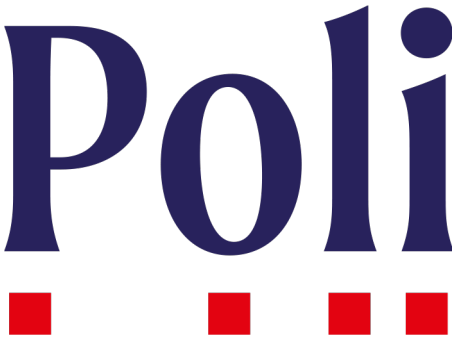


CRIAÇÃO DE UM CÓDIGO EM PYTHON PARA SIMULAR A TRANSFERÊNCIA DE CALOR EM UM PROCESSADOR COMPUTACIONAL EM DIFERENTES ARRANJOS DE CARGAS TÉRMICAS USANDO O MÉTODO DE ELEMENTOS FINITOS

Gustavo Prado Ferreira

Projeto de Graduação apresentado ao Curso de Engenharia Mecânica da Escola Politécnica, Universidade Federal do Rio de Janeiro, como parte dos requisitos necessários à obtenção do título de Engenheiro.

Orientador: Gustavo Rabello dos Anjos

Rio de Janeiro

Julho de 2022

UNIVERSIDADE FEDERAL DO RIO DE JANEIRO

Departamento de Engenharia Mecânica

DEM/POLI/UFRJ

CRIAÇÃO DE UM CÓDIGO EM PYTHON PARA SIMULAR A TRANSFERÊNCIA DE CALOR EM UM PROCESSADOR COMPUTACIONAL EM DIFERENTES ARRANJOS DE CARGAS TÉRMICAS USANDO O MÉTODO DE ELEMENTOS FINITOS

Gustavo Prado Ferreira

PROJETO FINAL SUBMETIDO AO CORPO DOCENTE DO DEPARTAMENTO DE ENGENHARIA MECÂNICA DA ESCOLA POLITÉCNICA DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE ENGENHEIRO MECÂNICO.

Aprovada por:

_____

Prof. Gustavo Rabello dos Anjos, Ph.D.

_____

Prof. Gabriel Lisbôa Verissimo, D.Sc.

_____

Prof. Albino José Kalab Leiroz, Ph.D.

RIO DE JANEIRO, RJ – BRASIL

JULHO DE 2022

Prado Ferreira, Gustavo

CRIAÇÃO DE UM CÓDIGO EM PYTHON PARA SIMULAR A TRANSFERÊNCIA DE CALOR EM UM PROCESSADOR COMPUTACIONAL EM DIFERENTES ARRANJOS DE CARGAS TÉRMICAS USANDO O MÉTODO DE ELEMENTOS FINITOS/ Gustavo Prado Ferreira. – Rio de Janeiro: UFRJ/Escola Politécnica, 2022.

XI, 100 p.: il.; 29, 7cm.

Orientador: Gustavo Rabello dos Anjos

Projeto de Graduação – UFRJ/ Escola Politécnica/ Curso de Engenharia Mecânica, 2022.

Referências Bibliográficas: p. 51 – 52.

1. Transferência de Calor. 2. Método de Elementos Finitos. 3. Processadores. I. Rabello dos Anjos, Gustavo. II. Universidade Federal do Rio de Janeiro, UFRJ, Curso de Engenharia Mecânica. III. CRIAÇÃO DE UM CÓDIGO EM PYTHON PARA SIMULAR A TRANSFERÊNCIA DE CALOR EM UM PROCESSADOR COMPUTACIONAL EM DIFERENTES ARRANJOS DE CARGAS TÉRMICAS USANDO O MÉTODO DE ELEMENTOS FINITOS.

"Not all those who wander are lost." — Bilbo Baggins

Agradecimentos

Gostaria de começar declarando que não sou um bom escritor, não tenho tanta fluência com a arte de descrever emoções e momentos como alguns colegas possuem mas me esforço para me expressar.

Começo agradecendo meus pais, Luiz e Patricia, por me trazerem ao mundo e sempre se dedicarem para que eu tivesse uma boa educação. Agradeço minha irmã Juliana, que sempre foi minha companheira durante minha infância e quem eu sempre pude contar quando tive dificuldades.

Agradeço meus amigos que sempre me apoiaram durante todos esses anos de convívio e estavam presentes tanto nas vitórias quanto nas derrotas. Agradeço à UFRJ por ter me fornecido todo o conhecimento necessário para me tornar um engenheiro, por mais difícil que tenha sido o trajeto até aqui. Agradeço minha namorada Letícia que se fez disponível para ajudar na parte textual deste documento.

Quero também dedicar esse trabalho ao meu avô Sidney que , infelizmente, nos deixou devido a complicações em sua internação por COVID.

Resumo do Projeto de Graduação apresentado à Escola Politécnica/UFRJ como parte dos requisitos necessários para a obtenção do grau de Engenheiro Mecânico

CRIAÇÃO DE UM CÓDIGO EM PYTHON PARA SIMULAR A TRANSFERÊNCIA DE CALOR EM UM PROCESSADOR COMPUTACIONAL EM DIFERENTES ARRANJOS DE CARGAS TÉRMICAS USANDO O MÉTODO DE ELEMENTOS FINITOS

Gustavo Prado Ferreira

Julho/2022

Orientador: Gustavo Rabello dos Anjos

Programa: Engenharia Mecânica

O presente projeto tem como objetivo desenvolver um programa computacional em Python para comparar o comportamento térmico de um processador em funcionamento utilizando um método numérioco em diferentes discretizações. Para otimizar os cálculos de transferência de calor foi utilizado o método dos elementos finitos (MEF) que é capaz de calcular soluções aproximadas do problema térmico para geometrias complexas, como, por exemplo, um processador submetido à situações críticas de funcionamento em temperaturas limites similares às situações propostas pelo fabricante. A solução proposta neste trabalho é mais acessível por ser executada a partir de softwares gratuitos.

Abstract of Undergraduate Project presented to POLI/UFRJ as a partial fulfillment of the requirements for the degree of Mechanical Engineer

CREATION OF A CODE IN PYTHON TO SIMULATE HEAT TRANSFER IN A COMPUTER PROCESSOR IN DIFFERENT THERMAL LOAD ARRANGEMENTS USING THE FINITE ELEMENT METHOD

Gustavo Prado Ferreira

July/2022

Advisor: Gustavo Rabello dos Anjos

Department: Mechanical Engineering

This project aims to develop a computer program in Python to compare the thermal behavior of a computer processor utilizing a numerical method in different discretizations. To optimize the heat transfer calculations, the finite element method (FEM) was used, which is capable of calculating approximate solutions of the thermal problem for complex geometries, such as, for example, a processor subjected to critical operating situations at threshold temperatures similar to situations proposed by the manufacturer. The solution presented in this project is more accessible because it was executed using free of charge software.

Sumário

Lista de Figuras

Capítulo 1

Introdução

O estudo do comportamento da transferência de calor é um assunto que desafia o conhecimento do Engenheiro Mecânico há décadas. A transferência de calor é descrita por uma séria de equações diferenciais difíceis de serem resolvidas e para chegar em algum resultado válido, muitas vezes, é necessário utilizar ferramentas matemáticas chamadas de métodos numéricos.

Os métodos numéricos têm a função de obter soluções aproximadas a fim de aproximar ao máximo os resultados de determinadas equações diferenciais como, por exemplo, o método dos elementos finitos (MEF). Tal método, tem a capacidade de subdividir uma região de interesse, como discos de freio ou processadores, em sub-regiões, ou elementos, com a finalidade de solucionar as equações destes elementos e depois acoplar as respostas para formar o resultado final da região de interesse.

O processador de um computador é como se fosse o cérebro do ser humano, sendo o componente mais importante em qualquer computador. Saber como o processador se comporta termicamente durante o seu uso é de suma importância para a sua manutenção e conservação, uma vez que ele está em constante uso e pode chegar a temperaturas de até 95^{o}C.

Com a finalidade de facilitar o trabalho da troca de calor, as pastas térmicas são componentes pastosos que são aplicados sobre o processador. É importante saber qual é a pasta térmica ideal para cada computador para otimizar o processo e manter o bom funcionamento dos seus componentes.

1.1 Motivação

Hoje em dia a otimização de componentes computacionais é cada vez mais comum. Saber onde está cada perda térmica ou elétrica é algo de interesse para quase todo entusiasta computacional. Pois, com isso, é possível detectar onde está o problema e como solucioná-lo, facilitando a vida do usuário.

1.2 Objetivo

Criar um programa em Python capaz de simular a transferência de calor dentro do processador em diferentes situações e comparar os resultados para averiguar a aplicabilidade do método.

1.3 Metodologia

Com o auxílio da linguagem de programação Python é possível aplicar a solução do problema de transferência de calor Transistente pelo Método de Elementos Finitos com a geração de uma região que representa o processador e que retorne um resultado que seja próximo da realidade.

Com a aplicação do MEF em 3D foi feita uma validação comparando os resultados com as equações analíticas da transferência de calor encontrados em Ozisik (1985). Tais resultados proporcionaram erros relativos aceitáveis para a continuação do estudo.

1.4 Organização do Trabalho

No Capítulo 1 é feita a introdução do trabalho juntamente com a motivação, objetivo, metodologia e a organização do mesmo. O Capítulo 2 revisa a base teórica de transferência de calor, do método numérico de elementos finito, do método de diferenças finitas e do método de volumes finitos. Também são abordadas algumas características dos processadores e das pastas térmicas. No Capítulo 3 é feita a aplicação do método de elementos finitos no cálculo da temperatura no processador a partir das equações de transferência de calor. A validação do método é feita

no Capítulo 4 que serve para demonstrar que o método aplicado é confiável. No Capítulo 5 é demonstrado o resultado da aplicação do Método no estudo da temperatura no processador, juntamente com imagens, demonstrando a distribuição da mesma pela região do processador. O Capítulo 6 conclui o trabalho analisando os resultados obtidos.

Capítulo 2

Revisão Bibliográfica

2.1 Transferência de Calor

Antes de entendermos o que é transferência de calor, iremos apresentar um tópico mais básico: o que é energia? Calor é uma maneira em que a energia se manifesta e, para Ozisik [1], o conceito de energia é utilizado para especificar o estado de um sistema. Não é criada nem destruída apenas varia de forma.

Ozisik [1] também diz que a termodinâmica estuda como a energia térmica se relaciona com outros tipos de energia no universo. Ou seja, é trabalho da Transferência de calor determinar como essas interações refletem na temperatura e na quantidade de energia em um sistema a partir do cálculo das taxas de transferência de calor e taxas de variação de temperaturas.

A ciência da termodinâmica tem como finalidade estudar o comportamento da energia, em forma de calor, em diversos ambientes e situações. Incropera [2] diz que a termodinâmica estuda como energia pode ser transferida de um sistema para sua vizinhança a partir de diversas interações. Em outras palavras, através da definição de um sistema, suas características e sua vizinhança, podemos determinar o comportamento da energia.

Entendendo mais sobre como a energia se comporta no sistema podemos definir de que modo ela é transportada, a partir das características do nosso sistema. Estes modos de transferência de calor são: Condução, convecção e radiação.

2.1.1 Condução

A condução é a transferência de energia feita em nível atômico, onde a partícula mais energética transfere sua energia para uma partícula, próxima, com menos energia. Energias moleculares maiores representam temperaturas maiores que se traduzem em moléculas mais agitadas que colidem com moléculas em sua vizinhança e transferem parte de sua energia para as mesmas.

$$q_k = -k\frac{\partial T}{\partial x} \tag{2.1}$$

A equação (2.1) é conhecida como Lei de Fourier. Ela calcula o Fluxo térmico q_k (W/m^2) a partir do gradiente de temperatura na direção de interesse $\partial T/\partial x$ (K/m) e que depende da condutividade térmica k $(W/(mK))$ que é uma característica do material em questão. O sinal negativo aparece na equação para poder representar o fato da energia estar sendo transferida na direção decrescente da temperatura.

2.1.2 Convecção

Convecção é a transferência de calor feita a partir do movimento do sistema. Diferente do modo anterior, este modo está intrinsecamente ligado ao transporte de calor por difusão (movimento microscópico) e pelo movimento de partículas (movimento macroscópico). Ocorre principalmente entre um fluido em movimento e uma superfície, estando os dois em diferentes temperaturas. Como se trata de movimento de fluidos, o conceito de camada limite, que é a região do fluido em que a velocidade de escoamento varia de zero até um valor finito, se torna importante para contextualizarmos o conceito da camada limite térmica, que tem uma definição similar. A camada limite térmica é a região em que a temperatura varia da temperatura da superfície até a temperatura do fluido. (Incropera[2])

$$q_h = h(T_s - T_\infty) \tag{2.2}$$

Na equação (2.2) podemos observar como é calculado o fluxo de calor por convecção q_h (W/m^2). O parâmetro h (W/m^2K) é chamado de coeficiente de transferência de calor por convecção e depende das condições da camada limite. O termo

T_s (K) corresponde ao valor da temperatura da superfície enquanto T_∞ (K) corresponde a temperatura do fluido.

2.1.3 Radiação

Diferente dos modos anteriores, a radiação não é transmitida por partículas e sim por ondas eletromagnéticas que energizam as partículas de um sistema, não sendo necessário um meio físico para transmitir calor. Pode ser definida como a energia emitida por todo corpo a uma temperatura não-nula (Incropera[2]).

$$q_r = \varepsilon\sigma(T_s^4 - T_v^4) \tag{2.3}$$

Em (2.3), o fluxo de calor por radiação q_r (W/m^2) é definido pela interação entre a constante de Stefan-Boltzmann, $\sigma = 5,67.10^{-8}\ W/(m^2\ K^4)$, a emissividade ε que varia de acordo com o material $(0 \leq \varepsilon \leq 1)$ e as temperaturas T_s e T_v que são, respectivamente, a temperatura da superfície radioativa e a temperatura da vizinhança (K).

2.2 Método de Elementos Finitos

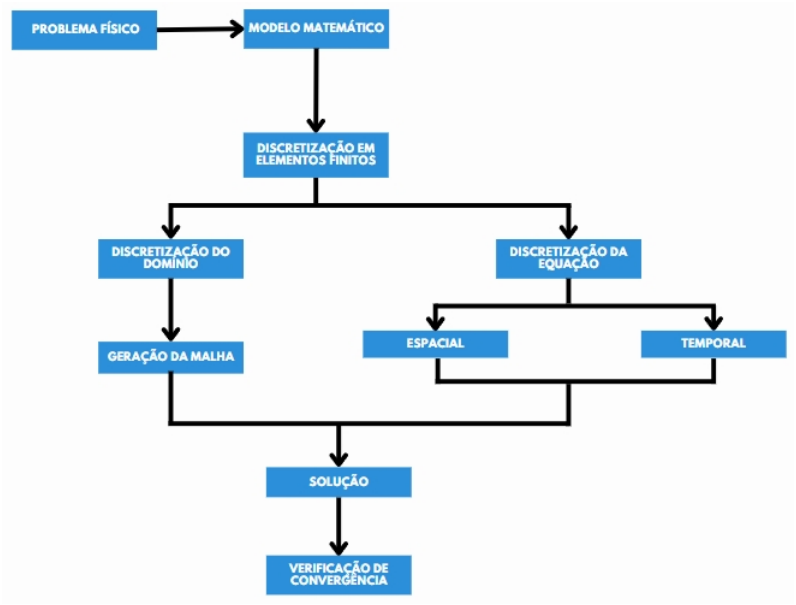
Na Engenharia é preciso encontrar ferramentas para resolvermos problemas numéricos que envolvem derivadas. Uma dessas ferramentas é o Método de Elementos Finitos (ou MEF) que, para Lewis [3], é um método numérico que determina soluções aproximadas para uma extensa variedade de diferentes problemas de engenharia. O método é bem conhecido por ser bem flexível em relação a sua abordagem e execução e é muito utilizado na área de análise de transferência de calor.

O MEF considera que a região de interesse é feita por muitas sub regiões ou elementos interconectados e que a solução do problema para essa sub região dá uma pequena amostra de como a solução, no geral, se apresenta. Ou seja, o MEF reduz as equações complexas das soluções das equações diferenciais parciais em muitas equações lineares ou não-lineares. Desta forma, a discretização do MEF reduz o problema contínuo, que possui infinitas incógnitas, em um problema com um número finito de incógnitas em pontos específicos da região de interesse, chamados

de (nodes). Sendo assim possível descrever o problema a partir de sub-regiões, ou elementos, de domínios ou regiões bem complexas. (Lewis [3])

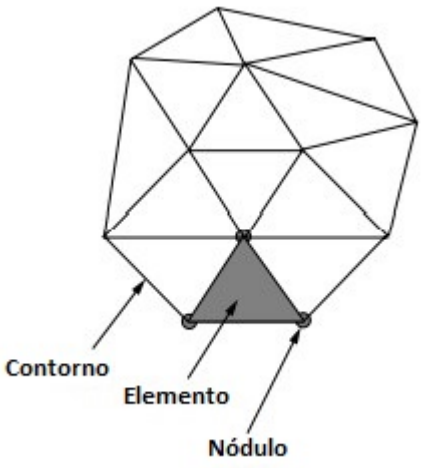
$$[\mathbf{K}]\{\mathbf{T}\} = [\mathbf{f}] \tag{2.4}$$

A equação (2.4) é a equação que determina o valor do vetor da temperatura T a partir da matriz global de forma K, do vetor de forma f.



[3]

Figura 2.1: Exemplo de Modelo numérico para os cálculos do MEF.



[3]

Figura 2.2: Malha de Elementos Finitos com elementos, nódulos e contornos.

Em Lewis[3] é possível encontrar o passo a passo da utilização do MEF que se resume em:

1 - **Discretizar o contínuo**: Dividir a região em elementos não sobrepostos.

2 - **Selecionar as equações de forma**: Escolher as equações de interpolação adequadas.

3 - **Formular as equações de elemento**: Determinar as matrizes de equações($[\mathrm{K}]_e$) e os vetores de carga ($\{\ \mathrm{f}\ \}_e$) que expressem os elementos individuais da nossa região.

4 - **Fazer o *Assembling* das equações de elemento em um sistema**: Montar a matriz global [K] a partir das matrizes de cada elemento, melhor descrito em Lewis[3].

5 - **Solucionar o sistema**: Resolver o sistema linear de tipo $[\mathrm{A}]\{\mathrm{x}\} = [\mathrm{b}]$, similar a equação (2.4).

6 - **Calcular as quantidades secundárias**: Calcular a taxa de transferência de calor a partir da temperatura em cada ponto.

2.3 Método de Diferenças Finitas

Além do MEF, outra ferramenta numérica muito utilizada na análise de problemas numéricos que possuem derivadas é o Método de Diferenças Finitas (ou MDF). O MDF consiste em aproximar a derivada de uma função via fórmulas discretas que requerem apenas um conjunto finito de pares ordenados (UFRGS [4]).

O MDF é utilizado na discretização temporal do nosso problema térmico por ser mais fácil de se aplicar e executar do que o MEF. O MEF permite a utilização de outros métodos em conjunto com ele e como a discretização temporal é simples, em relação a discretização espacial, é mais vantajoso aplicar o MDF (mesmo que o MEF também seja capaz de resolver o problema térmico transiente).

$$\frac{dT}{dt} = \lim_{\Delta t \to 0} \frac{T(t + \Delta t) - T(t)}{\Delta t} \tag{2.5}$$

$$\frac{dT}{dt} \approx \frac{T(t + \Delta t) - T(t)}{\Delta t} \tag{2.6}$$

A equação (2.5) é a definição de derivada. Aplicando o MDF chegamos na aproximação da derivada (2.6).

2.4 Método de Volumes Finitos

O Método de Volumes Finitos, ou MVF é um terceiro método numérico existente para resolver equações diferenciais. Diferente do MEF e do MDF, o MVF se baseia em uma aproximação mais física do que puramente matemática. Sua implementação consiste em discretizar a região de interesse em volumes e analisar os fluxos que percorrem esses volumes em função dos volumes vizinhos. Sua aplicação é menos flexível em relação ao MEF.

2.5 Processador

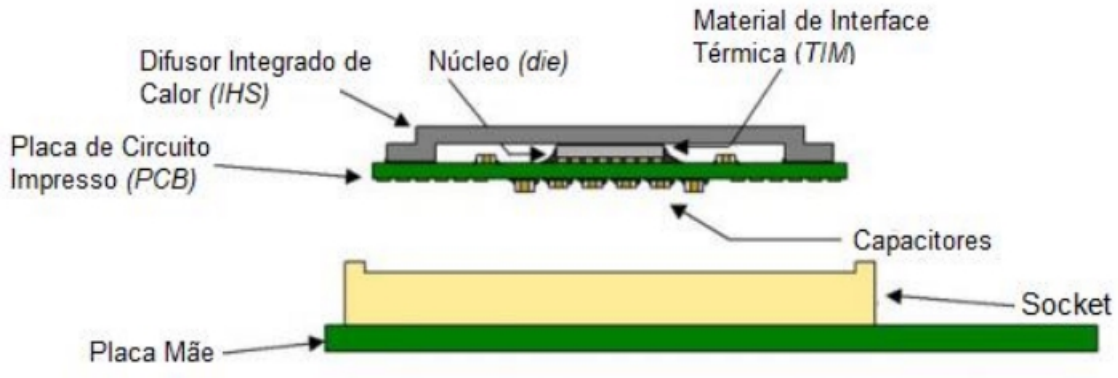
O processador (ou CPU - *Central Processing Unit*) por definição é "um aparelho eletrônico muito pequeno que controla todas os componentes de um computador, ou dispositivo similar, sendo responsável pelo seu funcionamento"(Dicio [5]). Um processador é composto, no geral, pelos seguintes componentes (Marcelo Schuh [6]):

1 - IHS (*Integrated Heat Spreader*): Tampa metálica que funciona como um dissipador de calor.

