UNIVERSIDADE FEDERAL DO RIO DE JANEIRO
DEPARTAMENTO DE ENGENHARIA MECÂNICA
CURSO DE ENGENHARIA MECÂNICA

THYAGO ARAUJO CAPITANIO
DRE : 116183833

IMPLEMENTAÇÃO DO MÉTODO DE ELEMENTOS FINITOS EM PYTHON PARA O ESTUDO PARAMÉTRICO DE DISSIPADORES DE CALOR DE PROCESSADORES COMPUTACIONAIS

Orientador: Gustavo Rabello dos Anjos

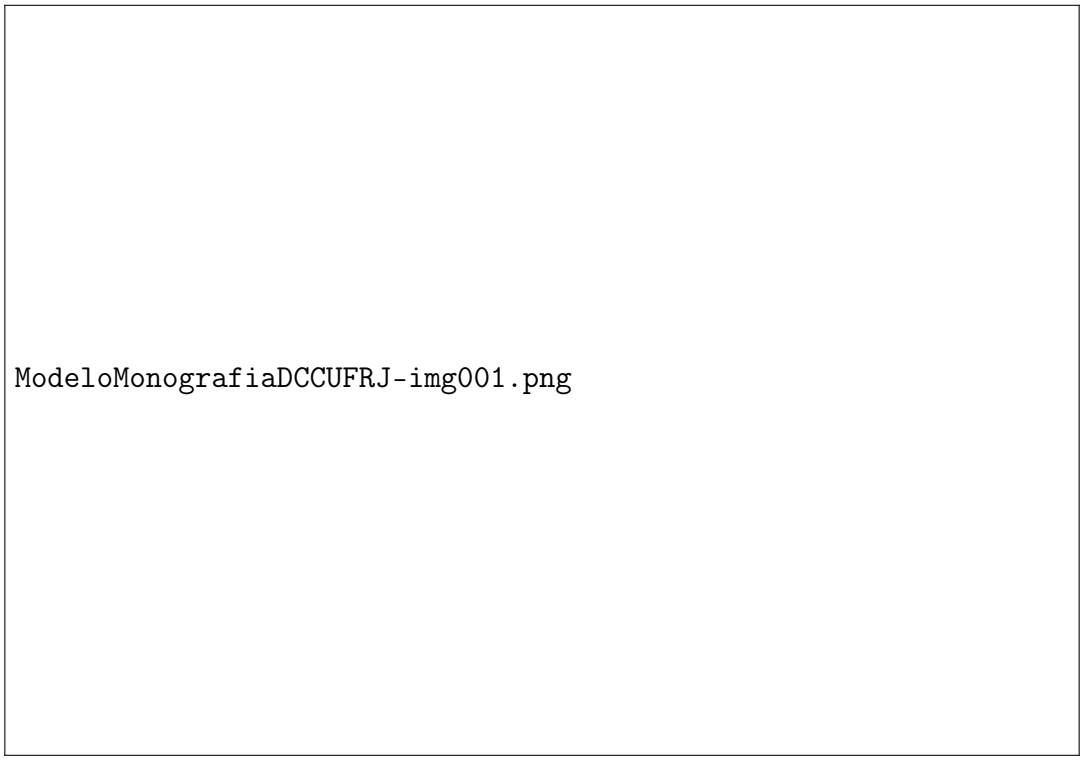
RIO DE JANEIRO
2023

THYAGO ARAUJO CAPITANIO

IMPLEMENTAÇÃO DO MÉTODO DE ELEMENTOS FINITOS EM PYTHON PARA O ESTUDO PARAMÉTRICO DE DISSIPADORES DE CALOR DE PROCESSADORES COMPUTACIONAIS

Trabalho de conclusão de curso de graduação apresentado ao Departamento de Engenharia Mecânica da Universidade Federal do Rio de Janeiro como parte dos requisitos para obtenção do grau de Bacharel em Engenharia Mecânica .

RIO DE JANEIRO
2023


ModeloMonografiaDCCUFRJ-img001.png

THYAGO ARAUJO CAPITANIO

IMPLEMENTAÇÃO DO MÉTODO DE ELEMENTOS FINITOS EM PYTHON PARA O ESTUDO PARAMÉTRICO DE DISSIPADORES DE CALOR DE PROCESSADORES COMPUTACIONAIS

Trabalho de conclusão de curso de graduação apresentado ao Departamento de Engenharia Mecânica da Universidade Federal do Rio de Janeiro como parte dos requisitos para obtenção do grau de Bacharel em Engenharia Mecânica .

