////////////////////////////////////////////////////////////////////
413         //     Calcula a contribuição de cada elemento das matrizes C e C_t  //
414         ////////////////////////////////////////////////////////////////////
415         calculate_c_c_t_e(  meshCFDType,                iNumberDimensions,
416                             nodesCoordinates,           u_e,
417                             v_e,                        w_e,
418                             d_integrationOrder_XSI,     d_integrationOrder_ETA,
419                             d_integrationOrder_ZETA,    c_e,
420                             c_t_e);
421
422         ////////////////////////////////////////////////////////////////////
423         //     Realiza o assembly das matrizes locais nas matrizes globais   //
424         ////////////////////////////////////////////////////////////////////
425         size_t iDOFLocalNumber_i = 0;
426         size_t iDOFLocalNumber_j = 0;
427
428         size_t iDOFGlobalNumber_i = 0;
429         size_t iDOFGlobalNumber_j = 0;
430
431         size_t iLine_C_C_t = 0;
432
433         for (size_t i = 0; i < iNumberNodes_u; ++i)
434         {
435             for (size_t j = 0; j < iNumberDimensions + 1; ++j)
436             {
437                 iDOFGlobalNumber_i  = DOFMap[IDX2C(i, (2 * j), iNumberNodes_u)];
438                 iDOFLocalNumber_i   = DOFMap[IDX2C(i, (2 * j + 1), iNumberNodes_u)];
439
440                 for (size_t k = 0; k < iNumberNodes_u; ++k)
441                 {
442                     for (size_t l = 0; l < iNumberDimensions + 1; ++l)
443                     {
444                         iDOFGlobalNumber_j  = DOFMap[IDX2C(k, 2 * l, iNumberNodes_u)];
445                         iDOFLocalNumber_j   = DOFMap[IDX2C(k, 2 * l + 1, iNumberNodes_u)];
446
447                         if (iDOFLocalNumber_i > 0 && iDOFLocalNumber_j > 0)
448                         {
449                             d_C[tIdGlobal * (iNumberDimensions * iNumberNodes_u +
450                                 iNumberNodes_p) * (iNumberDimensions * iNumberNodes_u +
451                                 iNumberNodes_p) + iLine_C_C_t].i = iDOFGlobalNumber_i - 1;
452                             d_C[tIdGlobal * (iNumberDimensions * iNumberNodes_u +
453                                 iNumberNodes_p) * (iNumberDimensions * iNumberNodes_u +
454                                 iNumberNodes_p) + iLine_C_C_t].j = iDOFGlobalNumber_j - 1;
455                             d_C[tIdGlobal * (iNumberDimensions * iNumberNodes_u +
456                                 iNumberNodes_p) * (iNumberDimensions * iNumberNodes_u +
457                                 iNumberNodes_p) + iLine_C_C_t].value =
458                                     c_e[IDX2C(iDOFLocalNumber_i - 1,
459                                         iDOFLocalNumber_j - 1,
460                                         15)];
461
```

```
                            d_C_t[tIdGlobal * (iNumberDimensions * iNumberNodes_u +
                                iNumberNodes_p) * (iNumberDimensions * iNumberNodes_u +
                                iNumberNodes_p) + iLine_C_C_t].i = iDOFGlobalNumber_i - 1;
                            d_C_t[tIdGlobal * (iNumberDimensions * iNumberNodes_u +
                                iNumberNodes_p) * (iNumberDimensions * iNumberNodes_u +
                                iNumberNodes_p) + iLine_C_C_t].j = iDOFGlobalNumber_j - 1;
                            d_C_t[tIdGlobal * (iNumberDimensions * iNumberNodes_u +
                                iNumberNodes_p) * (iNumberDimensions * iNumberNodes_u +
                                iNumberNodes_p) + iLine_C_C_t].value =
                                    c_t_e[  IDX2C(iDOFLocalNumber_i - 1,
                                            iDOFLocalNumber_j - 1,
                                            15)];

                            ++iLine_C_C_t;
                        }
                    }
                }
            }
        }
    }
}

//-----------------------------------------------------------------------------
// External function
//-----------------------------------------------------------------------------

//------------------------------ initializeVariables --------------------------
//! Inicializa as veriáveis no device
/*!
 * \param inumberDimensions Número de dimensões do problema analisado
 * \param iNumberNodes_u Número de nós para velocidade
 * \param iNumberNodes_p Número de nós para pressão
 * \param N Número de elementos finitos do problema
 * \param nodalData Vetor de dados nodais
 * \param integrationOrder_XSI Pontos de integração na direção XSI
 * \param integrationOrder_ETA Pontos de integração na direção ETA
 * \param integrationOrder_ZETA Pontos de integração na direção ZETA
 */
extern "C"
void initializeVariables(   size_t          iNumberDimensions,
                            size_t          iNumberNodes_u,
                            size_t          iNumberNodes_p,
                            size_t          N,
                            NodalData*      nodalData,
                            unsigned int*   integrationOrder_XSI,
                            unsigned int*   integrationOrder_ETA,
                            unsigned int*   integrationOrder_ZETA)
{
    ////////////////////////////////////////////////////////////////////////
    //      Aloca memória no device                                       //
    ////////////////////////////////////////////////////////////////////////
    HANDLE_ERROR(cudaMalloc((void**)&dev_nodalData,
                            1 * N * iNumberNodes_u * sizeof(NodalData)));