2 - TIM (*Thermal Interface Material*): Material intersticial entre o IHS e o Die que facilita a condução de calor entre os dois.

3 - Die (*Núcleo*): Onde ocorre o processamento e a lógica computacional. Também é a fonte de calor do componente.

4 - PCB (*Printed Circuit Board*): Liga o Die e a Placa mãe com a finalidade de conectar o processador ao resto do computador.



[6]

Figura 2.3: Componentes de um Processador e direcionamento do Fluxo de Calor

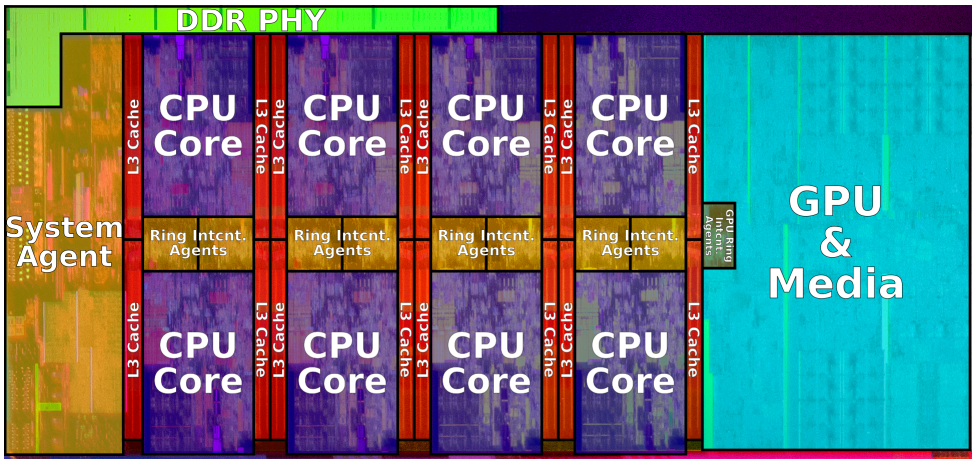


Figura 2.4: Arquitetura do Die - Coffee Lake

[7]

[7]

Figura 2.5: Cooler/Resfriador padrão da Intel

Em contato com o processador é posicionado um trocador de calor (resfriador ou cooler) que ajuda no resfriamento do componente por inteiro. Normalmente esses resfriadores são compostos por um radiador acoplado a uma ventoinha.

Também é necessário citar que o Die do processador utilizado na simulação possui a arquitetura Coffee Lake, descrita na figura (2.4). Essa construção apresenta 8 núcleos de processamento que melhoram o desempenho do computador permitindo que mais funções sejam executadas ao mesmo tempo.

2.6 Pasta Térmica

Pasta térmica é o nome comumente utilizado para se referir a Compostos Térmicos Poli Sintéticos de Alta Densidade. Tais compostos são utilizados em alta

demanda na indústria tecnológica para auxiliar na transferência de calor entre duas superfícies. Iremos focar na área de contato entre o processador e o resfriador de um computador, apesar de aparentarem ter superfícies lisas, microscopicamente pode-se ver que não é bem assim que funciona. Sabendo disso, é aplicada a pasta térmica entre as duas superfícies de interesse. A pasta serve como ponte de ligação para o calor passar de uma superfície para a outra, facilitando assim a transferência de calor entre os componentes. (Artic Silver [8])

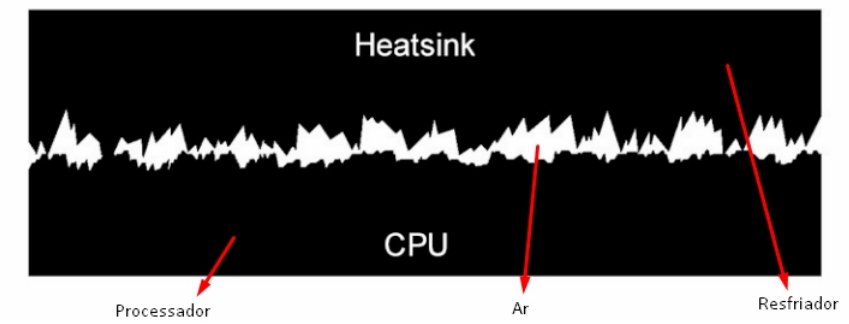
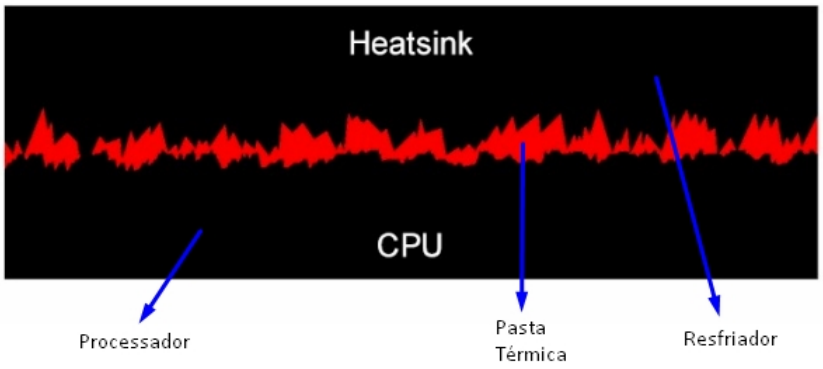


Figura 2.6: Contato microscópico entre as superfícies e lacunas aparentes (em branco)

[8]



[8]

Figura 2.7: Áreas que estavam vazias agora preenchidas com a pasta térmica (em vermelho)

A pasta térmica é aplicada entre o processador e o resfriador e as figuras (2.6) e (2.7) demonstram uma visualização microscópica de como as superfícies do processador e do resfriador interagem. A presença da Pasta Térmica auxilia na transferência de calor na superfície do processador e entre o processador e o resfriador.

Capítulo 3

Metodologia

3.1 Discretizações do Processador

Inicialmente se faz necessário a discretização da região de interesse para modelar o problema pelo MEF. Utilizando como base o processador Intel Core i9-9900K e as dimensões de seus componentes (der8aure[9]) é possível esboçar um modelo em 3D no programa Gmsh (ver. 3.0.6) que também nos proporciona uma malha tridimensional definida pelo modelo, discretizando a região de interesse e salvando a malha do modelo em um arquivo ".msh".

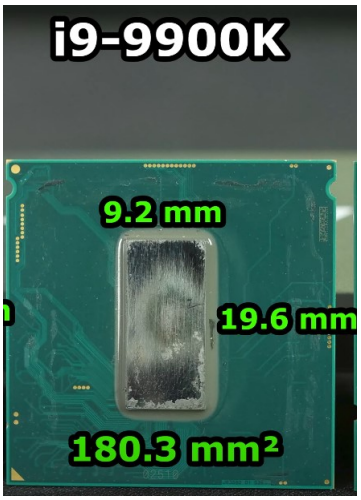


Figura 3.1: Dimensões da parte interna do Processador
[9]

Sabendo da existência dos diferentes componentes no processador e suas posições no modelo, podemos aplicar as propriedades termodinâmicas e as condições iniciais

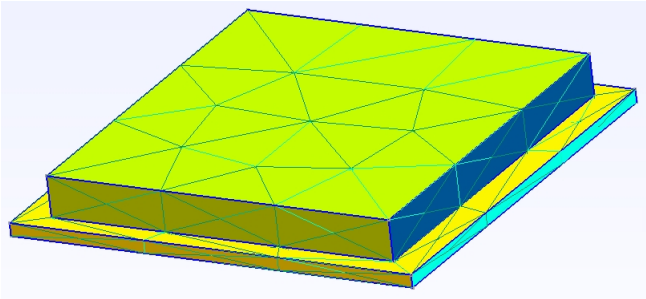


Figura 3.2: Modelo do Processador no Programa Gmsh

adequadas através do programa em Python que, ao mesmo tempo, construirá as equações e matrizes do MEF dependendo da posição de cada ponto. A lógica do programa também aplicará as propriedades da pasta térmica, que é comumente espalhada na face superior do IHS onde há o contato do processador com o resfriador.

Com a finalidade de comparar a diferença nos resultados, serão feitas 4 malhas baseadas no mesmo modelo do processador porém com número de elementos diferentes como é possível observar nas figuras (3.3), (3.4), (3.5) e (3.6).

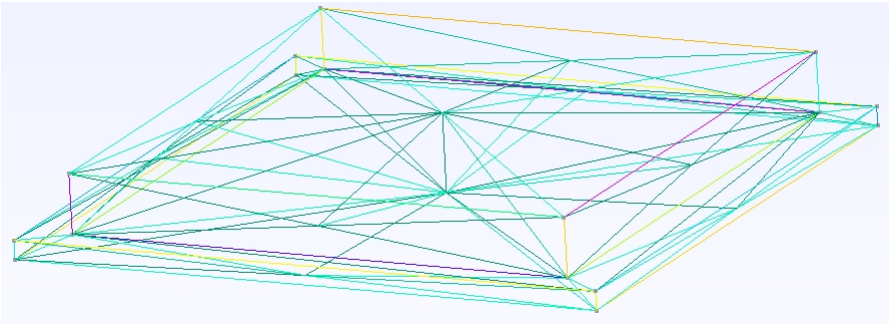


Figura 3.3: Malha com 144 elementos

3.2 Equacionamento

O estudo da transferência de calor na região de interesse será feito somente sobre o modo de condução. A região de interesse é composta, na sua maioria, por sólidos que estão estacionários e em constante contato. O fluido que aparece na região de interesse está estacionário. Também não há presença de uma fonte de radiação relevante para o problema. Com isso o único modo relevante é o de condução.

$$\alpha \nabla^2 T + \frac{Q}{\rho c_v} = 0 \tag{3.1}$$

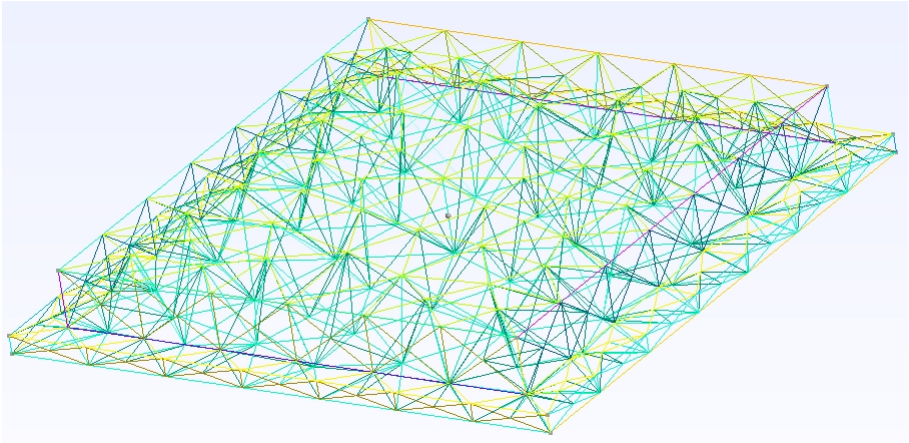


Figura 3.4: Malha com 1.470 elementos

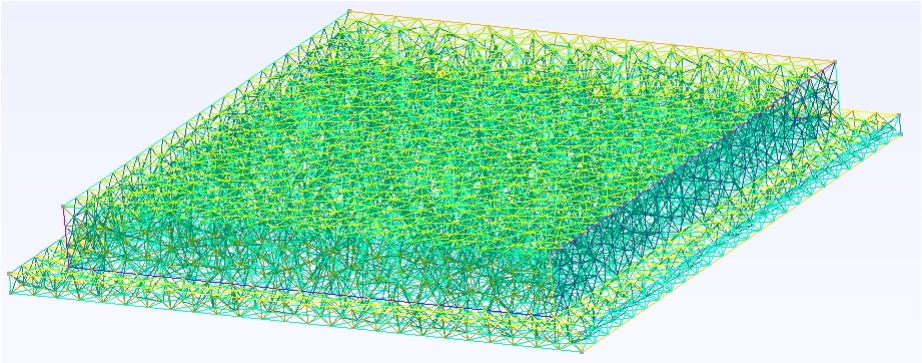


Figura 3.5: Malha com 14.334 elementos

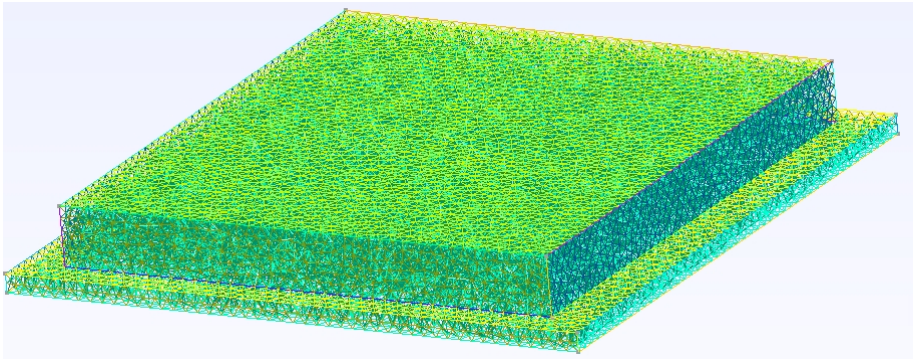


Figura 3.6: Malha com 70.507 elementos

$$\alpha = \frac{K}{\rho c_v} \tag{3.2}$$

$$\nabla^2 T = \left(\frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2} + \frac{\partial^2 T}{\partial z^2} \right) \tag{3.3}$$

A equação (3.1) é a equação de difusão de calor que é utilizada para calcular o gradiente de temperatura devido ao fluxo de calor condutivo em um sólido. Na equação Q representa uma taxa volumétrica de calor (W/m^3), ρ é a massa específica do material (kg/m^3), c_v é o calor específico do meio $(J/(kgK))$, α é a difusividade térmica do material (m^2/s). A equação (3.3) representa a definição do Laplaciano em 3D.

A equação (3.1) é utilizada para solucionar o problema térmico permanente, ou seja, sem a sua evolução no tempo. Como é preciso incluir a variação temporal nos cálculos a equação passa a ser escrita desta forma:

$$\frac{\partial T}{\partial t} = \alpha \nabla^2 T + \frac{Q}{\rho c_v} \tag{3.4}$$

$$\frac{\partial T}{\partial t} - \alpha \nabla^2 T - \frac{Q}{\rho c_v} = 0 \tag{3.5}$$

O termo $\frac{\partial T}{\partial t}$ (K/s) é a taxa de variação da temperatura no tempo.

As equações (3.4) 3 (3.5) são iguais, porém, a equação (3.5) será mais fácil de trabalhar pois, antes de aplicarmos o passo a passo do MEF é necessário preparar a equação, ou seja, transformar ela de sua Forma Forte (Equação diferencial original) para sua Forma Fraca (Equação integral-diferencial) através do cálculo variacional.

$$\int_\Omega w \left(\frac{\partial T}{\partial t} - \alpha \nabla^2 T - \frac{Q}{\rho c_v} \right) d\Omega = 0 \tag{3.6}$$

Para a equação acima temos que Ω representa o domínio do problema, ou seja, a região do processador e w é a função peso. A equação (3.6) também pode ser considerada a forma fraca inicial do problema.

Separando cada termo da equação (3.6) em uma soma de 3 integrais:

$$\int_\Omega w \frac{\partial T}{\partial t} d\Omega - \int_\Omega w \alpha \nabla^2 T d\Omega - \int_\Omega w Q' d\Omega = 0 \tag{3.7}$$

Em (3.7) em diante, $Q' = \frac{Q}{\rho c_v}$.

É preciso reduzir a ordem do segundo termo da equação (3.7) integrando por partes, logo:

$$\int_\Omega w \alpha \nabla^2 T d\Omega = \int_\gamma w \alpha \nabla T d\gamma - \int_\Omega \alpha \nabla T \nabla w d\Omega \tag{3.8}$$

Temos que, na equação (3.8), γ representa o contorno do domínio e Ω representa o miolo do mesmo.

Para facilitar a representação temos que $\int_\Omega$ significa integrais triplas no domínio 3D. $\int_\gamma$ representa integrais duplas no domínio 2D, ou seja, das faces externas do domínio 3D (contorno).

A equação (3.7) se torna:

$$\int_\Omega w\frac{\partial T}{\partial t}d\Omega - \int_\gamma w\alpha\nabla T d\gamma + \int_\Omega \alpha\nabla T\nabla w d\Omega - \int_\Omega wQ' d\Omega = 0 \tag{3.9}$$

Utilizando o Método de Galerkin para escolher as equações de forma do nosso problema (Lewis [3]) nos deparamos com as seguintes considerações:

$$T(x,y,z,t) \approx \Sigma N_i(x,y,z)T_i(t) \tag{3.10}$$

$$w(x,y,z) \approx \Sigma N_j(x,y,z)w_j \tag{3.11}$$

$$Q'(x,y,z) \approx \Sigma N_i(x,y,z)Q'_i \tag{3.12}$$

$$N_i = N_j \tag{3.13}$$

Os termos w_j e Q'_i são constantes e o termo T_i é uma função que varia com o tempo.

Substituindo as equações acima em (3.9) temos:

$$\int_\Omega w\frac{\partial T}{\partial t}d\Omega \to \Sigma\Sigma w_j\frac{\partial T_i}{\partial t}\int_\Omega N_iN_j d\Omega \tag{3.14}$$

$$\int_\Omega wQ' d\Omega \to \Sigma\Sigma w_jQ'_i\int_\Omega N_iN_j d\Omega \tag{3.15}$$

O termo $\int_\Omega N_iN_j d\Omega$ nas equações (3.15) e (3.16) são conhecidos como as integrais de massa e resultam na construção da Matriz de Massa $M_{i,j}$.

É importante definir qual será o tipo de condição de contorno adotada nessa aplicação do MEF. Das existentes, a que mais se aproxima do caso real é a de Neumann, ou seja, a temperatura nas regiões de contorno não são fixas.

$$\int_\gamma w\alpha\nabla T d\gamma - \int_\Omega \alpha\nabla T\nabla w d\Omega \rightarrow \Sigma\Sigma\alpha w_j T_i \left(\int_\gamma N_j\nabla N_i d\gamma - \int_\Omega \nabla N_i\nabla N_j d\Omega\right) \tag{3.16}$$

O termo entre parênteses na equação (3.16) representa a integral de rigidez. Tal integral se torna a Matriz de Rigidez $K_{i,j}$ que tem forma conhecida. Nota-se, também, que o termo w_j aparece nas 3 partes da equação. Logo, como a equação se iguala a zero o termo pode ser cortado.

Arrumando as equações apresentadas resulta na seguinte equação:

$$M_{i,j}\frac{\partial T_i}{\partial t} + K_{i,j}T_i - M_{i,j}Q'_i = 0 \tag{3.17}$$

Ainda resta a presença do termo Transiente, que é descrito pelo MDF.

$$M_{i,j}\frac{(T_i^{n+1} - T_i^n)}{\Delta t} + K_{i,j}T_i - M_{i,j}Q'_i = 0 \tag{3.18}$$

Arrumando a equação (3.18):

$$\left(\frac{M_{i,j}}{\Delta t} + \beta K_{i,j}\right) T_i^{n+1} = \left(\frac{M_{i,j}}{\Delta t} - (1-\beta)K_{i,j}\right) T_i^n + M_{i,j}Q'_i \tag{3.19}$$

Na equação (3.19) há a presença da constante β que representa qual método será utilizado no MDF: se $\beta = 1,0$ o método é explícito; se $\beta = 0$ o método é implícito; se $\beta = 0,5$ o método é de Crank-Nicholson (uma média entre o implícito e explícito).

As matrizes elementares k^e e m^e, para problemas em 3D com elementos tetraédricos, são matrizes com tamanho e formato conhecidos e só dependem do tipo de malha que será usado.

As matrizes elementares possuem as seguintes estruturas:

$m^e = \frac{V}{20}\begin{bmatrix} 2 & 1 & 1 & 1 \\ 1 & 2 & 1 & 1 \\ 1 & 1 & 2 & 1 \\ 1 & 1 & 1 & 2 \end{bmatrix}$ e $k^e = V\mathbf{B}^T\mathbf{B}$.

V corresponde ao volume do elemento e a matriz $\mathbf{B}$ é calculada a partir de diversas equações provenientes das coordenadas de cada ponto do elemento com descrita a seguir.

$$\mathbf{B} = \frac{1}{6V} \begin{bmatrix} b_i & b_j & b_k & b_l \\ c_i & c_j & c_k & c_l \\ d_i & d_j & d_k & d_l \end{bmatrix}$$

Na matriz anterior os valores de (b), (c) e (d) variam de acordo com as coordenadas de cada ponto em relação com as coordenadas dos outros pontos pertencentes ao mesmo elemento para cada elemento. Esses cálculos ficam mais claros nos programas presentes no Apêndice A.

Capítulo 4

Validação do Método

A fim de determinar a aplicabilidade do método é necessário comparar os resultados numéricos com valores reais. No caso da transferência de calor o valor real pode ser determinado a partir das soluções das Equações Analíticas. Em David e Ozisik [10] encontramos alguns métodos de solucionamento analítico de diversas aplicações e condições de contorno da equação de transferência de calor. Um dos exemplos resolve a equação tridimensional transiente com diferentes condições de contorno e esse exemplo será demonstrado a seguir.