Aprovado em ___ de _____ de _____

BANCA EXAMINADORA:

Prof. Gustavo Rabello dos Anjos, Ph.D.

Prof. Gabriel Lisbôa Veríssimo, D.Sc.

Nisio de Carvalho Lobo Brum, D.Sc.

AGRADECIMENTOS

Gostaria de começar por agradecer a meus pais, Silvana e Nymes, por me trazerem ao mundo e sempre terem feito de tudo para garantir que eu teria acesso às melhores oportunidades e que teria condições de me esforçar para garanti-las. Agradeço também a minha irmã Thayanna e a todos da família, mesmo os que já se foram, por todos os bons momentos e pelo apoio em toda a minha trajetória até aqui.

Agradeço também aos meus amigos, meus colegas de copo e de cruz, citando Chico Buarque. Foram muitos momentos difíceis nessa trajetória acadêmica, mas os bons momentos certamente foram mais e mais intensos. Um agradecimento especial a Thiago Ribeiro, Bernardo Duque e Mariana Pereira pelos anos de amizade dentro e fora da UFRJ.

Sou grato também a todos que encontrei durante o intercâmbio acadêmico, estes encontros foram de grandíssimo impacto pessoal, profissional e acadêmico. Agradeço principalmente à equipe acadêmica do INSA Toulouse, à equipe que me acolheu na Faurecia (Frédéric Guildbaud, Grégoire Boulard, Naveenkumar Sira, Julie Pouzoulet, Emine Özen, etc.) e à equipe da DARI UFRJ que me permitiu ter essa experiência tão enriquecedora.

Por fim, gostaria de agradecer a todos os profissionais que fizeram parte da minha formação dentro da UFRJ, aos profissionais mais diversos que permitem que esta universidade continue oferecendo ensino gratuito da mais alta qualidade. E sem dúvidas, um obrigado especial a Gustavo Rabello, orientador desse trabalho, tanto pela orientação quanto por todos os ensinamentos em Métodos Matemáticos e Transferência de Calor 2 que foram valiosíssimos até então.

"Mas eu o tentarei,
como ele próprio aconselhava,
pois o importante mesmo é tentar,
mesmo o impossível"

Jorge Amado
A morte e a morte de Quincas Berro d'Água

RESUMO

O projeto descrito neste relatório visa modelar o fenômeno de transferência de calor entre componentes presentes em um processador computacional típico e a partir disso executar um estudo paramétrico para o melhor entendimento do dissipador de calor. O método utilizado para a determinação das distribuições de temperatura nesses componentes foi uma mistura do método de elementos finitos (MEF) para a discretização espacial e o método de diferenças finitas (MDF) para a discretização temporal. Essa mistura é capaz de determinar soluções numéricas aproximadas para problemas em regime transiente regidos por equações diferenciais parciais cuja solução analítica é de difícil ou impossível obtenção. Estando o projeto incluído no contexto de conclusão de um curso de engenharia mecânica, um de seus objetivos foi também de desenvolver as competências de desenvolvimento de código próprio em *python* para tais simulações numéricas usando MEF e MDF e o melhor entendimento do funcionamento de *softwares* de código comercial que utilizam esses métodos.

Palavras-chave: *python*, MEF, MDF, processadores computacionais, distribuição de temperatura.

ABSTRACT

The project described in this report aims to model the phenomenon of heat transfer between components of a typical computer processor and from that run a parametric study to better understand the heat sink. To determine the temperature distributions in said components, a mix of finite element method (FEM) for the spatial discretization and the finite difference method (FDM) for the temporal discretization was used. This mix allows us to determine approximated numerical solutions to phenomena driven by partial differential equations with analytical solutions which are hard or impossible to determine. This project is also included in the context of the completion of a mechanical engineering undergrad course, so one of its goals was also to improve code-writing skills using python for numerical simulations using FEM and FDM methods and also to improve the understanding of commerical code that use those methods.

Keywords: python, FEM, FDM, computer processors, temperature distributions.