    HANDLE_ERROR(cudaMalloc((void**)&dev_integrationOrder_XSI,
                            5 * sizeof(unsigned int)));
    HANDLE_ERROR(cudaMalloc((void**)&dev_integrationOrder_ETA,
                            5 * sizeof(unsigned int)));
    HANDLE_ERROR(cudaMalloc((void**)&dev_integrationOrder_ZETA,
                            5 * sizeof(unsigned int)));

    ////////////////////////////////////////////////////////////////////////
    //      Copia os dados do host para o device                          //
    ////////////////////////////////////////////////////////////////////////
    HANDLE_ERROR(cudaMemcpy(    dev_nodalData,
                                nodalData,
                                1 * N * iNumberNodes_u * sizeof(NodalData),
                                cudaMemcpyHostToDevice));

    HANDLE_ERROR(cudaMemcpy(    dev_integrationOrder_XSI,
                                integrationOrder_XSI, 5 * sizeof(unsigned int),
                                cudaMemcpyHostToDevice));
    HANDLE_ERROR(cudaMemcpy(    dev_integrationOrder_ETA,
                                integrationOrder_ETA, 5 * sizeof(unsigned int),
                                cudaMemcpyHostToDevice));
    HANDLE_ERROR(cudaMemcpy(    dev_integrationOrder_ZETA,
```

```
                                integrationOrder_ZETA,
                                5 * sizeof(unsigned int),
                                cudaMemcpyHostToDevice));

    ////////////////////////////////////////////////////////////////////////
    //     Mapeais as variáveis no device para as variáveias mapeadas globais//
    ////////////////////////////////////////////////////////////////////////
    HANDLE_ERROR(cudaMemcpyToSymbol(    d_nodalData,
                                        &dev_nodalData,
                                        sizeof(dev_nodalData),
                                        0,
                                        cudaMemcpyHostToDevice));

    HANDLE_ERROR(cudaMemcpyToSymbol(    d_integrationOrder_XSI,
                                        &dev_integrationOrder_XSI,
                                        sizeof(dev_integrationOrder_XSI),
                                        0,
                                        cudaMemcpyHostToDevice));
    HANDLE_ERROR(cudaMemcpyToSymbol(    d_integrationOrder_ETA,
                                        &dev_integrationOrder_ETA,
                                        sizeof(dev_integrationOrder_ETA),
                                        0,
                                        cudaMemcpyHostToDevice));
    HANDLE_ERROR(cudaMemcpyToSymbol(    d_integrationOrder_ZETA,
                                        &dev_integrationOrder_ZETA,
                                        sizeof(dev_integrationOrder_ZETA),
                                        0,
                                        cudaMemcpyHostToDevice));
}

//------------------------------ initializeVariables_K_G_M --------------------
//! Inicializa as veriáveis no device
/*!
 * \param inumberDimensions Número de dimensões do problema analisado
 * \param iNumberNodes_u Número de nós para velocidade
 * \param iNumberNodes_p Número de nós para pressão
 * \param N Número de elementos finitos do problema
 */
extern "C"
void initializeVariables_K_G_M( size_t      iNumberDimensions,
                                size_t      iNumberNodes_u,
                                size_t      iNumberNodes_p,
                                size_t      N)
{
    ////////////////////////////////////////////////////////////////////////
    //     Aloca memória no device                                        //
    ////////////////////////////////////////////////////////////////////////
    HANDLE_ERROR(cudaMalloc((void**)&dev_K,
                            (iNumberDimensions * iNumberNodes_u + iNumberNodes_p) *
                                (iNumberDimensions * iNumberNodes_u + iNumberNodes_p) *
                                N * sizeof(Triplets)));
    HANDLE_ERROR(cudaMalloc((void**)&dev_G,
                            (iNumberDimensions * iNumberNodes_u + iNumberNodes_p) *
                                (iNumberDimensions * iNumberNodes_u + iNumberNodes_p) *
                                N * sizeof(Triplets)));
    HANDLE_ERROR(cudaMalloc((void**)&dev_M,
                            (iNumberDimensions * iNumberNodes_u + iNumberNodes_p) *
                                (iNumberDimensions * iNumberNodes_u + iNumberNodes_p) *
                                N * sizeof(Triplets)));

    ////////////////////////////////////////////////////////////////////////
    //     Mapeais as variáveis no device para as variáveias mapeadas globais//
    ////////////////////////////////////////////////////////////////////////
    HANDLE_ERROR(cudaMemcpyToSymbol(d_K,            &dev_K,
                                    sizeof(dev_K),  0,
                                    cudaMemcpyHostToDevice));
    HANDLE_ERROR(cudaMemcpyToSymbol(d_G,            &dev_G,
                                    sizeof(dev_G),  0,
                                    cudaMemcpyHostToDevice));
    HANDLE_ERROR(cudaMemcpyToSymbol(d_M,            &dev_M,
                                    sizeof(dev_M),  0,
                                    cudaMemcpyHostToDevice));
}

//------------------------------ assembly_K_G_M -------------------------------
//! Assembly das matrizes M, K e G
```