4.1 Equacionamento Analítico

$$\frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2} + \frac{\partial^2 T}{\partial z^2} = \frac{1}{\alpha}\frac{\partial T}{\partial t} \quad 0 < (x, y, z) < 1 \tag{4.1}$$

$$T(x = 0) = 0 \quad \text{em} \quad t > 0 \tag{4.2}$$

$$T(y = 0) = 0 \quad \text{em} \quad t > 0 \tag{4.3}$$

$$T(z = 0) = 0 \quad \text{em} \quad t > 0 \tag{4.4}$$

$$T(x = L) = 0 \quad \text{em} \quad t > 0 \tag{4.5}$$

$$\frac{\partial T}{\partial y} = 0 \quad \text{em} \quad t > 0 \quad \text{e} \quad y = 1 \tag{4.6}$$

$$-k\frac{\partial T}{\partial z} = hT \quad \text{em} \quad t > 0 \quad \text{e} \quad z = 1 \tag{4.7}$$

$$T(x, y, z, t = 0) = F(x, y, z) = T_0 \quad \text{em} \quad 0 \leq (x, y, z) \leq 1 \tag{4.8}$$

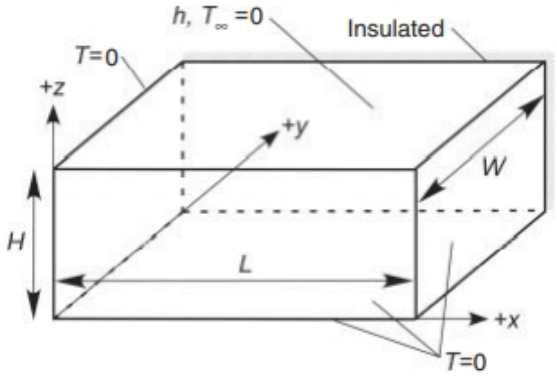


Figura 4.1: Descrição espacial do problema

[10]

As condições iniciais do problema definem a temperatura inicial $T_0 = 100°C$, a temperatura do ambiente $T_\infty = 0°C$, as condições de contorno do problema, descritas pelas equações anteriores, e a geometria do problema, que, no caso da nossa validação, será um cubo de lado unitário. Como a finalidade da validação é comparar o resultado com o valor da solução numérica, as propriedades termodinâmicas também foram definidas com valores unitários para facilitar no equacionamento.

Observamos, então, que a formulação do problema possui condições de contorno homogêneas e uma única condição inicial não homogênea. Isso nos proporciona uma condição propícia para a aplicação da separação de variáveis.

$$T(x, y, z, t) = X(x)Y(y)Z(z)\Gamma(t) \tag{4.9}$$

A solução de T equivale ao produto das soluções de X, Y, Z e Γ. Aplicando a equação (4.9) na equação (4.1) temos:

$$\frac{1}{X}\frac{d^2X}{dx^2} + \frac{1}{Y}\frac{d^2Y}{dy^2} + \frac{1}{Z}\frac{d^2Z}{dz^2} = \frac{1}{\alpha\Gamma}\frac{d\Gamma}{dt} = -\lambda^2 \tag{4.10}$$

Como temos uma igualdade, podemos separar as equações igualando ambas a mesma constante. Isso permite que ambos os lados possam ser resolvidos separadamente. Primeiro, resolvendo a parte da equação dependente do tempo, temos que:

$$\frac{d\Gamma}{dt} + \alpha\lambda^2\Gamma = 0 \tag{4.11}$$

$$\Gamma(t) = C_1 e^{-\alpha\lambda^2 t} \tag{4.12}$$

A constante λ será definida pelos resultados do equacionamento espacial. Rearrumando a equação (4.10) podemos solucionar as seções espaciais separadamente.

$$\frac{1}{Y}\frac{d^2Y}{dy^2} + \frac{1}{Z}\frac{d^2Z}{dz^2} + \lambda^2 = -\frac{1}{X}\frac{d^2X}{dx^2} = \beta^2 \tag{4.13}$$

Com a equação (4.13) podemos solucionar a porção do problema referente a dimensão x utilizando suas devidas condições de contorno.

$$X_n(x) = C_3 sen(\beta_n x) \quad \text{com} \quad \beta_n = n\pi \quad \text{para} \quad n = 0, 1, 2, ... \tag{4.14}$$

Novamente, rearrumando a equação (4.13) temos:

$$\frac{1}{Z}\frac{d^2Z}{dz^2} + \lambda^2 - \beta^2 = -\frac{1}{Y}\frac{d^2Y}{dy^2} = \eta^2 \tag{4.15}$$

Solucionando a porção referente a dimensão y utilizando as devidas condições de contorno temos que:

$$Y_m(y) = C_5 sen(\eta_m y) \quad \text{com} \quad \eta_m = \frac{(2m+1)\pi}{2} \quad \text{para} \quad m = 0, 1, 2, ... \tag{4.16}$$

Com isso nos resta o seguinte equacionamento:

$$\frac{1}{Z}\frac{d^2Z}{dz^2} + \lambda^2 - \beta^2 - \eta^2 = 0 \tag{4.17}$$

Considerando $\mu^2 = \lambda^2 - \beta^2 - \eta^2$ podemos solucionar para a dimensão z.

$$Z_p(z) = C_7 sen(\mu_p z) \tag{4.18}$$

No caso da dimensão z, devido à complexidade da sua condição de contorno em $z = 1$ temos que o valor de μ_p é igual às raízes da seguinte equação transcendental:

$$\frac{\mu_p}{tan(\mu_p)} = -1 \quad \text{com} \quad p = 1, 2, 3, ... \tag{4.19}$$

Podemos agora juntar as soluções das equações (4.12), (4.14), (4.16) e (4.18) para determinar o valor de T.

$$T_{nmp}(x,y,z,t) = C_{nmp} sen(\beta_n x) sen(\eta_m y) sen(\mu_p z) e^{-\alpha \lambda_{nmp}^2 t} \tag{4.20}$$

Na equação (4.20) temos a introdução do termo $C_{nmp} = C_1 C_3 C_5 C_7$ e agora definimos λ da seguinte forma:

$$\lambda_{nmp}^2 = \beta_n^2 + \eta_m^2 + \mu_p^2 \tag{4.21}$$

Tendo como solução geral:

$$T(x,y,z,t) = \sum_{n=0}^{\infty} \sum_{m=0}^{\infty} \sum_{p=1}^{\infty} C_{nmp} sen(\beta_n x) sen(\eta_m y) sen(\mu_p z) e^{-\alpha \lambda_{nmp}^2 t} \tag{4.22}$$

Para definir C_{nmp} utilizamos a condição inicial aplicada na equação (4.22).

$$F(x,y,z) = T_0 = \sum_{n=0}^{\infty} \sum_{m=0}^{\infty} \sum_{p=1}^{\infty} C_{nmp} sen(\beta_n x) sen(\eta_m y) sen(\mu_p z) \tag{4.23}$$

Usando a propriedade da ortogonalidade das 3 autofunções sobre seus respectivos intervalos nos deparamos com a seguinte equação que define a nossa constante.

$$C_{nmp} = \frac{\int_{x=0}^{x=1} \int_{y=0}^{y=1} \int_{z=0}^{z=1} T_0 sen(\beta_n x) sen(\eta_m y) sen(\mu_p z)\, dz\, dy\, dx}{\int_{x=0}^{x=1} sen^2(\beta_n x)\, dx \int_{y=0}^{y=1} sen^2(\eta_m y)\, dy \int_{z=0}^{z=1} sen^2(\mu_p z)\, dz} \tag{4.24}$$

Com isso podemos calcular a temperatura do nosso corpo cúbico de lado unitário, $T_0 = 100°C$ e $T_\infty = 0°C$ com as condições de contorno propostas.

4.2 Modelagem

Com os equacionamentos e condições de contorno definidos podemos construir uma lógica em Python que calcule os valores da temperatura em cada ponto da malha utilizada no método numérico com o passar do tempo.

```
z1 = [2.0288, 4.9132, 7.9787, 11.0856, 14.2075, 17.3364] #raizes de
    x*cotg(x)=-1


for t in range(0,nIter):
 if t == 0:
  for ponto in range(0,npoints):
```

```
  TAnalitic = T0*np.ones(npoints, dtype='float')
  if ponto in cc2D:
   if X[ponto] == 0:
    TAnalitic[ponto] = 0.0

   if X[ponto] == 1:
    TAnalitic[ponto] = 0.0

   if Y[ponto] == 0:
    TAnalitic[ponto] = 0.0

   if Z[ponto] == 0:
    TAnalitic[ponto] = 0.0

else:
 TAnalitic = np.zeros(npoints, dtype='float')
 for ponto in range(0, npoints):

  for n in range(1,7):

   for m in range(1,7):

    for p in range(0,6):

    xis = n*np.pi

    yip = (2*m + 1)*np.pi/2.0

    zet = z1[p]

    lamb = np.sqrt(xis**2 + yip**2 + zet**2)

    a =  T0 *
```

```
        ((1-np.cos(zet))*(1-np.cos(yip))*(-np.cos(xis)+1))/(xis*yip*zet)

    b =
        (1-0.5*(1+np.sin(2*xis)/(2*xis)))*(1-0.5*(1+np.sin(2*yip)/(2*yip)))
    *(1-0.5*(1+np.sin(2*zet)/(2*zet)))

    C = a/b

    TAnalitic[ponto] += C *
        np.sin(xis*X[ponto])*np.sin(yip*Y[ponto])*np.sin(zet*Z[ponto])
    *np.exp(-alpha*(lamb**2)*dt*t)
```

A seção do programa acima demosntra como é feito o calculo da temperatura. Em David e Osizik [10] podemos encontrar as 6 primeiras raízes positivas de μ_p e, com isso foi usado somente as 6 primeiras valores não nulos de β_n e η_m. Sendo assim, o somatório descrito na equação (4.22) terá 216 elementos ($n \in (1,6)$, $m \in (1,6)$ e $p \in (1,6)$). A lista z1 contém os valores de μ_p enquanto β_n, η_m e λ_{nmp} são calculados nos for loops.

A lista cc2D conté todos os pontos presentes nas faces do contorno da malha. Ela é utilizada para fixar as condições de contorno nos devidos pontos.

TAnalitic é um vetor construído de forma que cada número em seu interior seja correspondente com o ponto de mesmo número na malha. Os valores subsequentes de temperaturas são calculados ponto por ponto.

A constante C é calculada a partir do desenvolvimento da equação (4.24). Seu valor é definido para cada um dos 216 arranjos possíveis dos valores de n,m e p.

A construção lógica do MEF se dá de outra maneira, demosntrada na seção de código abaixo.

```
for i in range(0, ne):
    v1 = IEN[i,0]
    v2 = IEN[i,1]
    v3 = IEN[i,2]
    v4 = IEN[i,3]
```

```
b1 = ((Y[v2] - Y[v4]) * (Z[v3] - Z[v4])) - ((Y[v3] - Y[v4]) * (Z[v2]
    - Z[v4]))
b2 = ((Y[v3] - Y[v4]) * (Z[v1] - Z[v4])) - ((Y[v1] - Y[v4]) * (Z[v3]
    - Z[v4]))
b3 = ((Y[v1] - Y[v4]) * (Z[v2] - Z[v4])) - ((Y[v2] - Y[v4]) * (Z[v1]
    - Z[v4]))
b4 = (b1 + b2 + b3)*(-1.0)

c1 = ((X[v3] - X[v4]) * (Z[v2] - Z[v4])) - ((X[v2] - X[v4]) * (Z[v3]
    - Z[v4]))
c2 = ((X[v1] - X[v4]) * (Z[v3] - Z[v4])) - ((X[v3] - X[v4]) * (Z[v1]
    - Z[v4]))
c3 = ((X[v2] - X[v4]) * (Z[v1] - Z[v4])) - ((X[v1] - X[v4]) * (Z[v2]
    - Z[v4]))
c4 = (c1 + c2 + c3)*(-1.0)

d1 = ((X[v2] - X[v4]) * (Y[v3] - Y[v4])) - ((X[v3] - X[v4]) * (Y[v2]
    - Y[v4]))
d2 = ((X[v3] - X[v4]) * (Y[v1] - Y[v4])) - ((X[v1] - X[v4]) * (Y[v3]
    - Y[v4]))
d3 = ((X[v1] - X[v4]) * (Y[v2] - Y[v4])) - ((X[v2] - X[v4]) * (Y[v1]
    - Y[v4]))
d4 = (d1 + d2 + d3)*(-1.0)

volume = (1.0/6.0) * np.linalg.det([[1,X[v1],Y[v1],Z[v1]],
                                    [1,X[v2],Y[v2],Z[v2]],
                                    [1,X[v3],Y[v3],Z[v3]],
                                    [1,X[v4],Y[v4],Z[v4]]])

melem = (volume/20.0) * np.array([[2,1,1,1],
                                  [1,2,1,1],
                                  [1,1,2,1],
                                  [1,1,1,2]])
```

```
B = (1.0/(volume * 6.0)) * np.array([[b1,b2,b3,b4],
                                     [c1,c2,c3,c4],
                                     [d1,d2,d3,d4]])


kelem = alpha*volume*(np.transpose(B)@B)

for ilocal in range(0,4):
    iglobal = IEN[i,ilocal]
    for jlocal in range(0,4):
        jglobal = IEN[i,jlocal]

        K[iglobal,jglobal] += kelem[ilocal,jlocal]
        M[iglobal,jglobal] += melem[ilocal,jlocal]
```

As matrizes M e K são construídas da maneira acima. Utilizando a Matriz IEN, que contém os pontos que formam cada elemento da malha, é possível construir as matrizes elemento por elemento.

```
A = (M/dt + beta*K)
for i in cc2D:

    if X[i] == 0:
        T[i] = 0.0
        row = A.getrow(i)
        indices = row.indices

        for col in indices:
          # zero row
            A[i,col] = 0.0

        # set 1 in the diagonal(col)
        A[i,i] = 1.0

    if X[i] == 1:
```

```
        T[i] = 0.0
        row = A.getrow(i)
        indices = row.indices

        for col in indices:
          # zero row
              A[i,col] = 0.0

        # set 1 in the diagonal(col)
        A[i,i] = 1.0

    if Y[i] == 0:
        T[i] = 0.0
        row = A.getrow(i)
        indices = row.indices

        for col in indices:
          # zero row
              A[i,col] = 0.0

        # set 1 in the diagonal(col)
        A[i,i] = 1.0

    if Z[i] == 0:
        T[i] = 0.0
        row = A.getrow(i)
        indices = row.indices

        for col in indices:
          # zero row
              A[i,col] = 0.0

        # set 1 in the diagonal(col)
        A[i,i] = 1.0
```

```
    if Z[i] == 1:
        Q[i] = (-1.0)*h*T0
```

Novamente utilizamos a lista cc2D para poder definir os valores constantes dos contornos. Como a matriz A é muito grande e possui muitos pontos, é necessário utilizar a biblioteca SciPy para reduzir a quantidade de memória utilizado pelo computador para fazer cálculos com ela. Por isso os contornos são descritos de forma diferente em relação ao método analítico.

```
b = (M/dt)@T + M@Q

for i in cc2D:

    if X[i] == 0:
        b[i] = 0.0

    if X[i] == 1:
        b[i] = 0.0

    if Y[i] == 0:
        b[i] = 0.0

    if Z[i] == 0:
        b[i] = 0.0

    if Z[i] == 1:
        Q[i] = (-1)*h*T[i]

    T = spsolve(A.tocsc(),b)
```

Definindo todos os valores dos contornos novamente e definindo o valor do vetor b podemos encontrar o valor de T para a equação $AT = b$ onde T é a temperatura em $t > 0$.

Com os dois resultados é possível calcular os erros (absolutos e relativos) entre o método analítico e o método numérico. Considerando o resultado do método analítico como $[T_A]$ e o resultado do MEF como $[T_{MEF}]$ temos que a comparação se resume em:

$$[Err_{Absoluto}] = |[T_A] - [T_{MEF}]| \tag{4.25}$$

Na equação (4.29) $Err_{absoluto}$ representa o valor do Erro Absoluto (Gustavo dos Anjos [11]) entre o resultado Analítico ($[T_A]$) e o resultado numérico ($[T_{MEF}]$).

$$[Err_{Relativo}] = \left| \frac{Err_{Absoluto}}{[T_A]} \right| = \left| \frac{[T_A] - [T_{MEF}]}{[T_A]} \right| \tag{4.26}$$

Em (4.30) $Err_{Relativo}$ representa o erro relativo, ou percentual, do resultado do MEF em relação ao método Analítico. A simulação contém temperaturas que variam de 100°C até 0°C. Para evitar que os valores próximos de zero comprometam os cálculos do Erro Relativo as temperaturas serão convertidas para Kelvin, ou seja, a variação da temperatura da simulação vai de 373K até 273K.

Mesmo após a aplicação dos Erros das equações (4.29) e (4.30) o resultado ainda é um vetor de dimensão igual ao número de pontos da malha. Para auxiliar na avaliação dos valores dos erros foi necessário aplicar o conceito de norma nos vetores. Norma-1, Norma-2 e Norma-inf (Gustavo dos Anjos [11]).

$$\| x \|_1 = \Sigma_{i=1}^{n} |x_i| \tag{4.27}$$

$$\| x \|_2 = \left(\Sigma_{i=1}^{n} |x_i|^2 \right)^{\frac{1}{2}} \tag{4.28}$$

$$\| x \|_\infty = \mathbf{max} |x_i| \tag{4.29}$$

A malha da validação é um cubo de lado 1, como definido anteriormente. Depois de modelar a malha no programa Gmsh foi necessário definir um número de elementos para efetuar os cálculos e, depois de algumas tentativas, o maior número de elementos que o computador utilizado conseguiria gerar nos cálculos e utilizar nas contas em um tempo de processamento razoável (aproximadamente 45 minutos de processamento) é de 70.899 elemento tetraédricos e 12.393 pontos (nodes).

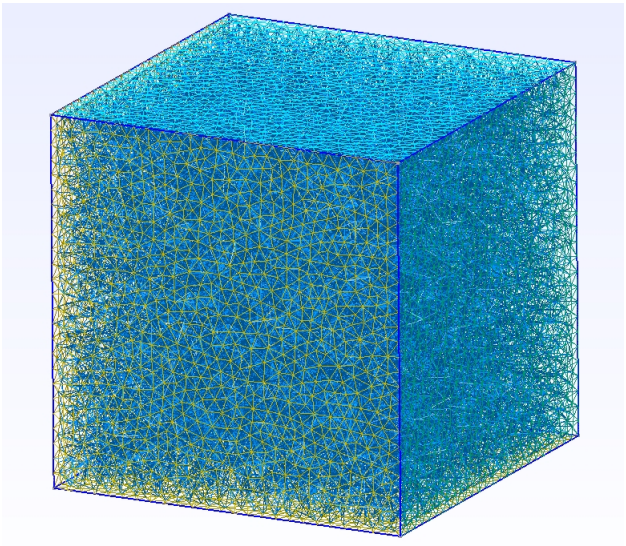


Figura 4.2: Malha cúbica com 70.899 elementos

4.3 Resultado da Validação

O programa em Python foi configurado para realizar a simulação em um intervalo de tempo de $\Delta t = 0.5s$ e um passo de tempo de $dt = 0.001s$. Isso resulta em 500 iterações das contas do programa.

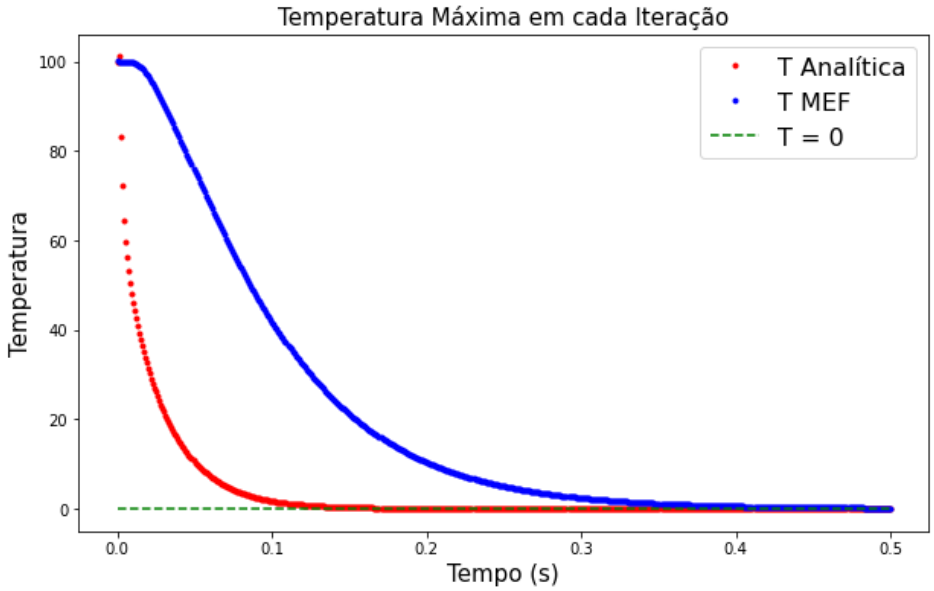


Figura 4.3: Temperatura máxima - Analítico vs MEF

O gráfico da temperatura, demonstrado na figura (4.3), demonstra o comportamento da temperatura máxia em ambos os modelos. Há uma discordância quantitativa entre os resultados pelo fato das curvas não serem iguais. Com o passar do

tempo vemos, porém, que as curvas passam a dispor de um comportamento simillar. Podemos então determinar que há uma similaridade que pode ser aferida a partir do erro calculado entre os modelos.