SUMÁRIO

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

CAD	*Computer-assisted drawing*
MEF	Método de Elementos Finitas
MDF	Método de Diferenças Finitas
EMA	Erro Médio Absoluto
REQM	Raíz do Erro Quadrático Médio
EM	Erro Máximo
EDP	Equação Diferencial Parcial
POO	Programação Orientada a Objeto
CFD	*Computational Fluid Dynamics*

LISTA DE SÍMBOLOS

Ω	Domínio do Sistema
Γ	Contorno do Sistema
T	Campo de temperatura em função da posição e tempo
Q	Campo de fonte de calor em função da posição e tempo
κ	Condutividade térmica do material
κ_x	Condutividade térmica do material no eixo x
ρ	Massa específica do material
c_v	Capacidade calorífica do material
α	Difusividade térmica do material
α_x	Difusividade térmica do material no eixo x
μ	Viscosidade dinâmica do fluido
ν	Viscosidade cinemática do fluido
h	Coeficiente de transferência de calor por convecção
Re	Número de Reynolds
Nu	Número de Nusselt
Pr	Número de Prandtl
∇^2	Operador Laplaciano

1 INTRODUÇÃO

1.1 CONTEXTUALIZAÇÃO DO PROBLEMA

Com a intensa digitalização de diversos aspectos da vida, seja em esferas pessoais, profissionais ou acadêmicas, existe uma crescente demanda de componentes de processamento menores, mais eficientes e mais performantes. Um dos pontos de limitação da performance de uma placa de processamento pode ser a alta temperatura atingida pelos componentes dessa placa, principalmente os núcleos de CPU e GPU. O difusor de temperatura serve justamente como um trocador de calor que se utiliza de um fluido externo (geralmente ar, mas ocasionalmente um fluido refrigerante) para o arrefecimento desses componentes. Esse componente está definido em mais detalhes da subseção 1.1.1.

1.1.1 Difusor de temperatura

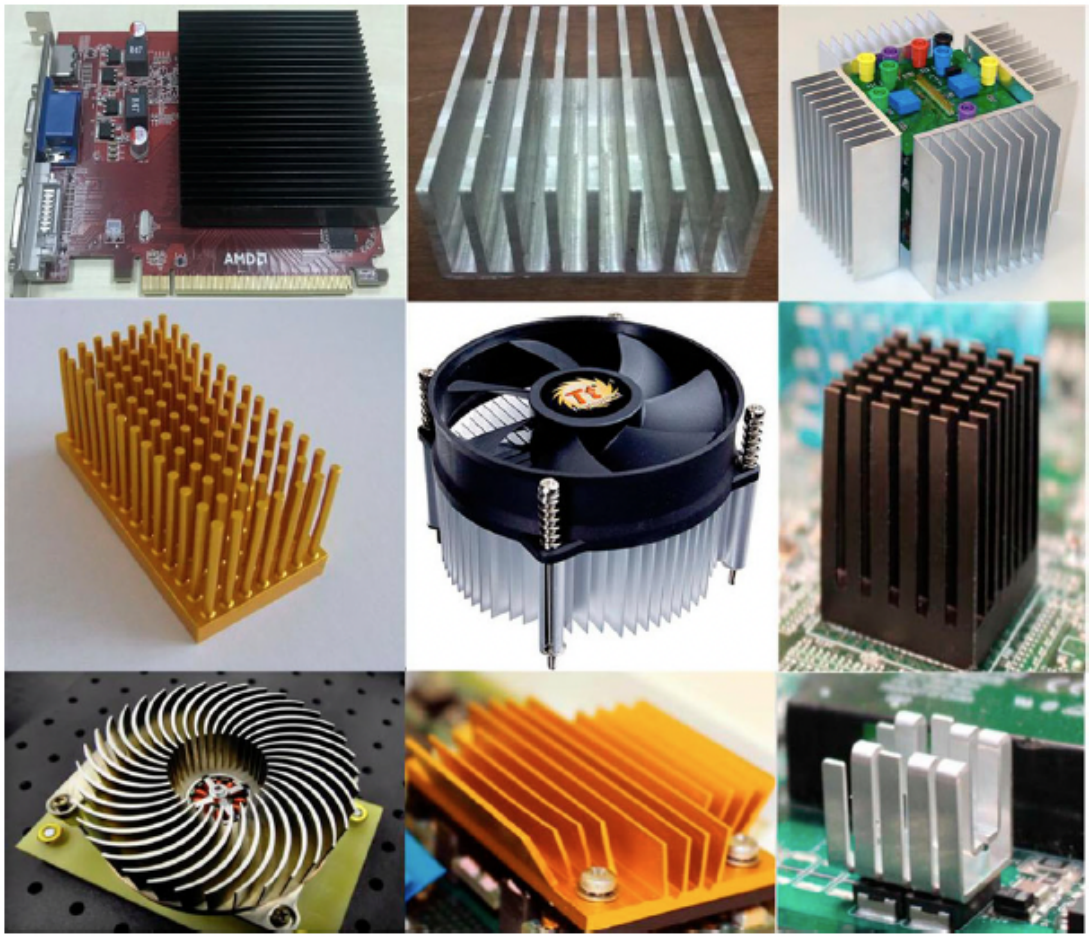


Figura 1 – Diferente tipos de difusor de calor

Fonte: (GENC et al., 2021)

O difusor de temperatura, como um trocador de calor, tem como objetivo intensificar a troca de calor que ocorreria naturalmente entre os componentes e os elementos vizinhos (outros componentes e o próprio ar) por condução, convecção e/ou radiação. Essa intensificação se dá sempre pelo aumento da superfície de troca entre os componentes quentes e o fluido refrigerante. Ocasionalmente, é aplicado também um ventilador para aumentar as trocas por convecção aumentando o coeficiente de transferência de calor convectiva.