```
614 /*!
615  * \param meshCFDType Tipo de elemento finito a ser utilizado no problema analisado
616  * \param inumberDimensions Número de dimensões do problema analisado
617  * \param iNumberNodes_u Número de nós para velocidade
618  * \param iNumberNodes_p Número de nós para pressão
619  * \param N Número de elementos finitos do problema
620  * \param RHO Massa espcífica do fluido
621  * \param MI Coeficiente de viscosidade dinâmica do fluido
622  * \param M Matriz de massa
623  * \param K Matirz de rigidez
624  * \param G Matriz operador gradiente
625  */
626 extern "C"
627 void assembly_K_G_M(    MeshCFDType meshCFDType,
628                         size_t      iNumberDimensions,
629                         size_t      iNumberNodes_u,
630                         size_t      iNumberNodes_p,
631                         size_t      N,
632                         Precision   RHO,
633                         Precision   MI,
634                         Triplets*   K,
635                         Triplets*   G,
636                         Triplets*   M)
637 {
638     ////////////////////////////////////////////////////////////////////////
639     //     Executa o assembly no device                                   //
640     ////////////////////////////////////////////////////////////////////////
641     do_assembly_K_G_M<<<(N + BLOCK_SIZE - 1) / BLOCK_SIZE, BLOCK_SIZE, BLOCK_SIZE *
642         iNumberNodes_u * sizeof(NodalData)>>>(meshCFDType, iNumberDimensions,
643         iNumberNodes_u,
644         iNumberNodes_p,
645         N,
646         RHO,
647         MI);
648
649     ////////////////////////////////////////////////////////////////////////
650     //     Copia os dados do device para o host                           //
651     ////////////////////////////////////////////////////////////////////////
652     HANDLE_ERROR(cudaMemcpy(K,
653                             dev_K,
654                             (iNumberDimensions * iNumberNodes_u + iNumberNodes_p) *
655                                 (iNumberDimensions * iNumberNodes_u + iNumberNodes_p) *
656                                 N * sizeof(Triplets),
657                             cudaMemcpyDeviceToHost));
658     HANDLE_ERROR(cudaMemcpy(G,
659                             dev_G,
660                             (iNumberDimensions * iNumberNodes_u + iNumberNodes_p) *
661                                 (iNumberDimensions * iNumberNodes_u + iNumberNodes_p) *
662                                 N * sizeof(Triplets), cudaMemcpyDeviceToHost));
663     HANDLE_ERROR(cudaMemcpy(M,
664                             dev_M,
665                             (iNumberDimensions * iNumberNodes_u + iNumberNodes_p) *
666                                 (iNumberDimensions * iNumberNodes_u + iNumberNodes_p) *
667                                 N * sizeof(Triplets), cudaMemcpyDeviceToHost));
668 }
669
670 ////////////////////////////////////////////////////////////////////////////////
671
672 //------------------------------ initializeVariables_C_C_t --------------------
673 //! Inicializa as veriáveis no device das matrizes C e C_t
674 /*!
675  * \param meshCFDType Tipo de elemento finito a ser utilizado no problema analisado
676  * \param inumberDimensions Número de dimensões do problema analisado
677  * \param iNumberNodes_u Número de nós para velocidade
678  * \param iNumberNodes_p Número de nós para pressão
679  * \param N Número de elementos finitos do problema
680  */
681 extern "C"
682 void initializeVariables_C_C_t( size_t      iNumberDimensions,
683                                 size_t      iNumberNodes_u,
684                                 size_t      iNumberNodes_p,
685                                 size_t      N)
686 {
687     ////////////////////////////////////////////////////////////////////////
688     //     Aloca memória no device                                        //
689     ////////////////////////////////////////////////////////////////////////
```