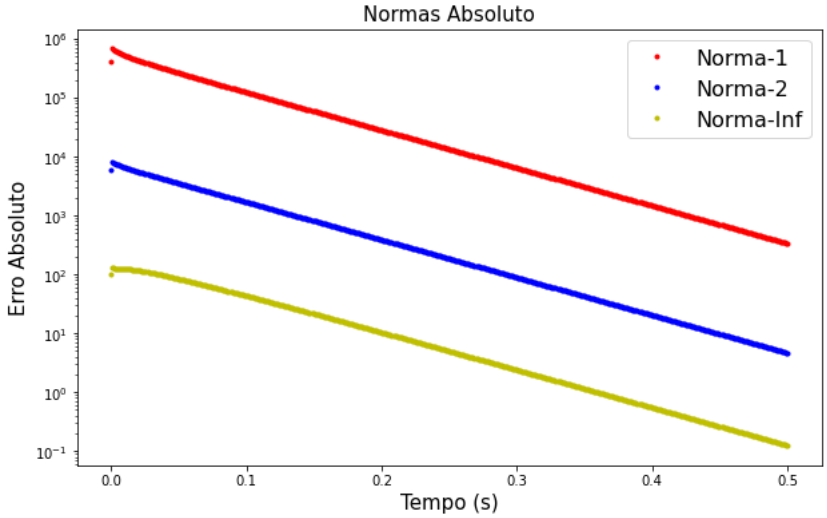


Figura 4.4: Normas dos Erro Absoluto - escala em Log

A figura (4.4) demonstra que o valor do Erro Absoluto reduz com o passar do tempo e isso já demonstra um comportamento desejável para que o método possa ser validado.

Na figura (4.5) é possível observar que o Erro Relativo também reduz com o passar do tempo, sendo necessário colocar o Eixo Y do gráfico em escala logarítmica para poder visualizar a redução por inteiro.

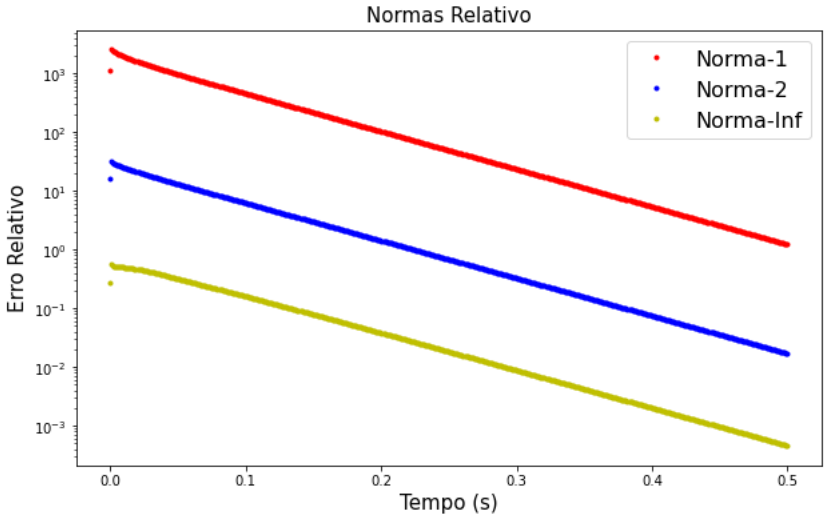


Figura 4.5: Normas do Erro Relativo - escala em Log

Outra possível maneira de observar o comportamento do Erro Relativo durante a simulação é calcular a média desses valores em cada iteração.

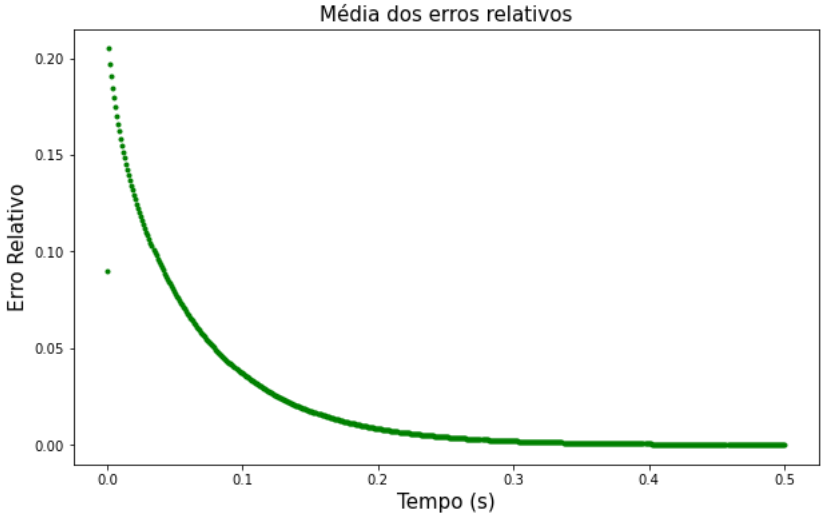


Figura 4.6: Erro Relativo médio

Na figura (4.6) é possível observar que a média do erro relativo entre os métodos reduz com uma grande rapidez e tem valor máximo em torno de 20%. Para melhor observar o comportamento total da média do erro relativo na simulação é necessário aplicar a escala logarítmica.

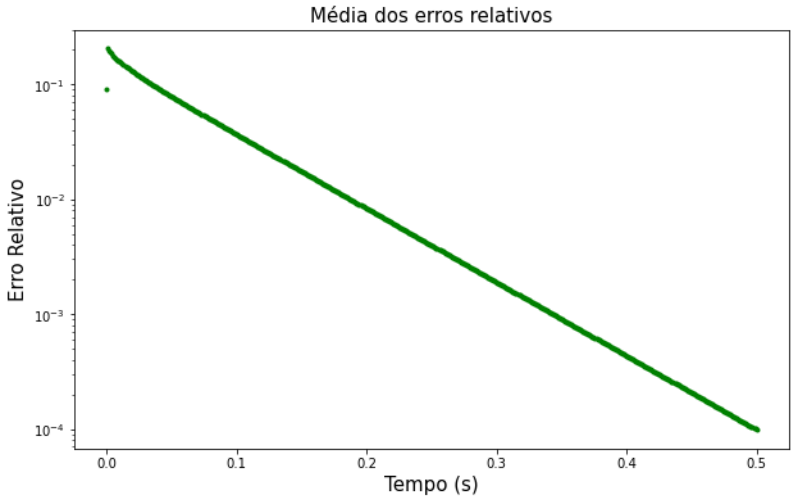


Figura 4.7: Erro Relativo Médio - Escala em Log

Com a nova escala podemos observar que o erro médio reduz até a ordem de 10^{-4}.

Capítulo 5

Resultados

A malha utilizada na simulação do MEF é uma aproximação da geometria de um processador. Utilizando dados do tamanho de cada componente extraídos do vídeo em [9], foi possível construir a malha no Software Gmsh. As características termodinâmicas de cada parte do processador (PCB, Die, IHS e TIM) foram aplicadas dentro do programa em Python que implementa as propriedades a cada ponto em função do seu posicionamento na malha. Há também a presença de ar atmosférico dentro do processador, na região entre o Die e o IHS. Suas propriedades termodinâmicas também foram consideradas. A geometria geral da malha do Processador se resume a 2 paralelepípedos sobrepostos. O computador utilizado na simulação contém um processador Intel(R) Core(TM) i5-9400F 2.90GHz.

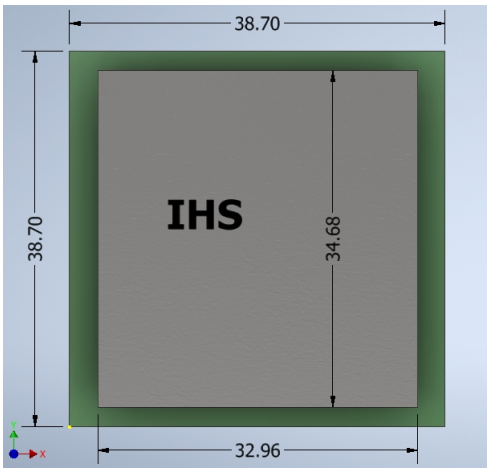


Figura 5.1: Cotas vista superior - IHS e PCB

O paralelepípedo inferior contém o PCB em sua maioria e parte do Die e o paralelepípedo superior consiste no IHS, TIM, Ar e o resto do Die. Na figura (5.1)

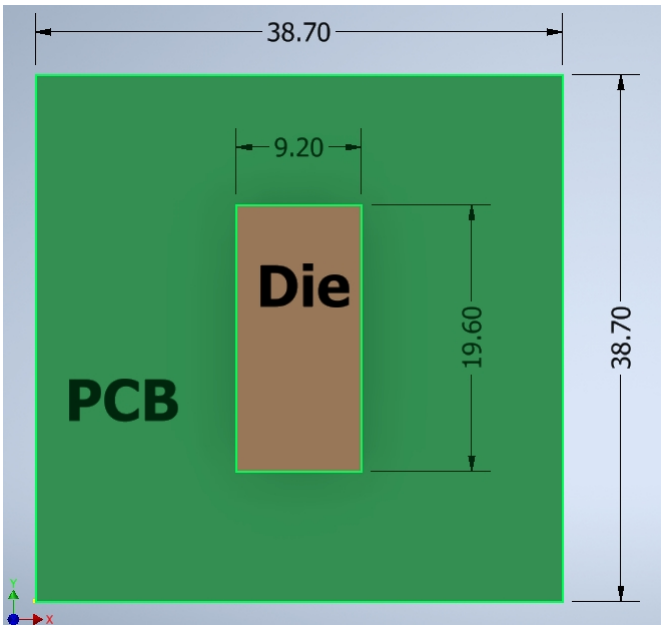


Figura 5.2: Vista superior sem o IHS - PCB e Die

é possível observar o IHS e o PCB. As figuras (5.1), (5.2), (5.3) e (5.4) demonstram a maneira que o processador é montado e as suas dimensões.

A fonte de calor é aplicada na região que se localiza o Die, onde a maior parte da corrente elétrica percorre no processador. Nesta simulação uma fonte de 65W está sendo convertida, integralmente, em calor. Para resfriar o processador foi considerado uma corrente de ar em temperatura ambiente ($T_{inf} = 25°C$) realizando uma troca de calor por convecção com o processador.

A simulação está sendo executada nas seguintes condições:

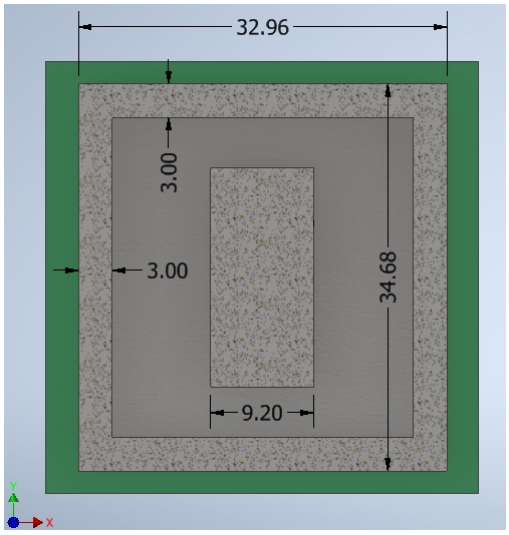


Figura 5.3: Vista superior com corte

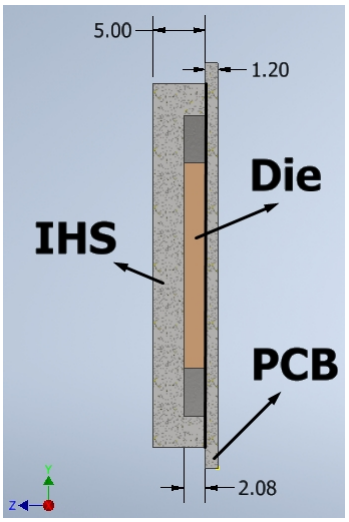


Figura 5.4: Vista lateral com Recorte - PCB, IHS e Die

- Temperatura inicial do processador é de $T_0 = 25°C$;

- Temperatura do Ambiente é de $T_{inf} = 25°C$;

- O Die Produz 65W de calor;

- A área superior do IHS perde calor para o ambiente por meio de Convecção Forçada ($h = 300[W/m^2K]$);

- A simulação foi feita com 8 Cores ativos;

- $dt = 0,001s$;

- $\Delta t = 0,5s$;

- Na simulação com pasta térmica suas propriedades termodinâmicas são aplicada na área superior do IHS, onde ocorre a convecção;

- A simulação pretende solucionar a equação (3.4) através do MEF, ou seja, solucionar a equação (3.19).

As propriedades termodinâmicas dos materias estão descritas na tabela abaixo.

As temperaturas representadas nos gráficos a seguir são aproximações feitas por uma média aritmética da temperatura e o número de pontos.

5.1 Estudo de Malha

As seguintes simulações serão feitas em malhas com um número crescente de elementos e pontos. A primeira malha possui 144 elementos e 26 pontos, a segunda possui 1470 elementos e 280 pontos, a terceira possui 14.334 elementos e 2703 e a

Propriedades Termodinâmicas dos componentes			
Componente	Massa específica [kg/m^3]	Calor específico [J/kg $^\circ C$]	Condutividade térmica [J/m s $^\circ C$]
Die (Silica)	2650	712	1,25
PCB (FR-4)	1850	1100	(x,y)0,81 (z)0,29
IHS (Cobre)	8960	380	401
Pasta Térmica	4050	1,0	8,9
Ar	1,225	1000	0,03

Tabela 5.1: Propriedades termodinâmicas dos matériais do processador [12] [13] [14]

última possui 70.507 e 12.758. A fonte de calor no Die é uniforme.

5.1.1 Malha com 144 elementos

Essa malha possui o menor número de elementos que o programa Gmsh consegue gerar na geometria do processador.

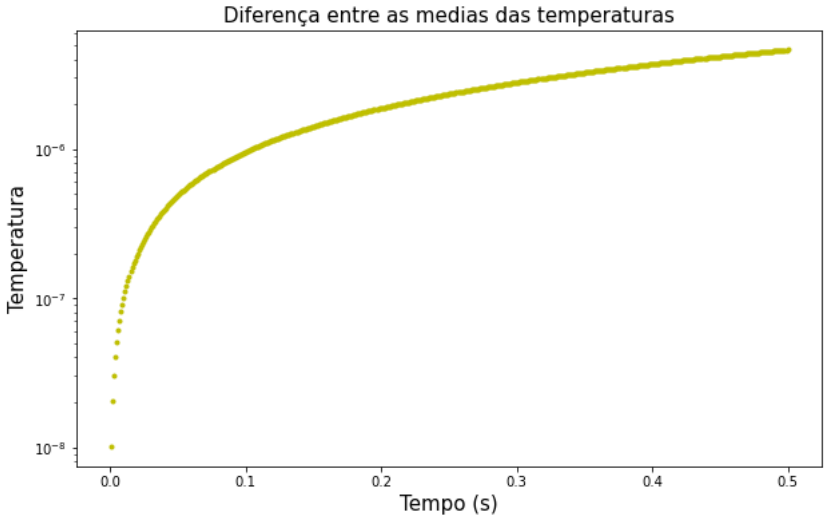


Figura 5.5: Diferença entre as Temperaturas Médias - 144 elementos

A figura (5.5) demonstra que as diferenças entre a temperatura média das simulações com pasta térmica e sem pasta térmica é muito pequena porém crescente. Não é um resultado muito aplicável pois o número de pontos na malha é muito

pequeno para haver um discernimento entre as diferentes regiões.

5.1.2 Malha com 1.470 elementos

Essa malha possui, aproximadamente, 10x mais elementos do que a malha anterior. Esse aumento na ordem de grandeza é feito para melhor observar as mudanças presentes quando a malha possui um maior refinamento.

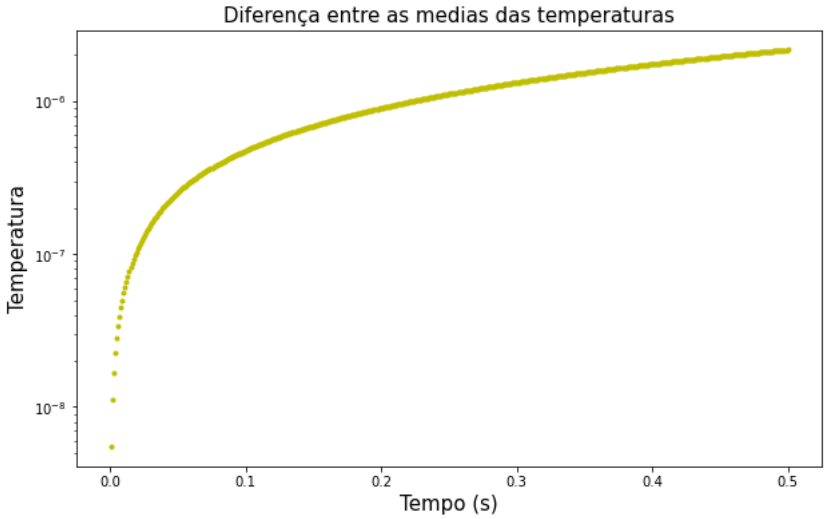


Figura 5.6: Diferença entre as Temperaturas Médias - 1470 elementos

Novamente o resultado ainda não é bom o suficiente. A diferença entre as simulações não apresenta uma magnitude relevante na aplicação. Isso tudo é perceptível na figura (5.6).

5.1.3 Malha com 14.334 elementos

O próximo passo é aumentar o número de elementos em mais uma ordem de grandeza. Essa malha também exibe um aumento de 10x, aproximadamente, em relação à malha anterior.

Nessa simulação as temperaturas médias das duas versões começam a apresentar diferenças mais perceptíveis, como é possível destacar as figuras (5.7) e (5.8). Essa malha começa a demonstrar o comportamento esperado das simulações. A simulação com a pasta térmica apresenta uma temperatura média maior do que a simulação sem pasta térmica. A presença da pasta térmica melhora a transferência de calor no processador.

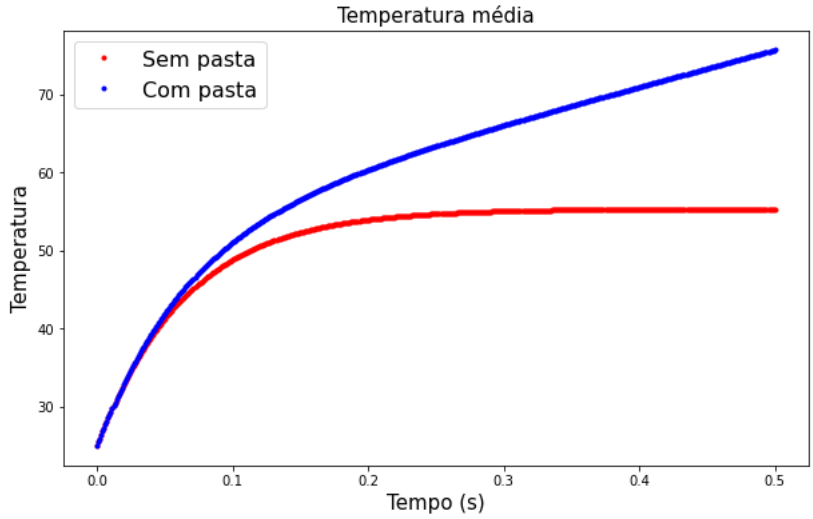


Figura 5.7: Temperatura Média - 14.334 elementos

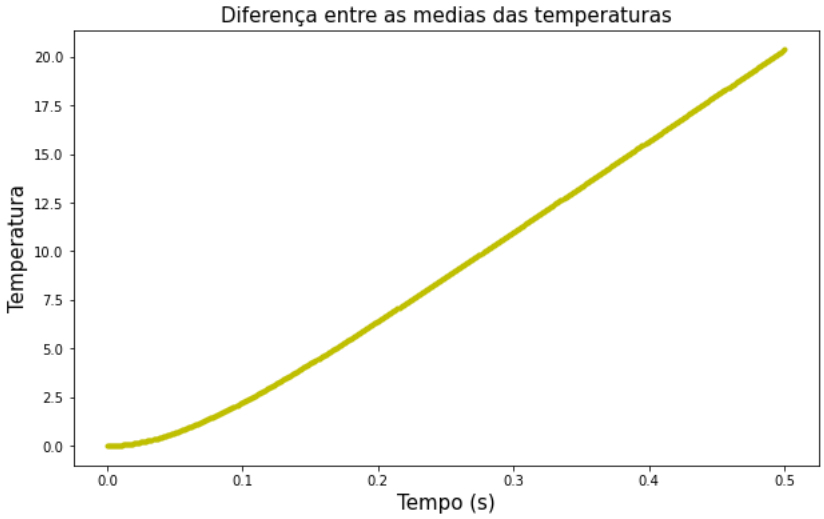


Figura 5.8: Diferença entre as Temperaturas Médias - 14.334 elementos

5.1.4 Malha com 70.507 elementos

O número de elementos agora é 5x maior, aproximadamente, que o número de elementos na malha anterior. Essa simulação despendeu 40 minutos.