```
690     HANDLE_ERROR(cudaMalloc((void**)&dev_C,
691                             (iNumberDimensions * iNumberNodes_u + iNumberNodes_p) *
692                                 (iNumberDimensions * iNumberNodes_u + iNumberNodes_p) *
693                                 N * sizeof(Triplets)));
694     HANDLE_ERROR(cudaMalloc((void**)&dev_C_t,
695                             (iNumberDimensions * iNumberNodes_u + iNumberNodes_p) *
696                                 (iNumberDimensions * iNumberNodes_u + iNumberNodes_p) *
697                                 N * sizeof(Triplets)));
698
699     ////////////////////////////////////////////////////////////////////////
700     //     Mapeais as variáveis no device para as variáveias mapeadas globais//
701     ////////////////////////////////////////////////////////////////////////
702     HANDLE_ERROR(cudaMemcpyToSymbol(    d_C,            &dev_C,
703                                         sizeof(dev_C),  0,
704                                         cudaMemcpyHostToDevice));
705     HANDLE_ERROR(cudaMemcpyToSymbol(    d_C_t,              &dev_C_t,
706                                         sizeof(dev_C_t),    0,
707                                         cudaMemcpyHostToDevice));
708 }
709
710 ////////////////////////////////////////////////////////////////////////////
711
712 //------------------------------ assembly_C_C_t ------------------------------
713 //! Assembly das matrizes C e C_t
714 /*!
715  * \param meshCFDType Tipo de elemento finito a ser utilizado no problema analisado
716  * \param inumberDimensions Número de dimensões do problema analisado
717  * \param iNumberNodes_u Número de nós para velocidade
718  * \param iNumberNodes_p Número de nós para pressão
719  * \param N Número de elementos finitos do problema
720  * \param nodalData Vetor de dados nodais
721  * \param C Matriz convectiva
722  * \param C_t Matriz convectiva tangente
723  */
724 extern "C"
725 void assembly_C_C_t(MeshCFDType meshCFDType,
726                     size_t      iNumberDimensions,
727                     size_t      iNumberNodes_u,
728                     size_t      iNumberNodes_p,
729                     size_t      N,
730                     NodalData*  nodalData,
731                     Triplets*   C,
732                     Triplets*   C_t)
733 {
734     ////////////////////////////////////////////////////////////////////////
735     //     Copia os dados do host para o device                           //
736     ////////////////////////////////////////////////////////////////////////
737     HANDLE_ERROR(cudaMemcpy(dev_nodalData,
738                             nodalData,
739                             1 * N * iNumberNodes_u * sizeof(NodalData),
740                             cudaMemcpyHostToDevice));
741
742     ////////////////////////////////////////////////////////////////////////
743     //     Executa o assembly no device                                   //
744     ////////////////////////////////////////////////////////////////////////
745     do_assembly_C_C_t<<<(N + BLOCK_SIZE - 1) / BLOCK_SIZE, BLOCK_SIZE, BLOCK_SIZE *
746         iNumberNodes_u * sizeof(NodalData)>>>(  meshCFDType,    iNumberDimensions,
747                                                 iNumberNodes_u, iNumberNodes_p, N);
748
749     ////////////////////////////////////////////////////////////////////////
750     //     Copia os dados do device para o host                           //
751     ////////////////////////////////////////////////////////////////////////
752     HANDLE_ERROR(cudaMemcpy(C,
753                             dev_C,
754                             (iNumberDimensions * iNumberNodes_u + iNumberNodes_p) *
755                                 (iNumberDimensions * iNumberNodes_u + iNumberNodes_p) *
756                                 N * sizeof(Triplets), cudaMemcpyDeviceToHost));
757     HANDLE_ERROR(cudaMemcpy(C_t,
758                             dev_C_t,
759                             (iNumberDimensions * iNumberNodes_u + iNumberNodes_p) *
760                                 (iNumberDimensions * iNumberNodes_u + iNumberNodes_p) *
761                                 N * sizeof(Triplets), cudaMemcpyDeviceToHost));
762 }
763
764 ////////////////////////////////////////////////////////////////////////////
765
```