Comparando as figuras (5.9) e (5.7) é possível perceber que os resultados numéricos, apesar de diferentes, apresentam um comportamento similar. A temperatura média no processador com pasta térmica é maior do que a temperatura média no processador sem pasta térmica. Essa diferença cresce com o passar do tempo, como visto na figura (5.10). A discrepância nos resultados provém do aumento da qualidade da malha, em outras palavras, confirma que o refinamento da

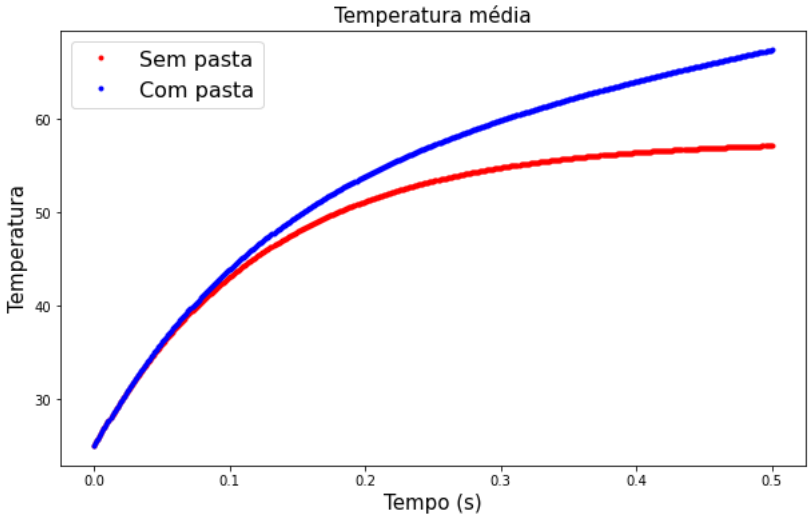


Figura 5.9: Temperatura Média - 70.507 elementos

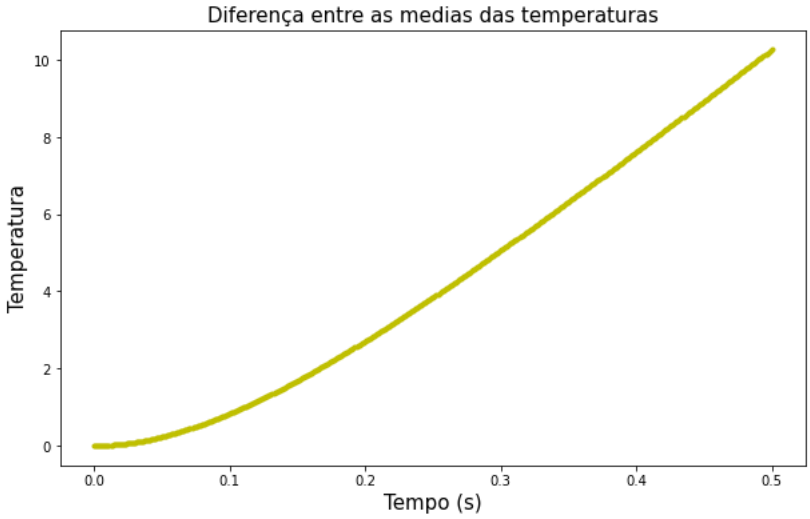


Figura 5.10: Diferença entre as Temperaturas Médias - 70.507 elementos

malha causa uma melhora no resultado final do método numérico por se aproximar cada vez mais do "contínuo", que simboliza a situação real. Essa malha é a maior malha que o computador utilizado poderia calcular em um intervalo de tempo aplicável.

5.2 Simulação de Die com potência térmica não-uniforme

O Die é subdividido em 10 partes: 8 núcleos, GPU e Sistema. Para uma melhor representação de diferentes situações do processador foi feita a repartição do processador nessas partes, como é possível ver na figura (5.11), e a geração de calor foi repartida no percentual de consumo energético de cada parte. As outras regiões descritas na figura (2.4) foram aglomeradas nas seções maiores mais próximas por serem um percentual menor da área total do Die. A geração de calor dos 8 núcleos em conjunto é equivalente a 70% do calor geral. O Sistema equivale a 20% e a GPU representa os 10% restantes. Logo, cada núcleo emite $5,7W$, o sistema emite $13W$ e a GPU emite $6,5W$. As próximas simulações também vão comparar temperaturas com e sem a pasta térmica.

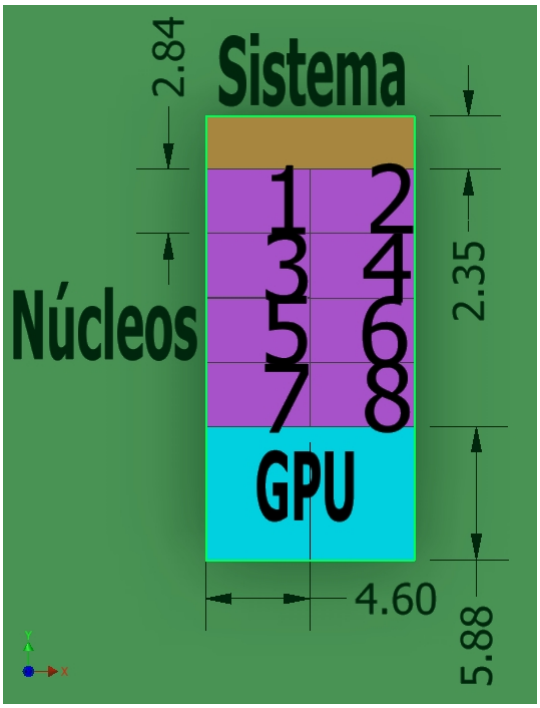


Figura 5.11: Die separado em Nucleos - mm

5.2.1 1 Núcleo

A primeira simulação foi feita com a região do Sistema, a GPU e o núcleo 1 ativos assim como na figura (5.12). A potência total é de $25,2W$.

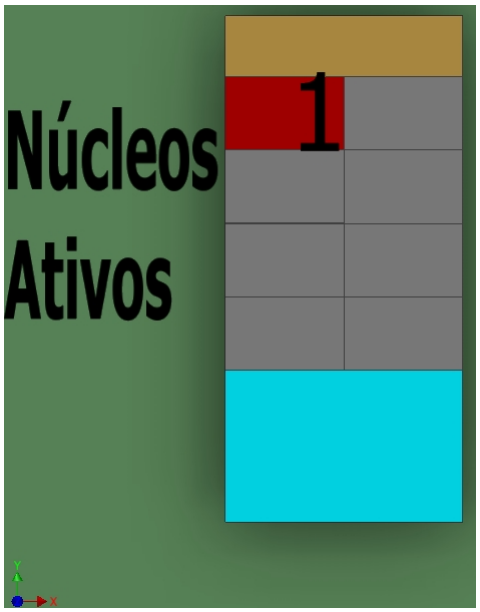


Figura 5.12: Representação das regiões do Die ativas - 1 Núcleo

A temperatura, nessa simulação, apresentou uma diferença pequena e crescente que pode ser observável na figura (5.14). Isso provém da fonte de calor ser pequena e pelo Die não estar 100% ativo. Mesmo assim, a característica da temperatura média com pasta ser maior que a temperatura média sem pasta ainda se mantém como é possível observar na figura (5.13).

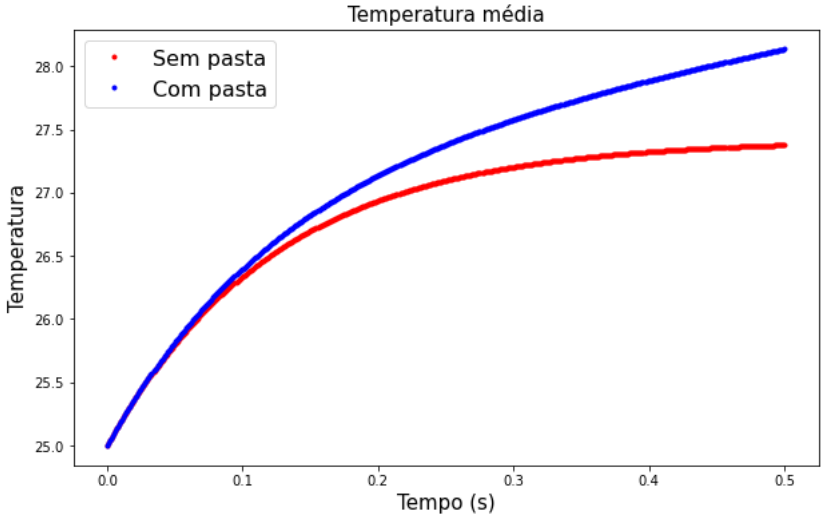


Figura 5.13: Temperatura Média - 1 Núcleo

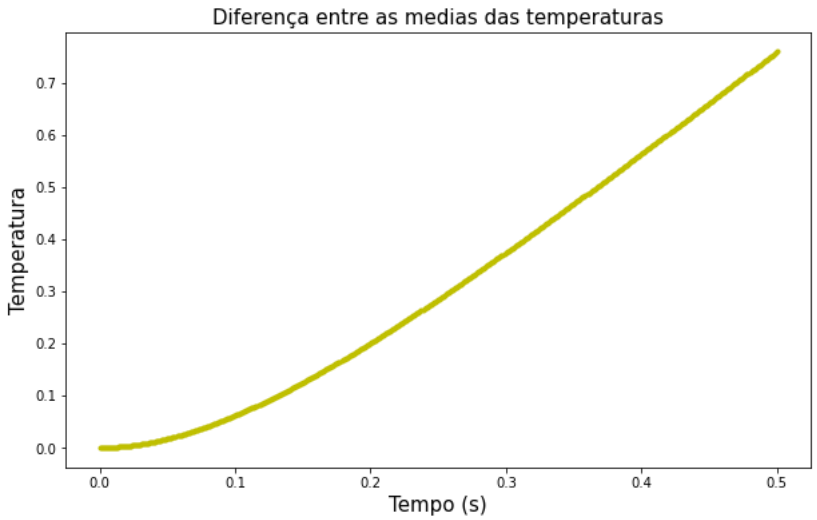


Figura 5.14: Diferença entre as Temperaturas Médias - 1 Núcleo

5.2.2 2 Núcleos

A próxima simulação foi feita com a região do Sistema, a GPU, o núcleo 1 e o núcleo 2 ativos como descrito na figura (5.15). A potência total é de $30,9W$.

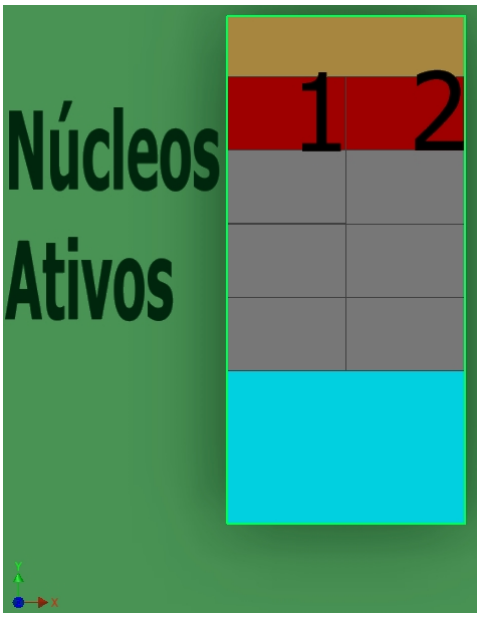


Figura 5.15: Representação das regiões do Die ativas - 2 Núcleos

Em relação a temperatura da simulação anterior houve um aumento geral. Isso é decorrido da presença de uma fonte a mais de calor (um núcleo extra). Na figura (5.17) é possível observar que, comparando com a figura (5.14), a diferença entre as médias das temperaturas está aumentando. Temperaturas médias observáveis na figura (5.16). Essas médias apresentam uma temperatura média maior que a

temperatura com 1 núcleo ativo. Mesmo com a diferença de fontes térmicas, ainda é possível observar que a temperatura da simulação com pasta térmica é maior que a temperatura média da simulação sem pasta térmica.

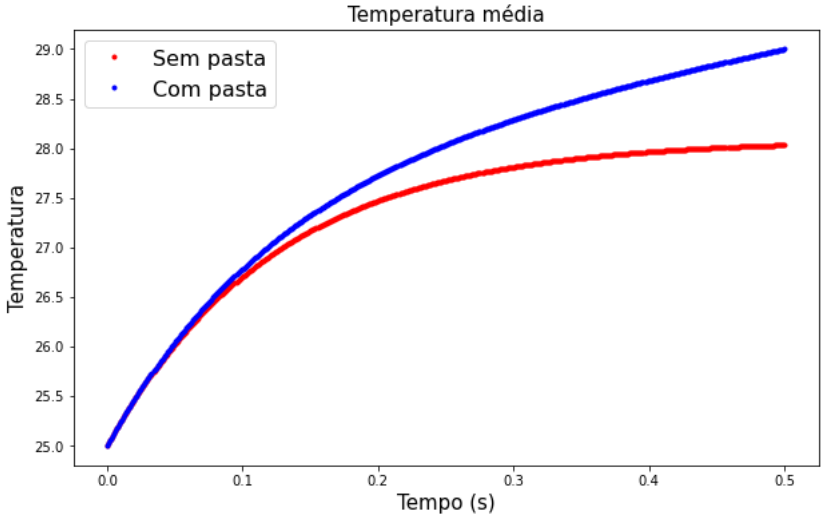


Figura 5.16: Temperatura Média - 2 Núcleos

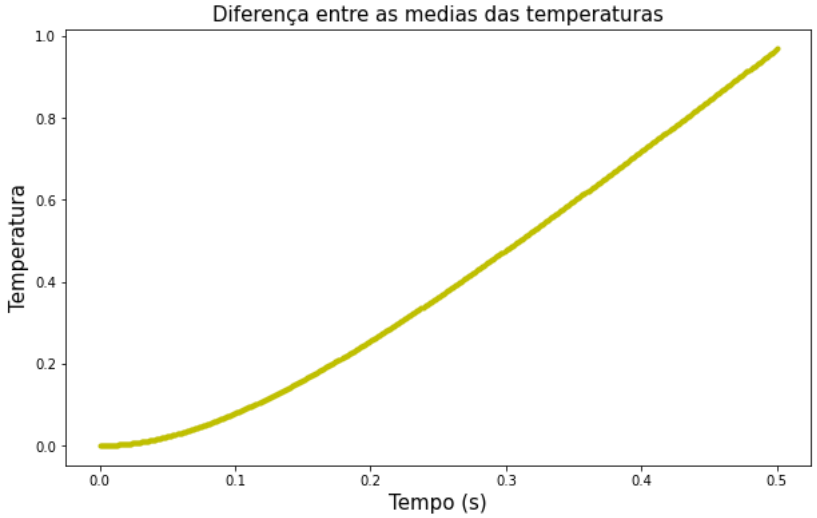


Figura 5.17: Diferença entre as Temperaturas Médias - 2 Núcleos

5.2.3 4 Núcleos

Além das regiões anteriores estarem ativas, agora os núcleos 3 e 4, descritos na figura (5.18), também estarão gerando calor nesta próxima simulação. A potência total é de $42,3W$.

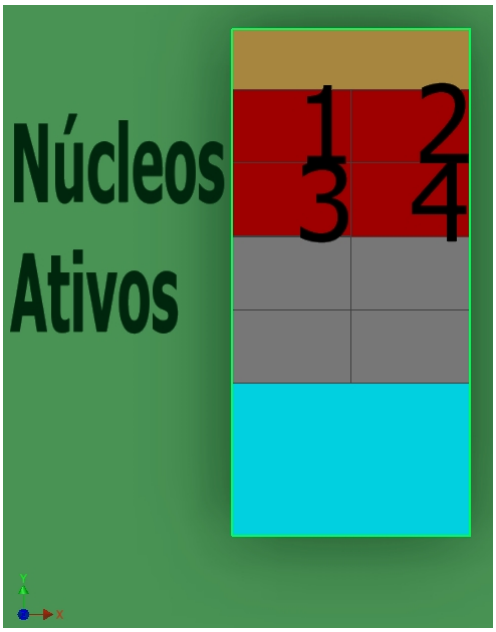


Figura 5.18: Representação das regiões do Die ativas - 4 Núcleos

Novamente há um salto de temperatura na simulação pela presença de mais fontes de calor. As temperaturas médias apresentadas no gráfico da figura (5.19), comparadas as temperaturas médias das figuras (5.16) e (5.13), são maiores. Juntamente com isso, a diferença das temperaturas médias entre as simulações com pasta térmica e sem pasta térmica também demonstrou um crescimento na figura (5.20). Novamente pode-se observar que a temperatura média com pasta térmica é maior que a temperatura média sem pasta térmica.

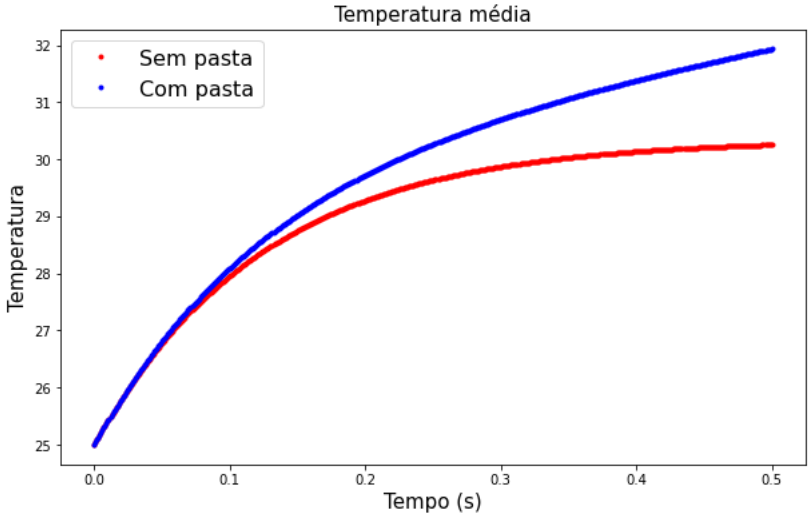


Figura 5.19: Temperatura Média - 4 Núcleos

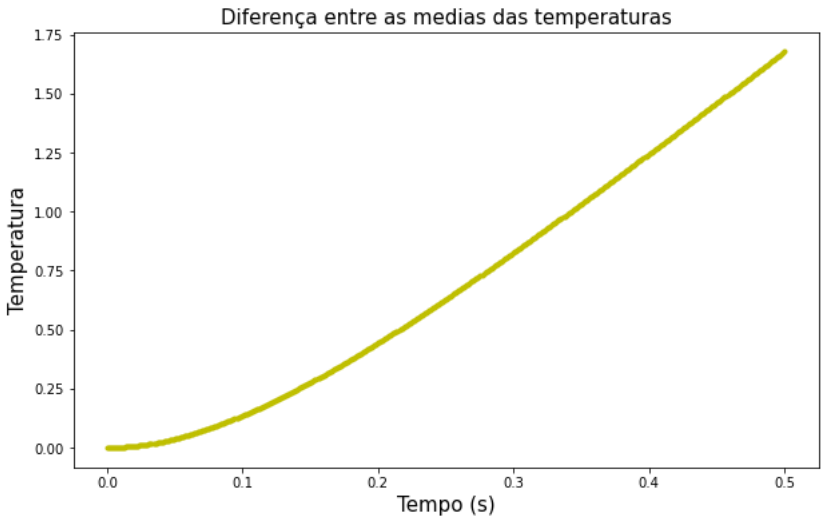


Figura 5.20: Diferença entre as Temperaturas Médias - 4 Núcleos

5.2.4 6 Núcleos

Nesta simulação os Núcleos 1, 2, 3, 4, 5 e 6 estarão ativos além das regiões da GPU e do Sistema. A potência total é de $53,7W$.

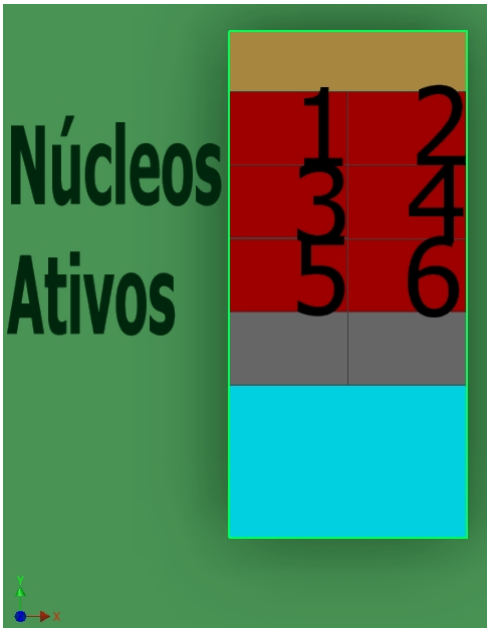


Figura 5.21: Representação das regiões do Die ativas - 6 Núcleos

Novamente é possível observar que a temperatura média com pasta mantém a sua característica de ser maior que a temperatura média sem pasta demonstrável na figura (5.22). A diferença entre as temperaturas médias continua a crescer comparando a figura (5.23) com as figuras (5.20), (5.17) e (5.14).

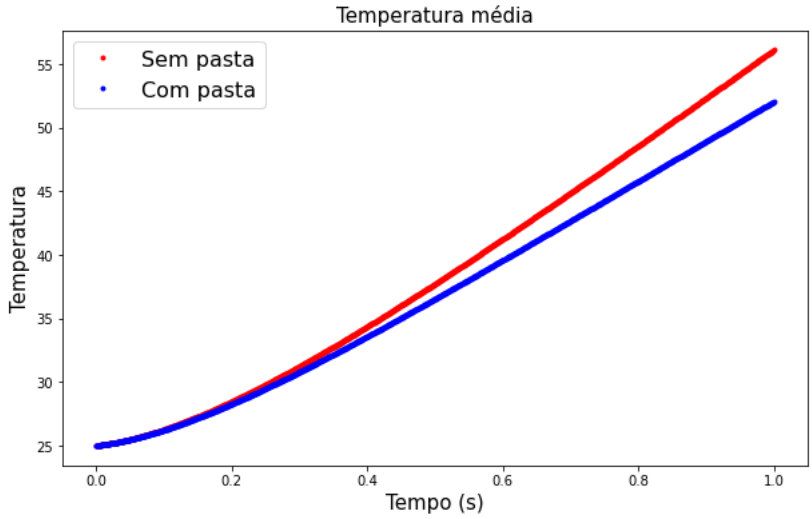


Figura 5.22: Temperatura Média - 6 Núcleos

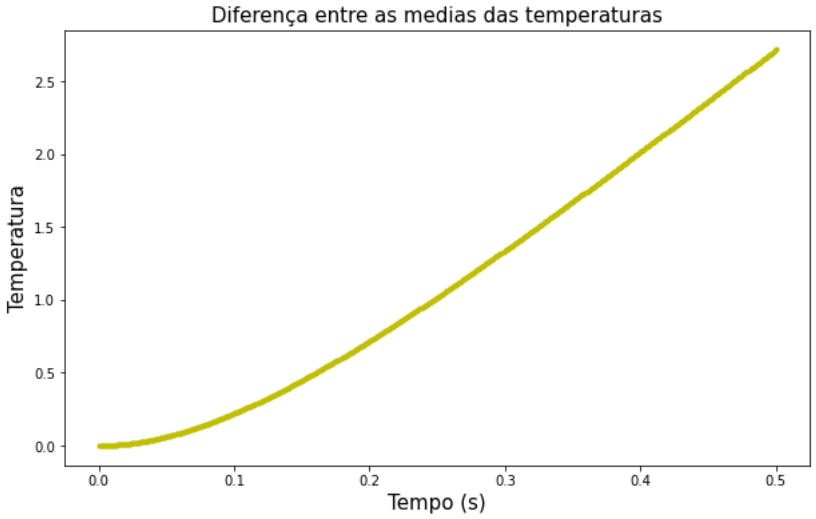


Figura 5.23: Diferença entre as Temperaturas Médias - 6 Núcleos

5.2.5 8 Núcleos

A próxima simulação seria a representação do Die completamente ativo, como descrito na figura (5.24). Isso se aproxima mais das simulações que comparavam as malhas, porém em uma situação mais realista com uma divisão não homogênea da fonte de calor.