```
766 //------------------------------ freeVariables --------------------------------
767 //! Libera as variáveis da memória do device
768 extern "C"
769 void freeVariables()
770 {
771     HANDLE_ERROR(cudaFree(dev_nodalData));
772 
773     HANDLE_ERROR(cudaFree(dev_integrationOrder_XSI));
774     HANDLE_ERROR(cudaFree(dev_integrationOrder_ETA));
775     HANDLE_ERROR(cudaFree(dev_integrationOrder_ZETA));
776 }
777 
778 //------------------------------ freeVariables_K_G_M --------------------------
779 //! Libera as matrizes M, K e G da memória do device
780 extern "C"
781 void freeVariables_K_G_M()
782 {
783     HANDLE_ERROR(cudaFree(dev_K));
784     HANDLE_ERROR(cudaFree(dev_G));
785     HANDLE_ERROR(cudaFree(dev_M));
786 }
787 
788 //------------------------------ freeVariablesC_C_t ---------------------------
789 //! Libera as matrizes C e C_t da memória do device
790 extern "C"
791 void freeVariables_C_C_t()
792 {
793     HANDLE_ERROR(cudaFree(dev_C));
794     HANDLE_ERROR(cudaFree(dev_C_t));
795 }
```

APÊNDICE B – ARQUIVO NEUTRO

O arquivo neutro é um arquivo ASCII[1] que possui as informações necessárias para importar ou exportar dados de elementos finitos, como informações de pré-processamento: malha de elementos finitos (coordenadas dos nós, conexão entre os nós, tipo de elemento finito, etc.), condições de contorno, como também informações de pós-processamento: resultados de velocidade e pressão em cada nó, entre outros.

Cada seção do arquivo neutro é separado por um cabeçalho seguido por linhas que representam os dados ou informações daquele cabeçalho. A próxima seção apresenta os cabeçalhos e as informações que podem conter em um arquivo neutro e um exemplo de utilização e resultados.

O arquivo neutro de pré-processamento é gerado pelo GiD, e o de resultados tanto pode ser gerado para leitura pelo GiD como pelo ParaView.

B.1 Formato do arquivo

Esta seção contém as informações dos cabeçalhos utilizados em um arquivo neutro.

B.1.1 Informações de controle

%SPACE.DIMENSIONS

Define o número de dimensões do problema analisado.

Formato: Variável numérica

Valor	Descrição
2	Problema bidimensional
3	Problema tridimensional

%FSI (Opcional)

Define se o problema é de interação fluido-estrutura.

Formato: Variável numérica

Valor	Descrição
1	O problema é de interação fluido-estrutura

[1] O arquivo de texto ou arquivo ASCII (*American Standard for Computer Information Interchange*) contém apenas caracteres que representam letras, números e outros símbolos, segundo um padrão universalmente aceito.

B.1.2 Integração numérica

%INTEGRATION.TYPE

Define o tipo de integração que será executada no polígono.

Formato: Variável numérica

Valor	Descrição
1	Integração com *Tessellation*
2	Integração no polígono

%QUADRATURE.GAUSS.EDGE (Opcional)

Define o número de pontos de Gauss em um lado do elemento finito por dimensão do problema.

Formato: Tabela

Coluna	Valor ou Tipo	Descrição
1	Valor inteiro	Código da quadratura
2	Valor inteiro	Número de pontos de Gauss em ξ
3	Valor inteiro	Número de pontos de Gauss em η
4	Valor inteiro	Número de pontos de Gauss em ζ

%QUADRATURE.GAUSS.FACE (Opcional)

Define o número de pontos de Gauss em uma face do elemento finito por dimensão do problema.

Formato: Tabela

Coluna	Valor ou Tipo	Descrição
1	Valor inteiro	Código da quadratura
2	Valor inteiro	Número de pontos de Gauss em ξ
3	Valor inteiro	Número de pontos de Gauss em η
4	Valor inteiro	Número de pontos de Gauss em ζ

%QUADRATURE.GAUSS.TRIANGLE (Opcional)

Define o número de pontos de Gauss em um elemento finito do tipo triangular por dimensão do problema.

Formato: Tabela

Coluna	Valor ou Tipo	Descrição
1	Valor inteiro	Código da quadratura
2	Valor inteiro	Número de pontos de Gauss em ξ
3	Valor inteiro	Número de pontos de Gauss em η
4	Valor inteiro	Número de pontos de Gauss em ζ

%QUADRATURE.GAUSS.QUADRILATERAL (Opcional)

Define o número de pontos de Gauss em um elemento finito do tipo quadrangular por dimensão do problema.

Formato: Tabela

Coluna	Valor ou Tipo	Descrição
1	Valor inteiro	Código da quadratura
2	Valor inteiro	Número de pontos de Gauss em ξ
3	Valor inteiro	Número de pontos de Gauss em η
4	Valor inteiro	Número de pontos de Gauss em ζ

%QUADRATURE.GAUSS.TETRAHEDRON (Opcional)

Define o número de pontos de Gauss em um elemento finito do tipo tetraédrico por dimensão do problema.

Formato: Tabela

Coluna	Valor ou Tipo	Descrição
1	Valor inteiro	Código da quadratura
2	Valor inteiro	Número de pontos de Gauss em ξ
3	Valor inteiro	Número de pontos de Gauss em η
4	Valor inteiro	Número de pontos de Gauss em ζ

%QUADRATURE.GAUSS.HEXAHEDRON (Opcional)

Define o número de pontos de Gauss em um elemento finito do tipo hexaédrico por dimensão do problema.

Formato: Tabela

Coluna	Valor ou Tipo	Descrição
1	Valor inteiro	Código da quadratura
2	Valor inteiro	Número de pontos de Gauss em ξ
3	Valor inteiro	Número de pontos de Gauss em η
4	Valor inteiro	Número de pontos de Gauss em ζ

B.1.3 Saída de dados

%IO.GID (Opcional)

Define a saída dos resultados no formato aceito pelo GID.

%IO.VTK (Opcional)

Define a saída dos resultados no formato aceito pelo ParaView.

B.1.4 Campos escalares ou vetoriais

%FIELD

Define os campos escalares ou vetoriais que serão analisados no problema.