As figuras (5.25) e (5.26) representam as temperaturas médias e as diferenças entre as temperaturas médias, respectivamente. Novamente o comportamento da temperatura média com a pasta térmica se mantém acima da temperatura média sem pasta térmica. A diferença entre as temperaturas também se mantém crescendo

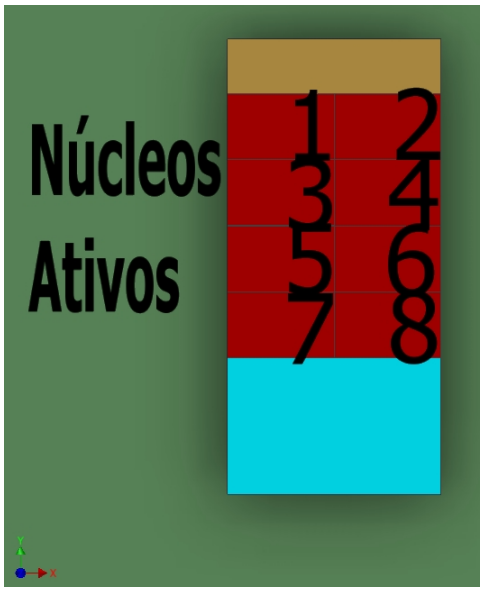


Figura 5.24: Representação das regiões do Die ativas - 8 Núcleos

conforme a fonte de calor aumenta. Como a distribuição da temperatura não é homogênea, como nas simulações da seção (5.1), há uma diferença entre as médias das temperaturas e, por consequência, entre as diferenças das temperaturas médias.

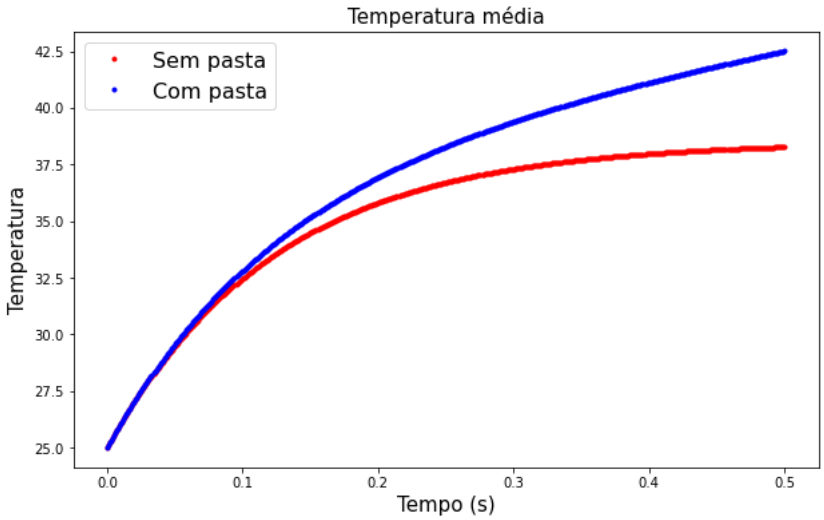


Figura 5.25: Temperatura Média - 8 Núcleos

5.2.6 Visualização das fontes de calor

As próximas imagens demonstram a distribuição térmica na malha das simulações com variação de núcleos. A visualização é feita pelo Paraview e é possível observar os diferentes formatos das fontes térmicas.

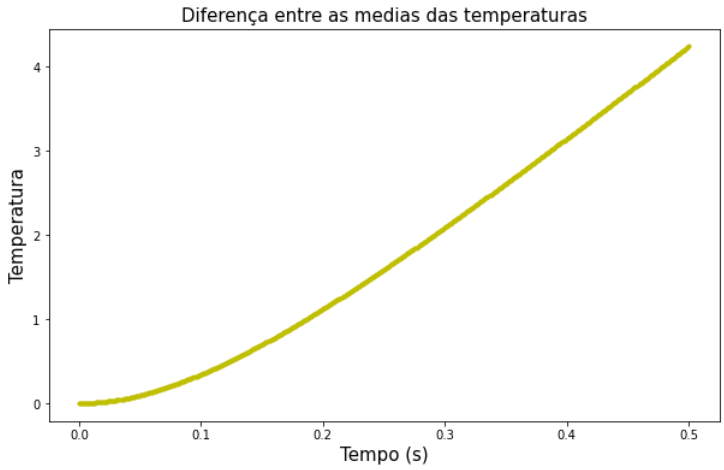
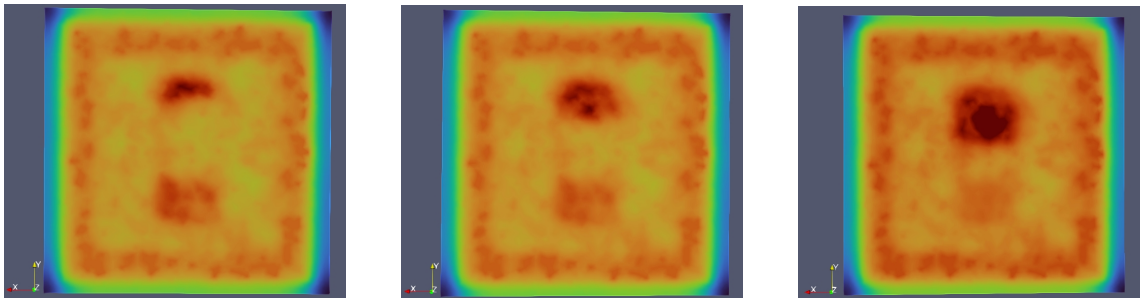
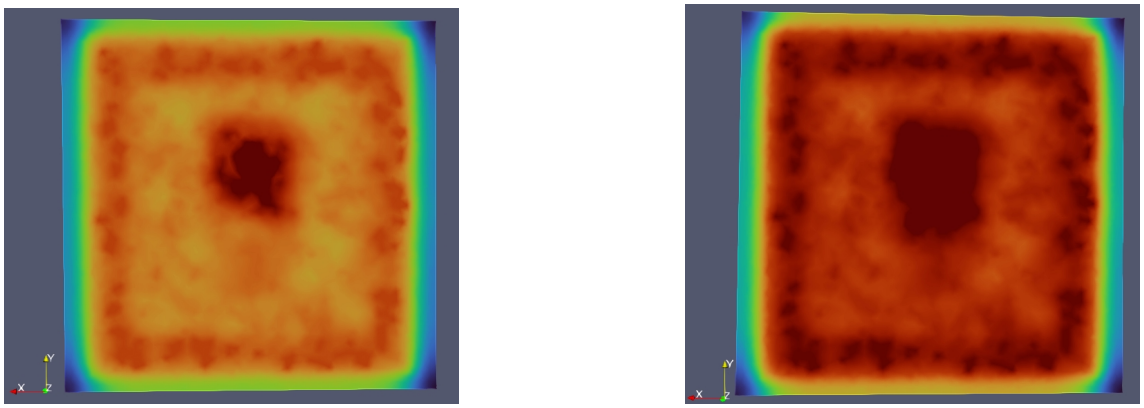


Figura 5.26: Diferença entre as Temperaturas Médias - 8 Núcleos



(a) 1 Núcleo (b) 2 Núcleos (c) 4 Núcleos

Figura 5.27: Visualização das diferentes potências térmicas - 1, 2 e 4 Núcleos



(a) 6 Núcleos (b) 8 Núcleos

Figura 5.28: Visualização das diferentes potências térmicas - 6 e 8 Núcleos

As imagens das figuras (5.27) e (5.28) demonstram como a fonte de calor é observável nas diferentes simulações. Conforme o número de Núcleos aumenta, a concentração das maiores temperaturas na região do Die aumenta.

Capítulo 6

Conclusões

Neste trabalho foi apresentada a aplicação do Método de Elementos Finitos no cálculo da Equação de Transferência de Calor Transiente em um Processador em funcionamento. Um dos objetivos do trabalho era averiguar se o programa seria aplicável na situação em questão. Os resultados obtidos através do MEF se distanciam dos resultados analíticos, como observado no capítulo 4. Com isso em mente é necessário progredir com cuidado ao analisar os resultados da aplicação do MEF, tendo como base o percentual de erro obtido na comparação com os resultados analíticos.

Outro ponto citado no início do trabalho era relacionado ao custo de aplicação do programa. Todos os softwares utilizados no trabalho, tanto o Paraview quanto o Gmsh, são softwares gratuitos disponíveis na Internet. A linguagem Python, além de ser uma linguagem de fácil acesso, possui diversas comunidades de apoio prontas para ajudar caso haja alguma dificuldade. Logo, o trabalho é aplicável com baixos custos, necessitando somente de um computador além de possuir amplo apoio em sua execução, sendo facilmente adaptável a condição do usuário.

Os resultados gerados foram satisfatórios quanto a sua aplicação. Há espaço para um estudo mais aprofundado neste assunto, tanto para o relacionamento de diferentes pastas térmicas, quanto para a comparação de diferentes fontes de refrigeração para um mesmo processador. Comparação de diferentes processadores também é uma opção válida para se trabalhar utilizando o MEF, mais especificamente os programas disponíveis nos Anexos. Outro ponto seria aprofundar o estudo sobre a melhora dos resultados do MEF em relação a solução analítica utilizando malhas

mais refinadas e um número maior de pontos na comparação com a finalidade de aproximar os dois resultados.

Outro aspecto explorado no trabalho foi a comparação da malha do processador representando diferentes núcleos (ou *cores*) que se faz pertinente pela presença cada vez maior de processadores com muitos núcleos. Hoje em dia há relatos de processadores sendo desenvolvidos com até 64 núcleos. Isso pode ser utilizado para ajudar a diagnosticar possíveis problemas na performance de cada núcleo.

O MEF possui uma grande vantagem quando se considera a geometria do problema. É possível aplicar o MEF em geometrias muito complexas sem haver perda de desempenho. Dependendo somente das equações pertinentes e suas condições de contorno. Se bem aplicado os resultados não serão menos confiáveis.

O grande limitador na aplicação do MEF é a capacidade computacional do computador utilizado. O computador utilizado para fazer esse trabalho foi capaz de fazer cálculos de temperaturas em malhas com 70.000 elementos, aproximadamente. Foi feita a tentativa de calcular em uma malha mais refinada, porém o tempo de processamento era muito grande e impraticável dentro do objetivo de conclusão deste trabalho. Com a disponibilidade de um computador com mais memória e um processador mais rápido é possível refinar mais ainda as malhas, sendo assim viável determinar um número otimizado de elementos que se aproxime do resultado real em um tempo de processamento aceitável.

No final é possível discernir que o MEF é uma ferramenta muito potente, de fácil acesso e fácil aplicação. A proximidade do resultado com a realidade depende muito do poder computacional do computador utilizado, mas, mesmo assim, o método ainda se mostra muito útil na transferência de calor e em outros assuntos de engenharia que possuam equações diferenciais.

Referências Bibliográficas

[1] ÖZISIK, M. N., *Transfêrencia de Calor, Um Texto Básico*. Editora Guanabara Koogan S.A., 1985.

[2] INCROPERA, F. P., *Fundamento de Transferência de Calor e de Massa th ed.*. LTC – Livros Técnicos e Científicos Editora Ltda., 2014.

[3] LEWIS, R. W., *Fundamentals of the Finite Element Method for Heat and Fluid Flow*. John Wiley Sons, Ltd., 2004.

[4] JUSTO, D. A. R., *Cálculo Numérico, Um Livro Colaborativo*. UFRGS, 2020.

[5] RIBEIRO, D., "https://www.dicio.com.br/processador/", 10.06.2022.

[6] SCHUH, M. O., "CARACTERIZAÇÃO E ANÁLISE TÉRMICA DE UM MICROPROCESSADOR DE ALTA, PERFORMANCE VIA MÉTODO NUMÉRICO", *https://lume.ufrgs.br/handle/10183/173726*, 2017.

[7] INTEL, "Technical Resources: Intel Core Processors", *https://www.intel.com.br/content/www/br/pt/products/docs/processors/core/core-technical-resources.html*, 09.06.2022.

[8] SILVER, A., "Arctic Silver 5, High-Density Polysynthetic Silver Thermal Compound", *https://www.arcticsilver.com/as5.htm*, 07.06.2022.

[9] DER8AUER, "Intel i9-10900K Delidding - Temperatures and Die-Analysis. Details vs. 8700K and 9900K", *https://www.youtube.com/watch?v=2b83EJu7uqQab*$_c$*hannel* = *der*8*auer*, 14.06.2022.

[10] OZISIK, M. N., HAHN, D. W., *Heat Conduction - Third edition*. John Wiley Sons, Inc., 2012.

[11] DOS ANJOS, G. R., *Projeto Final, Mecânica dos Flúidos e Transferência de calor Computacional*. UFRJ-DEM, https://gustavorabello.github.io/teaching/.

[12] ZM PETERSON, N., "FR4 Thermal Properties to Consider During Design", *https://www.nwengineeringllc.com/article/fr4-thermal-properties-to-consider-during-design.php*, 22.06.2022.

[13] WENHUI ZHU, GUANG ZHENG, S. C. . H. H., "Thermal conductivity of amorphous SiO2 thin film: A molecular dynamics study", *https://www.nature.com/articles/s41598-018-28925-6: :text=obtained20.06.2022.*

[14] HALLIDAY, RESNICK, WALKER, Fundamentos de física. 8ª edição, vol. 3. *LTC, 2009.*

[15] DOS ANJOS, G. R., Computação Científica para Engenheiros. *UFRJ-COPPE, https://gustavorabello.github.io/teaching/.*

[16] INTEL, "Making of a Chip Illustrations", https://download.intel.com/newsroom/kits/chipmaking/pdfs/Sand-to-Silicon$_3$2$nm - Version.pdf$, 06.06.2022.

[17] EYMARD, R., THIERRY, G., HERBIN, R., "Handbook of Numerical Analysis", v. 7, pp. 731–1018, 01 2000.

Apêndice A

Código Fonte: Validação

```
from datetime import datetime

now = datetime.now()

current_time = now.strftime("%H:%M:%S")
print("Current Time =", current_time)
import meshio
import numpy as np
import matplotlib.pyplot as plt
import scipy as sp
from scipy.sparse import lil_matrix,csr_matrix
from scipy.sparse import csc_matrix,coo_matrix
from scipy.sparse import dok_matrix
from scipy.sparse.linalg import spsolve

msh= meshio.read('malha_de_validao.msh')
X = msh.points[:,0]
Y = msh.points[:,1]
Z = msh.points[:,2]
IEN = msh.cells[2].data
IENbound2D = msh.cells[1].data
npoints = len(X)
ne = IEN.shape[0]
```

```
cc2D = np.unique(IENbound2D.reshape(IENbound2D.size))


mediaTN=[]
mediaTA=[]
tempAnalitico = []
tempMEF = []
norma1Abs = []
norma2Abs = []
normainfAbs = []
norma1Rel = []
norma2Rel = []
normainfRel = []
media=[]

q = 0.0 #Fonte de calor [W/m^3]
rho = 1.0 #Densidade do material [Kg/m^3]
cv = 1.0  #Capacidade termica [J/Kg oC] ou [J/Kg K]
k = 1.0   #Condutividade termica [J/(m s oC)] ou [J/(m s K)]
alpha = k/(rho * cv) #difusividade termica [m^2/s]
dt = 0.001 # Fraao  de tempo
nIter = 500 #Numero de  interaes
beta = 1.0 #Metodo no tempo para d^2/dx^2
T0 = 100.0
h = 1.0 #Coeficiente de tranferencia de calor por conveco


z1 = [2.0288, 4.9132, 7.9787, 11.0856, 14.2075, 17.3364] #razes de
    x*cotg(x)=-1

TAnalitic = np.zeros(npoints, dtype='float')

TNumeric = np.ones(npoints, dtype='float')
```

```
ErrABS = np.zeros((npoints),dtype='float')
ErrREL = np.zeros((npoints),dtype='float')

K = lil_matrix((npoints,npoints), dtype='float32')
M = lil_matrix((npoints,npoints), dtype='float32')

#------------------------Loop -
    Assembling-------------------------------------

for i in range(0, ne):
    v1 = IEN[i,0]
    v2 = IEN[i,1]
    v3 = IEN[i,2]
    v4 = IEN[i,3]

#------------------------Coeficientes b, c, d
    ----------------------------------

    b1 = ((Y[v2] - Y[v4]) * (Z[v3] - Z[v4])) - ((Y[v3] - Y[v4]) * (Z[v2]
        - Z[v4]))
    b2 = ((Y[v3] - Y[v4]) * (Z[v1] - Z[v4])) - ((Y[v1] - Y[v4]) * (Z[v3]
        - Z[v4]))
    b3 = ((Y[v1] - Y[v4]) * (Z[v2] - Z[v4])) - ((Y[v2] - Y[v4]) * (Z[v1]
        - Z[v4]))
    b4 = (b1 + b2 + b3)*(-1.0)

    c1 = ((X[v3] - X[v4]) * (Z[v2] - Z[v4])) - ((X[v2] - X[v4]) * (Z[v3]
        - Z[v4]))
    c2 = ((X[v1] - X[v4]) * (Z[v3] - Z[v4])) - ((X[v3] - X[v4]) * (Z[v1]
        - Z[v4]))
    c3 = ((X[v2] - X[v4]) * (Z[v1] - Z[v4])) - ((X[v1] - X[v4]) * (Z[v2]
        - Z[v4]))
    c4 = (c1 + c2 + c3)*(-1.0)
```

```
    d1 = ((X[v2] - X[v4]) * (Y[v3] - Y[v4])) - ((X[v3] - X[v4]) * (Y[v2]
        - Y[v4]))
    d2 = ((X[v3] - X[v4]) * (Y[v1] - Y[v4])) - ((X[v1] - X[v4]) * (Y[v3]
        - Y[v4]))
    d3 = ((X[v1] - X[v4]) * (Y[v2] - Y[v4])) - ((X[v2] - X[v4]) * (Y[v1]
        - Y[v4]))
    d4 = (d1 + d2 + d3)*(-1.0)

    volume = (1.0/6.0) * np.linalg.det([[1,X[v1],Y[v1],Z[v1]],
                                        [1,X[v2],Y[v2],Z[v2]],
                                        [1,X[v3],Y[v3],Z[v3]],
                                        [1,X[v4],Y[v4],Z[v4]]])

    melem = (volume/20.0) * np.array([[2,1,1,1],
                                      [1,2,1,1],
                                      [1,1,2,1],
                                      [1,1,1,2]])

    B = (1.0/(volume * 6.0)) * np.array([[b1,b2,b3,b4],
                                         [c1,c2,c3,c4],
                                         [d1,d2,d3,d4]])

#----------------Construcao das matrizes K e
    M---------------------------------

    kelem = alpha*volume*(np.transpose(B)@B)

    for ilocal in range(0,4):
        iglobal = IEN[i,ilocal]
        for jlocal in range(0,4):
            jglobal = IEN[i,jlocal]

            K[iglobal,jglobal] += kelem[ilocal,jlocal]
            M[iglobal,jglobal] += melem[ilocal,jlocal]
```

```
#-------------------------------------------------------------------------

A = (M/dt + beta*K)

b = np.zeros(npoints, dtype='float')

bcc = np.zeros(npoints, dtype='float')

Q = (q/(rho*cv)) * np.ones( (npoints), dtype='float')

T = T0*TNumeric

#----------------------C.
    C.'s-----------------------------------------------
A = A.tocsr()
for i in cc2D:

    if X[i] == 0:
        T[i] = 0.0
        row = A.getrow(i)
        indices = row.indices

        for col in indices:
          # zero row
              A[i,col] = 0.0

        # set 1 in the diagonal(col)
        A[i,i] = 1.0

    if X[i] == 1:
        T[i] = 0.0
        row = A.getrow(i)
        indices = row.indices
```

```
    for col in indices:
      # zero row
          A[i,col] = 0.0

    # set 1 in the diagonal(col)
    A[i,i] = 1.0

if Y[i] == 0:
    T[i] = 0.0
    row = A.getrow(i)
    indices = row.indices

    for col in indices:
      # zero row
          A[i,col] = 0.0

    # set 1 in the diagonal(col)
    A[i,i] = 1.0

if Z[i] == 0:
    T[i] = 0.0
    row = A.getrow(i)
    indices = row.indices

    for col in indices:
      # zero row
          A[i,col] = 0.0

    # set 1 in the diagonal(col)
    A[i,i] = 1.0

if Z[i] == 1:
    Q[i] = (-1.0)*h*T0
```

```
A = A.tolil()


for t in range(0,nIter):
    Tv=0.0
    Tva=0.0
    vtot=0.0
# Analtico ----------------------------------------------------------------------
    if t == 0:

        TAnalitic = T0*np.ones(npoints, dtype='float')

        for ponto in range(0,npoints):

            if X[ponto] == 0:
                TAnalitic[ponto] = 0.0

            if X[ponto] == 1:
                TAnalitic[ponto] = 0.0

            if Y[ponto] == 0:
                TAnalitic[ponto] = 0.0

            if Z[ponto] == 0:
                TAnalitic[ponto] = 0.0

    else:
        TAnalitic = np.zeros(npoints, dtype='float')
        for ponto in range(0, npoints):

            for n in range(1,7):

                for m in range(0,6):
```

```
                for p in range(0,6):

                    xis = n*np.pi

                    yip = (2*m + 1)*np.pi/2.0

                    zet = z1[p]

                    lamb = np.sqrt(xis**2 + yip**2 + zet**2)

                    a =  T0 *
                        ((1-np.cos(zet))*(1-np.cos(yip))*(-np.cos(xis)+1))/(xis*yip*zet)

                    b =
                        (1-0.5*(1+np.sin(2*xis)/(2*xis)))*(1-0.5*(1+np.sin(2*yip)/(2*yip

                    C = a/b

                    TAnalitic[ponto] += C *
                        np.sin(xis*X[ponto])*np.sin(yip*Y[ponto])*np.sin(zet*Z[ponto])*n

point_data = {'temperatura' : TAnalitic}
if t < 10:
    meshio.write_points_cells('SoluoAnalitica00' + str(t) + '.vtk',
                              msh.points,
                              msh.cells,
                              point_data=point_data,
                              )
if 9 < t < 100:
    meshio.write_points_cells('SoluoAnalitica0' + str(t) + '.vtk',
                              msh.points,
                              msh.cells,
                              point_data=point_data,
```

```
                                )
    if 99 < t:
        meshio.write_points_cells('SoluoAnalitica' + str(t) + '.vtk',
                                msh.points,
                                msh.cells,
                                point_data=point_data,
                                )
# Numrico ------------------------------------------------------------------------
    b = (M/dt)@T + M@Q

    for i in cc2D:

        if X[i] == 0:
            b[i] = 0.0

        if X[i] == 1:
            b[i] = 0.0

        if Y[i] == 0:
            b[i] = 0.0

        if Z[i] == 0:
            b[i] = 0.0

        if Z[i] == 1:
            Q[i] = (-1)*h*T[i]

    T = spsolve(A.tocsc(),b)

    point_data = {'temperatura' : T}
    if t < 10:
        meshio.write_points_cells('SoluoSemPasta00' + str(t) + '.vtk',
                                msh.points,
                                msh.cells,
```

```
                            point_data=point_data,
                            )
    if 9 < t < 100:
        meshio.write_points_cells('SoluoSemPasta0' + str(t) + '.vtk',
                            msh.points,
                            msh.cells,
                            point_data=point_data,
                            )
    if 99 < t:
        meshio.write_points_cells('SoluoSemPasta' + str(t) + '.vtk',
                            msh.points,
                            msh.cells,
                            point_data=point_data,
                            )


for i in range(0, ne):
    v1 = IEN[i,0]
    v2 = IEN[i,1]
    v3 = IEN[i,2]
    v4 = IEN[i,3]

    tmedio = 0.25*(T[v1]+T[v2]+T[v3]+T[v4])

    tmediaA =
        0.25*(TAnalitic[v1]+TAnalitic[v2]+TAnalitic[v3]+TAnalitic[v4])

    volume = (1.0/6.0) * np.linalg.det([[1,X[v1],Y[v1],Z[v1]],
                                        [1,X[v2],Y[v2],Z[v2]],
                                        [1,X[v3],Y[v3],Z[v3]],
                                        [1,X[v4],Y[v4],Z[v4]]])

    tvolume = tmedio * volume
    tvolumeA = tmediaA * volume
    Tv += tvolume
```

```
        Tva += tvolumeA

        vtot += volume

    mediaTN.append(Tv/vtot)
    mediaTA.append(Tva/vtot)

    for i in range(0,npoints):
        ErrABS[i] = abs((273. + TAnalitic[i]) - (273. + T[i]))

        ErrREL[i] = ErrABS[i] / (abs(273. + TAnalitic[i]))



    norma1Abs.append(np.linalg.norm(ErrABS, 1))
    norma2Abs.append(np.linalg.norm(ErrABS, 2))
    normainfAbs.append(np.linalg.norm(ErrABS, np.inf))

    norma1Rel.append(np.linalg.norm(ErrREL, 1))
    norma2Rel.append(np.linalg.norm(ErrREL, 2))
    normainfRel.append(np.linalg.norm(ErrREL, np.inf))

    tempAnalitico.append(np.linalg.norm(TAnalitic, np.inf))
    tempMEF.append(np.linalg.norm(T, np.inf))

    media.append(np.mean(ErrREL))

now = datetime.now()
current_time = now.strftime("%H:%M:%S")
print("Current Time =", current_time)

print("DONE!")
```