Formato: Tabela pré-definida de acordo com o número de dimensões do problema

Campo vetorial de velocidades (dinâmica dos fluidos e interação fluido-estrutura) ou deslocamentos (estruturas) para problemas bidimensionais:

Problemas bidimensionais – Campo vetorial de velocidade		
Coluna	Valor	Descrição
1	u	Campo vetorial de velocidade
2	2	Número de dimensões do problema
3	u	Nome da variável associado a direção 1
4	v	Nome da variável associado a direção 2

Campo vetorial de velocidades (dinâmica dos fluidos e interação fluido-estrutura) ou deslocamentos (estruturas) para problemas tridimensionais:

Problemas tridimensionais – Campo vetorial de velocidade		
Coluna	Valor	Descrição
1	u	Campo vetorial de velocidade
2	3	Número de dimensões do problema
3	u	Nome da variável associado a direção 1
4	v	Nome da variável associado a direção 2
5	w	Nome da variável associado a direção 3

Campo escalar de pressão (dinâmica dos fluidos e interação fluido-estrutura):

Campo escalar de pressão		
Coluna	Valor	Descrição
1	p	Campo escalar de pressão
2	1	Um único valor escalar
3	p	Nome da variável associado a este campo escalar

B.1.5 Material

%MATERIAL.NEO-HOOKEAN

Define as propriedades do material da estrutura.

Formato: Tabela

Coluna	Valor ou Tipo	Descrição
1	Valor inteiro	Código do material
2	Valor em ponto flutuante	Módulo de Young
3	Valor em ponto flutuante	Poisson

%MATERIAL.FLUID

Define as propriedades do material do fluido.

Formato: Tabela

Coluna	Valor ou Tipo	Descrição
1	Valor inteiro	Código do material
2	Valor em ponto flutuante	Densidade
3	Valor em ponto flutuante	Viscosidade cinemática

B.1.6 Tipo de tensão no contorno

%ANALYSIS.TYPE

Define o tipo de tensão do contorno para problemas de dinâmica dos fluidos e interação fluido-estrutura.

Formato: Valor pré-definido

Valor	Descrição
CFD_REAL_TRACTION	Tensão no contorno
CFD_PSEUDO_TRACTION	Pseudotensão no contorno

B.1.7 Análise não-linear

%EQUATION.TYPE.NONLINEAR

Define o erro e a tolerância na análise não-linear.

Formato: Tabela

Coluna	Valor ou Tipo	Descrição
1	Valor em ponto flutuante	Erro
2	Valor em ponto flutuante	Tolerância

B.1.8 Informações da análise

%FLOW.TYPE

Define o tipo de análise para dinâmica dos fluidos e interação fluido-estrutura.

Formato: Variável numérica

Valor	Descrição
1	Análise estacionária
2	Análise transiente

%PROBLEM.TYPE

Define o tipo de análise para estruturas.

Formato: Variável numérica

Valor	Descrição
1	Análise estática
2	Análise dinâmica

%TIME.INTEGRATION

Define as informações da integração no tempo.

Formato: Tabela

Coluna	Valor ou Tipo	Descrição
1	Valor em ponto flutuante	Tempo de início
2	Valor em ponto flutuante	Tempo de fim
3	Valor em ponto flutuante	Intervalo de tempo

%TIME.INTEGRATION.NEWMARK (Opcional)

Define as informações da integração no tempo utilizando Newmark.

Formato: Tabela

Coluna	Valor ou Tipo	Descrição
1	Valor em ponto flutuante	γ

%TIME.RECORDER

Define o intervalo de tempo para gravação dos resultados.

Formato: Tabela

Coluna	Valor ou Tipo	Descrição
1	Valor em ponto flutuante	Intervalo de tempo para gravação dos resultados

B.1.9 Informações da malha de elementos finitos

B.1.9.1 Nós

%NODE.COORD

Define as coordenadas dos nós da malha de elementos finitos.

Formato: Tabela

Coluna	Valor ou Tipo	Descrição
1	Valor inteiro	Número do nó
2	Valor em ponto flutuante	Coordenada do nó na direção 1
3	Valor em ponto flutuante	Coordenada do nó na direção 2
4	Valor em ponto flutuante	Coordenada do nó na direção 3

%NODE.DOF.TYPES

Define as restrições aplicadas aos nós da malha de elementos finitos.

Formato: Tabela

Coluna	Valor ou Tipo	Descrição
1	Valor inteiro	Número do nó
2	[0..1]	Restrição da velocidade na direção 1 (0: livre e 1: bloqueado)
3	[0..1]	Restrição da velocidade na direção 2 (0: livre e 1: bloqueado)
4	[0..1]	Restrição da velocidade na direção 3 (0: livre e 1: bloqueado)
5	[0..1]	Restrição da pressão (0: livre e 1: bloqueado)

%NODE.PRESCRIBED.VALUE

Define os valores prescritos aplicados aos nós da malha de elementos finitos.

Formato: Tabela

Coluna	Valor ou Tipo	Descrição
1	Valor inteiro	Número do nó
2	Valor em ponto flutuante	Valor da velocidade prescrita na direção 1
3	Valor em ponto flutuante	Valor da velocidade prescrita na direção 2
4	Valor em ponto flutuante	Valor da velocidade prescrita na direção 3
5	Valor em ponto flutuante	Valor da pressão prescrita

B.1.9.2 Forças

%LOAD.CASE.NODAL.FORCE

Define as forças aplicadas aos nós da malha de elementos finitos.

Formato: Tabela

Coluna	Valor ou Tipo	Descrição
1	Valor inteiro	Número do nó
2	Valor em ponto flutuante	Força aplicada na direção 1
3	Valor em ponto flutuante	Força aplicada na direção 2
4	Valor em ponto flutuante	Força aplicada na direção 3

%LOAD.CASE.LINE.FORCE.UNIFORM

Define as forças distribuídas uniformemente aos lados do elemento finito.

Formato: Tabela

Coluna	Valor ou Tipo	Descrição
1	Valor inteiro	Número do nó
2	Valor em ponto flutuante	Força aplicada na direção 1
3	Valor em ponto flutuante	Força aplicada na direção 1
4	Valor em ponto flutuante	Força aplicada na direção 1

%LOAD.CASE.LINE.FORCE.VARIABLE

Define as forças distribuídas de forma variável aos lados do elemento finito.

Formato: Tabela

Coluna	Valor ou Tipo	Descrição
1	Valor inteiro	Número do nó
2	Valor em ponto flutuante	Força aplicada na direção 1 no nó inicial