Apêndice B

Código Fonte: Processador

As propriedades termodinâmicas foram utilizadas fora do SI (por estarem em milímetros) a fim de adequar com a modelagem das malhas.

B.1 Código Comparação entre as Malhas

```
from datetime import datetime

now = datetime.now()

current_time = now.strftime("%H:%M:%S")
print("Current Time =", current_time)

import position
import meshio
import numpy as np
import matplotlib.pyplot as plt


#--------------------------Malha-----------------------------------------------
msh= meshio.read('nome da malha.msh')
X = msh.points[:,0]
Y = msh.points[:,1]
Z = msh.points[:,2]
```

```
IEN = msh.cells[3].data
IENbound2D = msh.cells[2].data
IENbound = msh.cells[1].data #IEN

npoints = len(X)
ne = IEN.shape[0]
nf = IENbound2D.shape[0]
ng = IENbound.shape[0]

elementos = ne + nf + ng


# cria lista de nos do contorno
cc = np.unique(IENbound.reshape(IENbound.size))
cc2D = np.unique(IENbound2D.reshape(IENbound2D.size))

#-------------------------Dados-----------------------------------------------
q = 65./(480.) #Fonte de calor [W/mm^2]

Tinf= 25.0

#die - SiO2 = Silica
rhodie = 2.65 * 10**(-6)       #Densidade do material [Kg/mm^3]
cvdie = 712.                   #Calor especifico [J/Kg oC]
kdie = 1.25 * 10**(-3)         #Condutividade termica [J/(mm s oC)]
alphadie = kdie/(rhodie * cvdie) #difusividade termica [mm^2/s]

rhopcb = 1.85 * 10**(-6) #FR-4 #Densidade do material [Kg/mm^3]
cvpcb = 1100.                  #Calor especifico [J/Kg oC]
kxpcb = kypcb = 810.           #Condutividade termica [J/(mm s oC)]
kzpcb = 290.                   #Condutividade termica [J/(mm s oC)]
alphapcbxy = kxpcb/(rhopcb*cvpcb)
alphapcbz = kzpcb/(rhopcb*cvpcb)
alphapcb = (2*alphapcbxy + alphapcbz)/3. #difusividade termica [mm^2/s]
```

```
rhoar = 1.225 * 10**(-9) #Ar  #Densidade do material [Kg/mm^3]
cvar = 1000.                   #Calor especifico [J/Kg oC]
kar = 0.03 * 10**(-3)         #Condutividade termica [J/(mm s oC)]
alphaar = kar/(rhoar*cvar)    #difusividade termica [mm^2/s]

rhoihs = 8.96 * 10**(-6) #Cobre #Densidade do material [Kg/mm^3]
cvihs = 380                  #Calor especifico [J/Kg oC]
kihs = 401 * 10**3            #Condutividade termica [J/(mm s oC)]
alphaihs = kihs/(rhoihs*cvihs) #difusividade termica [mm^2/s]

rhopasta = 4.05 * 10**(-6)#Pasta #Densidade do material [Kg/mm^3]
cvpasta = 1.0                 #Calor especifico [J/Kg oC]
kpasta = 8.9 * 10**(-3)       #Condutividade termica [J/(mm s oC)]
alphapasta = kpasta/(rhopasta*cvpasta) #difusividade termica [mm^2/s]

dt = 0.001 # Fraao  de tempo
nIter = 500 #Numero de interaes
beta = 1.0 #Metodo no tempo para d^2/dx^2

#-------------------------Matrizes K e
    M---------------------------------------

K = np.zeros((npoints,npoints), dtype='float')
M = np.zeros((npoints,npoints), dtype='float')

Kp = np.zeros((npoints,npoints), dtype='float')
Mp = np.zeros((npoints,npoints), dtype='float')

M2 = np.zeros((npoints,npoints), dtype='float')
Mp2 = np.zeros((npoints,npoints), dtype='float')

Q = np.ones((npoints), dtype='float')
```

```
T = np.zeros((npoints), dtype='float')
Tp = np.zeros((npoints), dtype='float')

mediaT=[]
mediaTp=[]

die = []
topo = []

#------------------------Loop -
    Assembling-------------------------------------
for i in range(0, ne):
    v1 = IEN[i,0]
    v2 = IEN[i,1]
    v3 = IEN[i,2]
    v4 = IEN[i,3]

#------------------------Coeficientes b, c, d
    ---------------------------------
    b1 = ((Y[v2] - Y[v4]) * (Z[v3] - Z[v4])) - ((Y[v3] - Y[v4]) * (Z[v2]
        - Z[v4]))
    b2 = ((Y[v3] - Y[v4]) * (Z[v1] - Z[v4])) - ((Y[v1] - Y[v4]) * (Z[v3]
        - Z[v4]))
    b3 = ((Y[v1] - Y[v4]) * (Z[v2] - Z[v4])) - ((Y[v2] - Y[v4]) * (Z[v1]
        - Z[v4]))
    b4 = (b1 + b2 + b3)*(-1.0)

    c1 = ((X[v3] - X[v4]) * (Z[v2] - Z[v4])) - ((X[v2] - X[v4]) * (Z[v3]
        - Z[v4]))
    c2 = ((X[v1] - X[v4]) * (Z[v3] - Z[v4])) - ((X[v3] - X[v4]) * (Z[v1]
        - Z[v4]))
    c3 = ((X[v2] - X[v4]) * (Z[v1] - Z[v4])) - ((X[v1] - X[v4]) * (Z[v2]
        - Z[v4]))
    c4 = (c1 + c2 + c3)*(-1.0)
```

```
d1 = ((X[v2] - X[v4]) * (Y[v3] - Y[v4])) - ((X[v3] - X[v4]) * (Y[v2]
    - Y[v4]))
d2 = ((X[v3] - X[v4]) * (Y[v1] - Y[v4])) - ((X[v1] - X[v4]) * (Y[v3]
    - Y[v4]))
d3 = ((X[v1] - X[v4]) * (Y[v2] - Y[v4])) - ((X[v2] - X[v4]) * (Y[v1]
    - Y[v4]))
d4 = (d1 + d2 + d3)*(-1.0)

volume = (1.0/6.0) * np.linalg.det([[1,X[v1],Y[v1],Z[v1]],
                                    [1,X[v2],Y[v2],Z[v2]],
                                    [1,X[v3],Y[v3],Z[v3]],
                                    [1,X[v4],Y[v4],Z[v4]]])

melem = (volume/20.0) * np.array([[2,1,1,1],
                                  [1,2,1,1],
                                  [1,1,2,1],
                                  [1,1,1,2]])

B = (1.0/(volume * 6.0)) * np.array([[b1,b2,b3,b4],
                                     [c1,c2,c3,c4],
                                     [d1,d2,d3,d4]])

alphamedio=[]
alphamediop=[]

alphamedio.append(position.p(v1))
alphamedio.append(position.p(v2))
alphamedio.append(position.p(v3))
alphamedio.append(position.p(v4))

alphamediop.append(position.pa(v1))
alphamediop.append(position.pa(v2))
alphamediop.append(position.pa(v3))
```

```
    alphamediop.append(position.pa(v4))

    alpham = np.mean(alphamedio)
    alphamp = np.mean(alphamediop)

#----------------Construcao das matrizes K e
    M---------------------------------

    kelem = volume * np.transpose(B)@B

    for ilocal in range(0,4):
        iglobal = IEN[i,ilocal]
        for jlocal in range(0,4):
            jglobal = IEN[i,jlocal]

            K[iglobal,jglobal] += kelem[ilocal,jlocal] * alpham
            M[iglobal,jglobal] += melem[ilocal,jlocal]
            M2[iglobal,jglobal] +=
                melem[ilocal,jlocal]/(position.m2(iglobal))

            Kp[iglobal,jglobal] += kelem[ilocal,jlocal] * alphamp
            Mp[iglobal,jglobal] += melem[ilocal,jlocal]
            Mp2[iglobal,jglobal] +=
                melem[ilocal,jlocal]/(position.m2p(iglobal))

        T[iglobal] = position.t(iglobal)
        Tp[iglobal] = position.t(iglobal)
        Q[iglobal] = position.q(iglobal)


        if position.tf(iglobal) != 0:
            if position.tf(iglobal) not in die:
                die.append(position.tf(iglobal))
```

```
        if position.ta(iglobal) != 0:
            if position.ta(iglobal) not in topo:
                topo.append(position.ta(iglobal))

#-------------------------Finalizacao-----------------------------------------

A = (M/dt + K)

Ap = (Mp/dt + Kp)

b = np.zeros( (npoints), dtype='float')
bp = np.zeros( (npoints), dtype='float')

for i in topo:
    Q[i] = -(3.*10**(-4) * (T[i]-Tinf))

#--------------------Loop
    Transiente-----------------------------------------
a=0
s=10
p=10
for t in range(0,nIter):

    b =  (M/dt)@T + M2@Q

    bp = (Mp/dt)@Tp + Mp2@Q


    if a == 0:

        T = T
        Tp = Tp

        a+=1
```

```
    else:

        for i in topo:
            Q[i] = -(3*10**(-4) * (T[i]-Tinf))

        T = np.linalg.solve(A,b)
        Tp = np.linalg.solve(Ap,bp)

#-------------------------Gravacao em
    VTK-------------------------------------

    if s!=10:
        s+=1
    if s==10:

        point_data = {'temperatura' : T}
        if t < 10:
            meshio.write_points_cells('SoluoSemPasta00' + str(t) + '.vtk',
                                      msh.points,
                                      msh.cells,
                                      point_data=point_data,
                                      )
        if 9 < t < 100:
            meshio.write_points_cells('SoluoSemPasta0' + str(t) + '.vtk',
                                      msh.points,
                                      msh.cells,
                                      point_data=point_data,
                                      )
        if 99 < t:
            meshio.write_points_cells('SoluoSemPasta' + str(t) + '.vtk',
                                      msh.points,
                                      msh.cells,
                                      point_data=point_data,
```

```
                                    )

        point_data = {'temperatura' : Tp}
        if t < 10:
            meshio.write_points_cells('SoluoComPasta00' + str(t) + '.vtk',
                                    msh.points,
                                    msh.cells,
                                    point_data=point_data,
                                    )
        if 9 < t < 100:
            meshio.write_points_cells('SoluoComPasta0' + str(t) + '.vtk',
                                    msh.points,
                                    msh.cells,
                                    point_data=point_data,
                                    )
        if 99 < t:
            meshio.write_points_cells('SoluoComPasta' + str(t) + '.vtk',
                                    msh.points,
                                    msh.cells,
                                    point_data=point_data,
                                    )

        s=0



    mediaT.append(np.mean(T))
    mediaTp.append(np.mean(Tp))

#------------------------Graficos
    Relevantes-----------------------------------

diff = np.zeros((len(mediaT)), dtype='float')
```

```
for i in range(0,len(mediaT)):
    diff[i] = abs(mediaT[i]-mediaTp[i])

t = np.linspace(0.0,dt*nIter,nIter)

plt.figure(figsize=(10,6))
plt.plot(t,mediaT, 'r.', label='Sem pasta')
plt.plot(t,mediaTp, 'b.', label='Com pasta')
plt.title('Temperatura mdia', fontsize=15)
plt.xlabel('Tempo (s)', fontsize=15)
plt.ylabel('Temperatura', fontsize=15)
plt.legend(prop={"size":16})

plt.figure(figsize=(10,6))
plt.plot(t,diff,'y.')
plt.title('Diferena entre as medias das temperaturas', fontsize=15)
plt.xlabel('Tempo (s)', fontsize=15)
plt.ylabel('Temperatura', fontsize=15)

now = datetime.now()

current_time = now.strftime("%H:%M:%S")
print("Current Time =", current_time)
print('DONE!')
```

B.2 Código auxiliar: position.py

```
import meshio
import numpy as np

#--------------------------Malha-----------------------------------------------
msh= meshio.read('nome da malha.msh')
X = msh.points[:,0]
```

```
Y = msh.points[:,1]
Z = msh.points[:,2]
IEN = msh.cells[3].data
IENbound2D = msh.cells[2].data
IENbound = msh.cells[1].data #IEN

npoints = len(X)
ne = IEN.shape[0]
nebound = IENbound.shape[0]

# cria lista de nos do contorno
cc = np.unique(IENbound.reshape(IENbound.size))
cc2D = np.unique(IENbound2D.reshape(IENbound2D.size))

#--------------------------Dados-------------------------------------------------
q = 65./(480.) #Fonte de calor [W/mm^2]

Tinf= 25.0

#die - SiO2 = Silica
rhodie = 2.65 * 10**(-6)       #Densidade do material [Kg/mm^3]
cvdie = 712.                   #Calor especifico [J/Kg oC]
kdie = 1.25 * 10**(-3)         #Condutividade termica [J/(mm s oC)]
alphadie = kdie/(rhodie * cvdie) #difusividade termica [mm^2/s]

rhopcb = 1.85 * 10**(-6) #FR-4 #Densidade do material [Kg/mm^3]
cvpcb = 1100.                  #Calor especifico [J/Kg oC]
kxpcb = kypcb = 810.           #Condutividade termica [J/(mm s oC)]
kzpcb = 290.                   #Condutividade termica [J/(mm s oC)]
alphapcbxy = kxpcb/(rhopcb*cvpcb)
alphapcbz = kzpcb/(rhopcb*cvpcb)
alphapcb = (2*alphapcbxy + alphapcbz)/3. #difusividade termica [mm^2/s]

rhoar = 1.225 * 10**(-9) #Ar  #Densidade do material [Kg/mm^3]
```

```
cvar = 1000.                    #Calor especifico [J/Kg oC]
kar = 0.03 * 10**(-3)          #Condutividade termica [J/(mm s oC)]
alphaar = kar/(rhoar*cvar)     #difusividade termica [mm^2/s]

rhoihs = 8.96 * 10**(-6) #Cobre #Densidade do material [Kg/mm^3]
cvihs = 380                   #Calor especifico [J/Kg oC]
kihs = 401 * 10**3             #Condutividade termica [J/(mm s oC)]
alphaihs = kihs/(rhoihs*cvihs) #difusividade termica [mm^2/s]

rhopasta = 4.05 * 10**(-6)#Pasta #Densidade do material [Kg/mm^3]
cvpasta = 1.0                  #Calor especifico [J/Kg oC]
kpasta = 8.9 * 10**(-3)        #Condutividade termica [J/(mm s oC)]
alphapasta = kpasta/(rhopasta*cvpasta) #difusividade termica [mm^2/s]

dt = 0.001 # Fraao  de tempo
nIter = 500 #Numero de interaes
beta = 1.0 #Metodo no tempo para d^2/dx^2

Qq = q



pcb = rhopcb*cvpcb
die = rhodie*cvdie
ar = rhoar*cvar
ihs = rhoihs*cvihs
pasta = rhopasta*cvpasta


def p(x):

    if Z[x] <= 1.2:
        return(alphapcb)
```

```
    if 1.2 < Z[x] <= 2.08:

        if 5.915 < X[x] < 32.875 and 5.01 < Y[x] < 32.69:

            if (14.75 <= X[x] <= 23.75 and 9.55 <= Y[x] <= 29.15):

                return(alphadie)

            else:

                return(alphaar)
        else:

            return(alphaihs)

    if 2.08 < Z[x] < 5:

        return(alphaihs)

    if Z[x] == 5:

        return(alphaihs)

def q(x):

    if Z[x] <= 1.2:
        return(0)

    if 1.2 < Z[x] <= 2.08:

        if 5.915 < X[x] < 32.875 and 5.01 < Y[x] < 32.69:

            if (14.75 <= X[x] <= 23.75 and 9.55 <= Y[x] <= 29.15):
```

```
                return(Qq)

            else:

                return(0)
        else:

            return(0)


    if 2.08 < Z[x] < 5:

        return(0)

    if Z[x] == 5:

        return(0)

def t(x):
    if Z[x] <= 1.2:
        return(25.)

    if 1.2 < Z[x] <= 2.08:

        if 5.915 < X[x] < 32.875 and 5.01 < Y[x] < 32.69:

            if (14.75 <= X[x] <= 23.75 and 9.55 <= Y[x] <= 29.15):

                return(25.)

            else:

                return(25.)
        else:
```

```
            return(25.)

    if 2.08 < Z[x] < 5:

        return(25.)

    if Z[x] == 5:

        return(25.)

def tf(x):

    if 1.2 < Z[x] <= 2.08:

        if 5.915 < X[x] < 32.875 and 5.01 < Y[x] < 32.69:

            if (14.75 <= X[x] <= 23.75 and 9.55 <= Y[x] <= 29.15):

                return(x)
            else:
                return(0)
        else:
            return(0)
    else:
        return(0)

def ta(x):
     if Z[x] <= 1.2:
            return(0)

     if 1.2 < Z[x] <= 2.08:

            if 5.915 < X[x] < 32.875 and 5.01 < Y[x] < 32.69:
```

```
                if (14.75 <= X[x] <= 23.75 and 9.55 <= Y[x] <= 29.15):

                    return(0)

                else:

                    return(0)
            else:

                return(0)

     if 2.08 < Z[x] < 5:

            return(0)

     if Z[x] == 5:

            return(x)

def pa(x):

    if Z[x] <= 1.2:
        return(alphapcb)

    if 1.2 < Z[x] <= 2.08:

        if 5.915 < X[x] < 32.875 and 5.01 < Y[x] < 32.69:

            if (14.75 <= X[x] <= 23.75 and 9.55 <= Y[x] <= 29.15):

                return(alphadie)

            else:
```

```
                return(alphaar)
        else:

            return(alphaihs)

    if 2.08 < Z[x] < 5:

        return(alphaihs)

    if Z[x] == 5:

        return(alphapasta)

def m2(x):

    if Z[x] <= 1.2:
        return(pcb)

    if 1.2 < Z[x] <= 2.08:

        if 5.915 < X[x] < 32.875 and 5.01 < Y[x] < 32.69:

            if (14.75 <= X[x] <= 23.75 and 9.55 <= Y[x] <= 29.15):

                return(die)

            else:

                return(ar)
        else:

            return(ihs)
```

```
    if Z[x] > 2.08:

        return(ihs)

def m2p(x):

    if Z[x] <= 1.2:
        return(pcb)

    if 1.2 < Z[x] <= 2.08:

        if 5.915 < X[x] < 32.875 and 5.01 < Y[x] < 32.69:

            if (14.75 <= X[x] <= 23.75 and 9.55 <= Y[x] <= 29.15):

                return(die)

            else:

                return(ar)
        else:

            return(ihs)

    if  2.08 < Z[x] < 5:

        return(ihs)

    if Z[x] == 5:

        return(pasta)
```

Apêndice C

Código Fonte: Núcleos

As propriedades termodinâmicas foram utilizadas fora do SI (por estarem em milímetros) a fim de adequar com a modelagem das malhas.

C.1 Código Comparação de Núcleos

```
from datetime import datetime
import position8 as position
import meshio
import numpy as np
import matplotlib.pyplot as plt



now = datetime.now()


current_time = now.strftime("%H:%M:%S")
print("Current Time =", current_time)


#---------------------------Malha-----------------------------------------------
msh= meshio.read('nome da malha.msh')
X = msh.points[:,0]
Y = msh.points[:,1]
Z = msh.points[:,2]
IEN = msh.cells[3].data
```

```
IENbound2D = msh.cells[2].data
IENbound = msh.cells[1].data #IEN

npoints = len(X)
ne = IEN.shape[0]
nf = IENbound2D.shape[0]
ng = IENbound.shape[0]

elementos = ne + nf + ng


# cria lista de nos do contorno
cc = np.unique(IENbound.reshape(IENbound.size))
cc2D = np.unique(IENbound2D.reshape(IENbound2D.size))

#-------------------------Dados-----------------------------------------------

qs = (65./(480.))*0.2 #Fonte de calor [W/mm^2]
qc = ((65./(480.))*0.7)/8.
qg = (65./(480.))*0.1

Tinf= 25.0

#die - SiO2 = Silica
rhodie = 2.65 * 10**(-6)       #Densidade do material [Kg/mm^3]
cvdie = 712.                   #Calor especifico [J/Kg oC]
kdie = 1.25 * 10**(-3)         #Condutividade termica [J/(mm s oC)]
alphadie = kdie/(rhodie * cvdie) #difusividade termica [mm^2/s]

rhopcb = 1.85 * 10**(-6) #FR-4 #Densidade do material [Kg/mm^3]
cvpcb = 1100.                  #Calor especifico [J/Kg oC]
kxpcb = kypcb = 810.           #Condutividade termica [J/(mm s oC)]
kzpcb = 290.                   #Condutividade termica [J/(mm s oC)]
alphapcbxy = kxpcb/(rhopcb*cvpcb)
```

```
alphapcbz = kzpcb/(rhopcb*cvpcb)
alphapcb = (2*alphapcbxy + alphapcbz)/3. #difusividade termica [mm^2/s]

rhoar = 1.225 * 10**(-9) #Ar  #Densidade do material [Kg/mm^3]
cvar = 1000.                  #Calor especifico [J/Kg oC]
kar = 0.03 * 10**(-3)         #Condutividade termica [J/(mm s oC)]
alphaar = kar/(rhoar*cvar)    #difusividade termica [mm^2/s]

rhoihs = 8.96 * 10**(-6) #Cobre #Densidade do material [Kg/mm^3]
cvihs = 380                  #Calor especifico [J/Kg oC]
kihs = 401 * 10**3            #Condutividade termica [J/(mm s oC)]
alphaihs = kihs/(rhoihs*cvihs) #difusividade termica [mm^2/s]

rhopasta = 4.05 * 10**(-6)#Pasta #Densidade do material [Kg/mm^3]
cvpasta = 1.0                 #Calor especifico [J/Kg oC]
kpasta = 8.9 * 10**(-3)       #Condutividade termica [J/(mm s oC)]
alphapasta = kpasta/(rhopasta*cvpasta) #difusividade termica [mm^2/s]

dt = 0.001 # Fraao  de tempo
nIter = 500 #Numero de interaes
beta = 1.0 #Metodo no tempo para d^2/dx^2

#-------------------------Matrizes K e
    M--------------------------------------

K = np.zeros((npoints,npoints), dtype='float')
M = np.zeros((npoints,npoints), dtype='float')

Kp = np.zeros((npoints,npoints), dtype='float')
Mp = np.zeros((npoints,npoints), dtype='float')

M2 = np.zeros((npoints,npoints), dtype='float')
Mp2 = np.zeros((npoints,npoints), dtype='float')
```

```
Q = np.ones((npoints), dtype='float')

T = np.zeros((npoints), dtype='float')
Tp = np.zeros((npoints), dtype='float')

mediaT=[]
mediaTp=[]

die = []
topo = []

#------------------------Loop -
    Assembling-------------------------------------

for i in range(0, ne):
    v1 = IEN[i,0]
    v2 = IEN[i,1]
    v3 = IEN[i,2]
    v4 = IEN[i,3]

#------------------------Coeficientes b, c, d
    ----------------------------------

    b1 = ((Y[v2] - Y[v4]) * (Z[v3] - Z[v4])) - ((Y[v3] - Y[v4]) * (Z[v2]
        - Z[v4]))
    b2 = ((Y[v3] - Y[v4]) * (Z[v1] - Z[v4])) - ((Y[v1] - Y[v4]) * (Z[v3]
        - Z[v4]))
    b3 = ((Y[v1] - Y[v4]) * (Z[v2] - Z[v4])) - ((Y[v2] - Y[v4]) * (Z[v1]
        - Z[v4]))
    b4 = (b1 + b2 + b3)*(-1.0)

    c1 = ((X[v3] - X[v4]) * (Z[v2] - Z[v4])) - ((X[v2] - X[v4]) * (Z[v3]
        - Z[v4]))
```

```
c2 = ((X[v1] - X[v4]) * (Z[v3] - Z[v4])) - ((X[v3] - X[v4]) * (Z[v1]
    - Z[v4]))
c3 = ((X[v2] - X[v4]) * (Z[v1] - Z[v4])) - ((X[v1] - X[v4]) * (Z[v2]
    - Z[v4]))
c4 = (c1 + c2 + c3)*(-1.0)

d1 = ((X[v2] - X[v4]) * (Y[v3] - Y[v4])) - ((X[v3] - X[v4]) * (Y[v2]
    - Y[v4]))
d2 = ((X[v3] - X[v4]) * (Y[v1] - Y[v4])) - ((X[v1] - X[v4]) * (Y[v3]
    - Y[v4]))
d3 = ((X[v1] - X[v4]) * (Y[v2] - Y[v4])) - ((X[v2] - X[v4]) * (Y[v1]
    - Y[v4]))
d4 = (d1 + d2 + d3)*(-1.0)

volume = (1.0/6.0) * np.linalg.det([[1,X[v1],Y[v1],Z[v1]],
                                    [1,X[v2],Y[v2],Z[v2]],
                                    [1,X[v3],Y[v3],Z[v3]],
                                    [1,X[v4],Y[v4],Z[v4]]])

melem = (volume/20.0) * np.array([[2,1,1,1],
                                  [1,2,1,1],
                                  [1,1,2,1],
                                  [1,1,1,2]])

B = (1.0/(volume * 6.0)) * np.array([[b1,b2,b3,b4],
                                     [c1,c2,c3,c4],
                                     [d1,d2,d3,d4]])

alphamedio=[]
alphamediop=[]

alphamedio.append(position.p(v1))
alphamedio.append(position.p(v2))
alphamedio.append(position.p(v3))
```

```
    alphamedio.append(position.p(v4))

    alphamediop.append(position.pa(v1))
    alphamediop.append(position.pa(v2))
    alphamediop.append(position.pa(v3))
    alphamediop.append(position.pa(v4))

    alpham = np.mean(alphamedio)
    alphamp = np.mean(alphamediop)

#----------------Construcao das matrizes K e
    M---------------------------------

    kelem = volume * np.transpose(B)@B

    for ilocal in range(0,4):
        iglobal = IEN[i,ilocal]
        for jlocal in range(0,4):
            jglobal = IEN[i,jlocal]

            K[iglobal,jglobal] += kelem[ilocal,jlocal] * alpham
            M[iglobal,jglobal] += melem[ilocal,jlocal]
            M2[iglobal,jglobal] +=
                melem[ilocal,jlocal]/(position.m2(iglobal))

            Kp[iglobal,jglobal] += kelem[ilocal,jlocal] * alphamp
            Mp[iglobal,jglobal] += melem[ilocal,jlocal]
            Mp2[iglobal,jglobal] +=
                melem[ilocal,jlocal]/(position.m2p(iglobal))

        T[iglobal] = position.t(iglobal)
        Tp[iglobal] = position.t(iglobal)
        Q[iglobal] = position.q(iglobal)
```

```
        if position.tf(iglobal) != 0:
            if position.tf(iglobal) not in die:
                die.append(position.tf(iglobal))

        if position.ta(iglobal) != 0:
            if position.ta(iglobal) not in topo:
                topo.append(position.ta(iglobal))

#-------------------------Finalizacao-----------------------------------------

A = (M/dt + K)

Ap = (Mp/dt + Kp)

b = np.zeros( (npoints), dtype='float')
bp = np.zeros( (npoints), dtype='float')

for i in topo:
    Q[i] = -(3.*10**(-4) * (T[i]-Tinf))

#--------------------Loop
    Transiente-----------------------------------------

a=0
s=10
p=10
for t in range(0,nIter):

    b = (M/dt)@T + M2@Q

    bp = (Mp/dt)@Tp + Mp2@Q
```

```
    if a == 0:

        T = T
        Tp = Tp

        a+=1

    else:

        for i in topo:
            Q[i] = -(3*10**(-4) * (T[i]-Tinf))

        T = np.linalg.solve(A,b)
        Tp = np.linalg.solve(Ap,bp)

#-------------------------Gravacao em
    VTK-------------------------------------

    if s!=10:
        s+=1
    if s==10:

        point_data = {'temperatura' : T}
        if t < 10:
            meshio.write_points_cells('SoluoSemPasta00' + str(t) + '.vtk',
                                      msh.points,
                                      msh.cells,
                                      point_data=point_data,
                                      )
        if 9 < t < 100:
            meshio.write_points_cells('SoluoSemPasta0' + str(t) + '.vtk',
                                      msh.points,
                                      msh.cells,
                                      point_data=point_data,
```

```
                                )
    if 99 < t:
        meshio.write_points_cells('SoluoSemPasta' + str(t) + '.vtk',
                                msh.points,
                                msh.cells,
                                point_data=point_data,
                                )

    point_data = {'temperatura' : Tp}
    if t < 10:
        meshio.write_points_cells('SoluoComPasta00' + str(t) + '.vtk',
                                msh.points,
                                msh.cells,
                                point_data=point_data,
                                )
    if 9 < t < 100:
        meshio.write_points_cells('SoluoComPasta0' + str(t) + '.vtk',
                                msh.points,
                                msh.cells,
                                point_data=point_data,
                                )
    if 99 < t:
        meshio.write_points_cells('SoluoComPasta' + str(t) + '.vtk',
                                msh.points,
                                msh.cells,
                                point_data=point_data,
                                )

    s=0

mediaT.append(np.mean(T))
mediaTp.append(np.mean(Tp))
```

```
#------------------------Graficos
    Relevantes-----------------------------------

diff = np.zeros((len(mediaT)), dtype='float')


for i in range(0,len(mediaT)):
    diff[i] = abs(mediaT[i]-mediaTp[i])

t = np.linspace(0.0,dt*nIter,nIter)


plt.figure(figsize=(10,6))
plt.plot(t,mediaT, 'r.', label='Sem pasta')
plt.plot(t,mediaTp, 'b.', label='Com pasta')
plt.title('Temperatura mdia', fontsize=15)
plt.xlabel('Tempo (s)', fontsize=15)
plt.ylabel('Temperatura', fontsize=15)
plt.legend(prop={"size":16})

plt.figure(figsize=(10,6))
plt.plot(t,diff,'y.')
plt.title('Diferena entre as medias das temperaturas', fontsize=15)
plt.xlabel('Tempo (s)', fontsize=15)
plt.ylabel('Temperatura', fontsize=15)

now = datetime.now()

current_time = now.strftime("%H:%M:%S")
print("Current Time =", current_time)
print('DONE!')
```

C.2 Códigos Auxiliar

O código abaixo representa os 5 códigos auxiliares que representam os diferentes posicionametos das fontes de calor pertinentes aos capítulos do trabalho.

```
import meshio
import numpy as np

#---------------------------Malha-----------------------------------------------
msh= meshio.read('nome da malha.msh')
X = msh.points[:,0]
Y = msh.points[:,1]
Z = msh.points[:,2]
IEN = msh.cells[3].data
IENbound2D = msh.cells[2].data
IENbound = msh.cells[1].data #IEN

npoints = len(X)
ne = IEN.shape[0]
nebound = IENbound.shape[0]

# cria lista de nos do contorno
cc = np.unique(IENbound.reshape(IENbound.size))
cc2D = np.unique(IENbound2D.reshape(IENbound2D.size))

#-------------------------Dados-----------------------------------------------
qs = (65./(480.))*0.2 #Fonte de calor [W/mm^2]
qc = ((65./(480.))*0.7)/8.
qg = (65./(480.))*0.1

Tinf= 25.0

#die - SiO2 = Silica
rhodie = 2.65 * 10**(-6)       #Densidade do material [Kg/mm^3]
cvdie = 712.                   #Calor especifico [J/Kg oC]
```

```
kdie = 1.25 * 10**(-3)        #Condutividade termica [J/(mm s oC)]
alphadie = kdie/(rhodie * cvdie) #difusividade termica [mm^2/s]

rhopcb = 1.85 * 10**(-6) #FR-4 #Densidade do material [Kg/mm^3]
cvpcb = 1100.                  #Calor especifico [J/Kg oC]
kxpcb = kypcb = 810.           #Condutividade termica [J/(mm s oC)]
kzpcb = 290.                   #Condutividade termica [J/(mm s oC)]
alphapcbxy = kxpcb/(rhopcb*cvpcb)
alphapcbz = kzpcb/(rhopcb*cvpcb)
alphapcb = (2*alphapcbxy + alphapcbz)/3. #difusividade termica [mm^2/s]

rhoar = 1.225 * 10**(-9) #Ar  #Densidade do material [Kg/mm^3]
cvar = 1000.                   #Calor especifico [J/Kg oC]
kar = 0.03 * 10**(-3)          #Condutividade termica [J/(mm s oC)]
alphaar = kar/(rhoar*cvar)     #difusividade termica [mm^2/s]

rhoihs = 8.96 * 10**(-6) #Cobre #Densidade do material [Kg/mm^3]
cvihs = 380                   #Calor especifico [J/Kg oC]
kihs = 401 * 10**3             #Condutividade termica [J/(mm s oC)]
alphaihs = kihs/(rhoihs*cvihs) #difusividade termica [mm^2/s]

rhopasta = 4.05 * 10**(-6)#Pasta #Densidade do material [Kg/mm^3]
cvpasta = 1.0                  #Calor especifico [J/Kg oC]
kpasta = 8.9 * 10**(-3)        #Condutividade termica [J/(mm s oC)]
alphapasta = kpasta/(rhopasta*cvpasta) #difusividade termica [mm^2/s]

dt = 0.001 # Fraao  de tempo
nIter = 500 #Numero de interaes
beta = 1.0 #Metodo no tempo para d^2/dx^2

Qs = qs

Qc = qc
```

```
Qg = qg


def p(x):

    if Z[x] <= 1.2:
        return(alphapcb)

    if 1.2 < Z[x] <= 2.08:

        if 5.915 < X[x] < 32.875 and 5.01 < Y[x] < 32.69:

            if (14.75 <= X[x] <= 23.75 and 9.55 <= Y[x] <= 29.15):

                return(alphadie)

            else:

                return(alphaar)
        else:

            return(alphaihs)

    if Z[x] > 2.08:

        return(alphaihs)

def q(x):

    if Z[x] <= 1.2:
        return(0)

    if 1.2 < Z[x] <= 2.08:
```

```
if 5.915 < X[x] < 32.875 and 5.01 < Y[x] < 32.69:

    if (14.75 <= X[x] <= 23.75 and 9.55 <= Y[x] <= 29.15):

        if 26.79 < Y[x] <= 29.15: #Sistema

            return(Qs)

        if (19.35 <= X[x] <= 23.75) and (23.956 <= Y[x] <=
            26.79):#limite do Core 1
        #ou
        #if  (23.956 <= Y[x] <= 26.79):#limite do Core 1 e 2
        #ou
        #if (21.11 <= Y[x] <= 26.79):#limite do Core 1, 2, 3 e 4
        #ou
        #if (18.27 <= Y[x] <= 26.79):#limite do Core 1, 2, 3, 4, 5
            e 6
        #ou
        #if (15.43 <= Y[x] <= 26.79):#limite do Core 1, 2, 3, 4,
            5, 6, 7 e 8

            return(Qc)

        if 9.55 <= Y[x] < 15.43: #GPU

            return(Qg)

        else:

            return(0)

    else:

        return(0)
```

```
        else:

            return(0)


    if 2.08 < Z[x] < 5:

        return(0)

    if Z[x] == 5:

        return(0)

def t(x):
    if Z[x] <= 1.2:
        return(25.)

    if 1.2 < Z[x] <= 2.08:

        if 5.915 < X[x] < 32.875 and 5.01 < Y[x] < 32.69:

            if (14.75 <= X[x] <= 23.75 and 9.55 <= Y[x] <= 29.15):

                return(25.)

            else:

                return(25.)
        else:

            return(25.)

    if 2.08 < Z[x] < 5:
```

```
        return(25.)

    if Z[x] == 5:

        return(25.)

def tf(x):

    if 1.2 < Z[x] <= 2.08:

        if 5.915 < X[x] < 32.875 and 5.01 < Y[x] < 32.69:

            if (14.75 <= X[x] <= 23.75 and 9.55 <= Y[x] <= 29.15):

                return(x)
            else:
                return(0)
        else:
            return(0)
    else:
        return(0)

def ta(x):
     if Z[x] <= 1.2:
            return(0)

     if 1.2 < Z[x] <= 2.08:

            if 5.915 < X[x] < 32.875 and 5.01 < Y[x] < 32.69:

                if (14.75 <= X[x] <= 23.75 and 9.55 <= Y[x] <= 29.15):

                    return(0)
```

```
                else:

                    return(0)
            else:

                return(0)

    if 2.08 < Z[x] < 5:

            return(0)

    if Z[x] == 5:

            return(x)

def pa(x):

    if Z[x] <= 1.2:
        return(alphapcb)

    if 1.2 < Z[x] <= 2.08:

        if 5.915 < X[x] < 32.875 and 5.01 < Y[x] < 32.69:

            if (14.75 <= X[x] <= 23.75 and 9.55 <= Y[x] <= 29.15):

                return(alphadie)

            else:

                return(alphaar)
        else:

            return(alphaihs)
```

```
    if 2.08 < Z[x] < 5:

        return(alphaihs)

    if Z[x] == 5:

        return(alphapasta)

pcb = rhopcb*cvpcb
die = rhodie*cvdie
ar = rhoar*cvar
ihs = rhoihs*cvihs
pasta = rhopasta*cvpasta

def m2(x):

    if Z[x] <= 1.2:
        return(pcb)

    if 1.2 < Z[x] <= 2.08:

        if 5.915 < X[x] < 32.875 and 5.01 < Y[x] < 32.69:

            if (14.75 <= X[x] <= 23.75 and 9.55 <= Y[x] <= 29.15):

                return(die)

            else:

                return(ar)
        else:

            return(ihs)
```

```
    if Z[x] > 2.08:

        return(ihs)

def m2p(x):

    if Z[x] <= 1.2:
        return(pcb)

    if 1.2 < Z[x] <= 2.08:

        if 5.915 < X[x] < 32.875 and 5.01 < Y[x] < 32.69:

            if (14.75 <= X[x] <= 23.75 and 9.55 <= Y[x] <= 29.15):

                return(die)

            else:

                return(ar)
        else:

            return(ihs)

    if  2.08 < Z[x] < 5:

        return(ihs)

    if Z[x] == 5:

        return(pasta)
```