3	Valor em ponto flutuante	Força aplicada na direção 2 no nó inicial
4	Valor em ponto flutuante	Força aplicada na direção 3 no nó inicial
5	Valor em ponto flutuante	Força aplicada na direção 1 no nó final
6	Valor em ponto flutuante	Força aplicada na direção 2 no nó final
7	Valor em ponto flutuante	Força aplicada na direção 3 no nó final

B.1.9.3 Elemento

%ELEMENT.P2P1 (Opcional)

Define os elementos finitos híbridos do tipo triangular de 6 nós.

Formato: Tabela

Coluna	Valor ou Tipo	Descrição
1	Valor inteiro	Número do elemento finito
2	Valor inteiro	Código do material deste elemento finito
3	1	–
4	Valor inteiro	Código da integração numérica
[5..10]	Valor inteiro	Número do nó

%ELEMENT.T3 (Opcional)

Define os elementos finitos do tipo triangular de 3 nós.

Formato: Tabela

Coluna	Valor ou Tipo	Descrição
1	Valor inteiro	Número do elemento finito
2	Valor inteiro	Código do material deste elemento finito
3	1	–
4	Valor inteiro	Código da integração numérica
[5..7]	Valor inteiro	Número do nó

%ELEMENT.T6 (Opcional)

Define os elementos finitos do tipo triangular de 6 nós.

Formato: Tabela

Coluna	Valor ou Tipo	Descrição
1	Valor inteiro	Número do elemento finito
2	Valor inteiro	Código do material deste elemento finito
3	1	–
4	Valor inteiro	Código da integração numérica
[5..10]	Valor inteiro	Número do nó

%ELEMENT.Q2Q1 (Opcional)

Define os elementos finitos híbridos do tipo quadrangular de 9 nós.

Formato: Tabela

Coluna	Valor ou Tipo	Descrição
1	Valor inteiro	Número do elemento finito
2	Valor inteiro	Código do material deste elemento finito
3	1	–
4	Valor inteiro	Código da integração numérica
[5..13]	Valor inteiro	Número do nó

%ELEMENT.Q4 (Opcional)

Define os elementos finitos do tipo quadrangular de 4 nós.

Formato: Tabela

Coluna	Valor ou Tipo	Descrição
1	Valor inteiro	Número do elemento finito
2	Valor inteiro	Código do material deste elemento finito
3	1	–
4	Valor inteiro	Código da integração numérica
[5..8]	Valor inteiro	Número do nó

%ELEMENT.Q9 (Opcional)

Define os elementos finitos do tipo quadrangular de 9 nós.

Formato: Tabela

Coluna	Valor ou Tipo	Descrição
1	Valor inteiro	Número do elemento finito
2	Valor inteiro	Código do material deste elemento finito
3	1	–
4	Valor inteiro	Código da integração numérica
[5..13]	Valor inteiro	Número do nó

%ELEMENT.TET10 (Opcional)

Define os elementos finitos híbridos do tipo tetraédrico de 10 nós.

Formato: Tabela

Coluna	Valor ou Tipo	Descrição
1	Valor inteiro	Número do elemento finito
2	Valor inteiro	Código do material deste elemento finito
3	1	–
4	Valor inteiro	Código da integração numérica
[5..14]	Valor inteiro	Número do nó

%ELEMENT.HEX27 (Opcional)

Define os elementos finitos híbridos do tipo hexaédrico de 27 nós.

Formato: Tabela

Coluna	Valor ou Tipo	Descrição
1	Valor inteiro	Número do elemento finito
2	Valor inteiro	Código do material deste elemento finito
3	1	–
4	Valor inteiro	Código da integração numérica
[5..31]	Valor inteiro	Número do nó

B.1.9.4 Superfície molhada

%WET.SURFACE.EDGES (Opcional)

Define os lados da superfície molhada da estrutura.

Formato: Tabela

Coluna	Valor ou Tipo	Descrição
1	Valor inteiro	Número do elemento finito
2	Valor inteiro	Número do lado do elemento finito

%WET.SURFACE.FACES (Opcional)

Define as faces da superfície molhada da estrutura.

Formato: Tabela

Coluna	Valor ou Tipo	Descrição
1	Valor inteiro	Número do elemento finito
2	Valor inteiro	Número da face do elemento finito

%WET.SURFACE.MESH (Opcional)

Define que a superfície molhada da estrutura é um elemento finito.

Formato: Variável numérica

Valor	Descrição
1	Superfície molhada é um elemento finito

%WET.SURFACE.CIRCLE (Opcional)

Define que a superfície molhada da estrutura é um círculo.

Formato: Tabela

Coluna	Valor ou Tipo	Descrição
1	Valor em ponto flutuante	Raio do círculo
2	Valor em ponto flutuante	Posição do centro do círculo na direção 1
3	Valor em ponto flutuante	Posição do centro do círculo na direção 2
4	Valor em ponto flutuante	Posição do centro do círculo na direção 3

%WET.SURFACE.ELLIPSE (Opcional)

Define que a superfície molhada da estrutura é uma elipse.

Formato: Tabela

Coluna	Valor ou Tipo	Descrição
1	Valor em ponto flutuante	Raio da elipse na horizontal
2	Valor em ponto flutuante	Raio da elipse na vertical
3	Valor em ponto flutuante	Ângulo de rotação da elipse com relação a horizontal
4	Valor em ponto flutuante	Posição do centro da elipse na direção 1
5	Valor em ponto flutuante	Posição do centro da elipse na direção 2
6	Valor em ponto flutuante	Posição do centro da elipse na direção 3

B.1.9.5 Multiplicadores de Lagrange

%LAGRANGE.MULTIPLIERS.TYPE (Opcional)

Define tipo de multiplicador de Lagrange.

Formato: Variável numérica

Valor	Descrição
1	Multiplicador de Lagrange posicionado no nó do meio do lado ou face do elemento finito
2	Multiplicador de Lagrange posicionado em ângulos pré-definidos
3	Multiplicador de Lagrange posicionado em arcos pré-definidos

%LAGRANGE.MULTIPLIERS.DIVISIONS (Opcional)

Define o número de divisões dos multiplicadores de Lagrange.

Formato: Variável numérica

Valor ou Tipo	Descrição
Valor inteiro	Número de divisões dos multiplicadores de Lagrange

B.2 Exemplo

A seguir um exemplo de um problema de interação fluido-estrutura com dois arquivos neutros, um para a estrutura e outro para o fluido.

B.2.1 Arquivo neutro da estrutura

```
##############################################################################
###                                NEUTRAL FILE                            ###
##############################################################################

%SPACE.DIMENSIONS
2

##############################################################################

%FIELD
u 2 u v

##############################################################################

%MATERIAL.NEO-HOOKEAN
1        0.000000        0.000000

##############################################################################

%THICKNESS
1 0.1

##############################################################################

%EQUATION.TYPE.NONLINEAR
0.0 0.0

##############################################################################

%PROBLEM.TYPE
1

%TIME.INTEGRATION
0. 0. 1.

%TIME.RECORDER
0.

##############################################################################

%QUADRATURE.GAUSS.QUADRILATERAL
1  0 0 0

%QUADRATURE.GAUSS.EDGE
1  0 0 0

##############################################################################

%NODE.COORD
1         0.617557        0.338197        0.000000
2         0.613638        0.341044        0.000000
3         0.616060        0.342803        0.000000
4         0.612982        0.344453        0.000000
5         0.614564        0.347410        0.000000
6         0.609720        0.343891        0.000000
7         0.612325        0.347863        0.000000
8         0.609903        0.346104        0.000000
9         0.610086        0.348317        0.000000
```

```
10        0.605801        0.346738        0.000000
...

##############################################################################

%ELEMENT.Q9
1  1 1 1    6    1    5    9    2    3    7    8    4
2  1 1 1 1025 1113 1088 1037 1074 1096 1061 1028 1065
3  1 1 1 1741 1751 1742 1732 1746 1749 1738 1736 1744
4  1 1 1 1743 1750 1740 1733 1748 1747 1737 1739 1745
5  1 1 1 1085 1111 1024 1038 1095 1073 1029 1063 1066
6  1 1 1   19    6    9   17   10    8   14   15   13
7  1 1 1   18   17    9    5   16   14    7   11   12
8  1 1 1  913 1025 1037  977  954 1028 1005  936  984
9  1 1 1 1058  977 1037 1088 1021 1005 1061 1072 1042
10 1 1 1 1713 1741 1732 1718 1729 1736 1724 1716 1726
...

##############################################################################

%WET.SURFACE.EDGES
1       2
1       1
2       2
2       1
3       1
3       2
4       2
4       1
5       1
5       2
6       1
7       4
8       1
9       4
10      1
...

%WET.SURFACE.MESH
1

##############################################################################

%LAGRANGE.MULTIPLIERS.TYPE
1
```

B.2.2 Arquivo neutro do fluido

```
##############################################################################
###                                NEUTRAL FILE                            ###
##############################################################################

%SPACE.DIMENSIONS
2

%FSI
1

##############################################################################

%IO.GID


##############################################################################

%FIELD
u 2 u v
p 1 p

##############################################################################

%MATERIAL.FLUID
1        0.001000        1.000000
```

```
26
27 ##############################################################################
28
29 %ANALYSIS.TYPE
30 CFD_REAL_TRACTION
31
32 ##############################################################################
33
34 %THICKNESS
35 1 0.1
36
37 ##############################################################################
38
39 %EQUATION.TYPE.NONLINEAR
40 1 1E-6
41
42 ##############################################################################
43
44 %FLOW.TYPE
45 1
46
47 %TIME.INTEGRATION
48 0. 0. 1.
49
50 %TIME.RECORDER
51 0.
52
53 ##############################################################################
54
55 %QUADRATURE.GAUSS.TRIANGLE
56 1 k 3 0 0
57 1 m 6 0 0
58 1 g 6 0 0
59 1 c 6 0 0
60 1 c_t 6 0 0
61
62 %QUADRATURE.GAUSS.EDGE
63 1 G_External 4 0 0
64 1 F_Internal 3 0 0
65
66 ##############################################################################
67
68 %NODE.COORD
69 1         2.200000        0.410000        0.000000
70 2         2.167647        0.410000        0.000000
71 3         2.200000        0.375833        0.000000
72 4         2.167647        0.375833        0.000000
73 5         2.135294        0.410000        0.000000
74 6         2.200000        0.341667        0.000000
75 7         2.154231        0.346716        0.000000
76 8         2.121878        0.380883        0.000000
77 9         2.167895        0.319686        0.000000
78 10        2.102941        0.410000        0.000000
79 ...
80
81 ##############################################################################
82
83 %NODE.DOF.TYPES
84 1        1      1      0
85 2        1      1      0
86 5        1      1      0
87 10       1      1      0
88 17       1      1      0
89 25       1      1      0
90 32       1      1      0
91 44       1      1      0
92 57       1      1      0
93 73       1      1      0
94 89       1      1      0
95 ...
96
97 ##############################################################################
98
99 %NODE.PRESCRIBED.VALUE
100 996         0.000000 0.0
101 997         0.091667 0.0
```

```
998         0.166667 0.0
999         0.225000 0.0
1000        0.266667 0.0
1002        0.291667 0.0
1003        0.300000 0.0
1004        0.291667 0.0
1005        0.266667 0.0
1006        0.225000 0.0
1007        0.166667 0.0
1008        0.091667 0.0
1009        0.000000 0.0

##############################################################################

%ELEMENT.P2P1
1  1 1 1 1009 989 1007 1001 995 1008
2  1 1 1  157 146  112  151 128  133
3  1 1 1  882 913  899  898 907  892
4  1 1 1  555 526  531  541 529  542
5  1 1 1  396 428  414  412 421  406
6  1 1 1  789 759  769  774 762  777
7  1 1 1  329 306  300  317 302  315
8  1 1 1  127 169  153  149 161  138
9  1 1 1  643 673  659  658 666  651
10 1 1 1  904 875  883  890 879  894
...
```