Como mostrado na figura 1, as possibilidades de geometria para difusores de calor disponíveis no mercado são diversas, porém geralmente consiste de diversas placas ou pinos verticais. Estes difusores são geralmente fabricados em dois possíveis materiais: cobre ou alumínio. Os dois materiais possuem coeficientes de condutividade térmica considerados apropriados para tal tarefa e também são materiais próprios para a fabricação por extrusão. O processo de fabricação de um difusor de calor como os mostrados na figura 1 é geralmente uma combinação de extrusão e usinagem.

Geometrias mais complexas podem ser atingidas com técnicas de fabricação mais robustas, como a fabricação aditiva. Essas geometrias mais complexas podem inclusive ser fruto de uma otimização topológica como mostra na figura 2 (FAWAZ et al., 2022). Nesse estudo, foi feita uma otimização topológica para a determinação do difusor de calor mais eficiente dado o um conjunto de restrições.

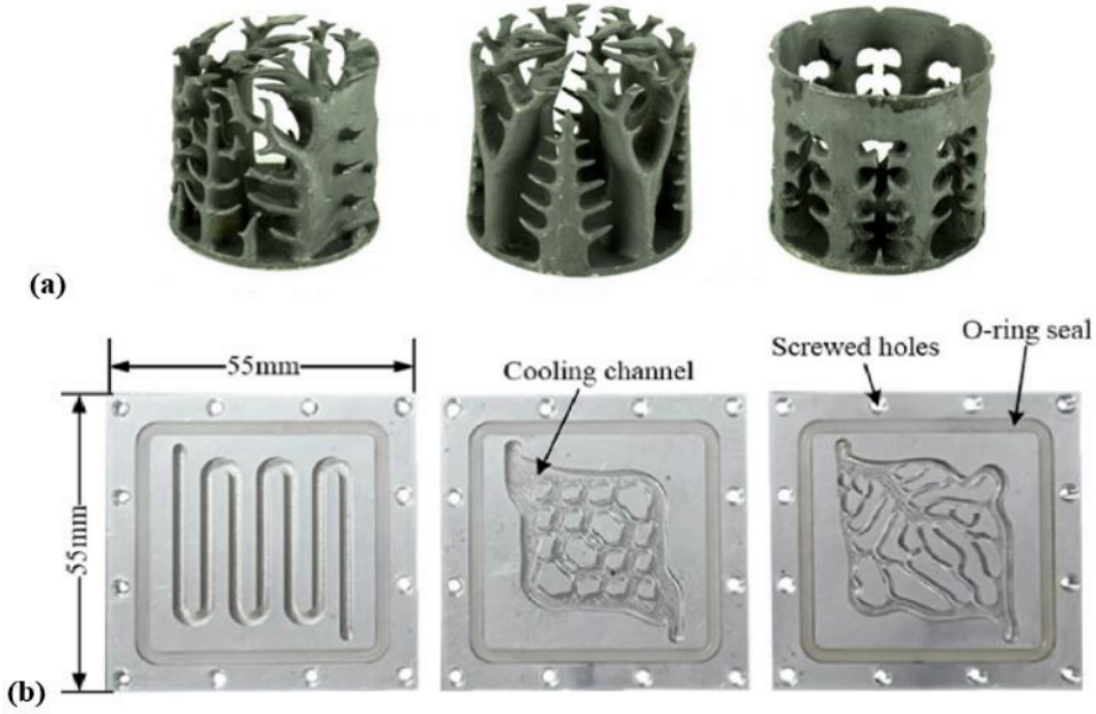


Figura 2 – Estudo de (FAWAZ et al., 2022) para a otimização topológica de um difusor de calor

1.1.2 Possíveis métodos de fabricação do dissipador

Sendo dissipadores de calor componentes de baixa complexidade mecânica e alta versatilidade de usos, é esperado que haja no mercado múltiplos métodos para a sua fabricação. Um artigo de (COOLING, 2007) cita seis métodos de fabricação encontrados no mercado para dissipadores de calor (figura 3).

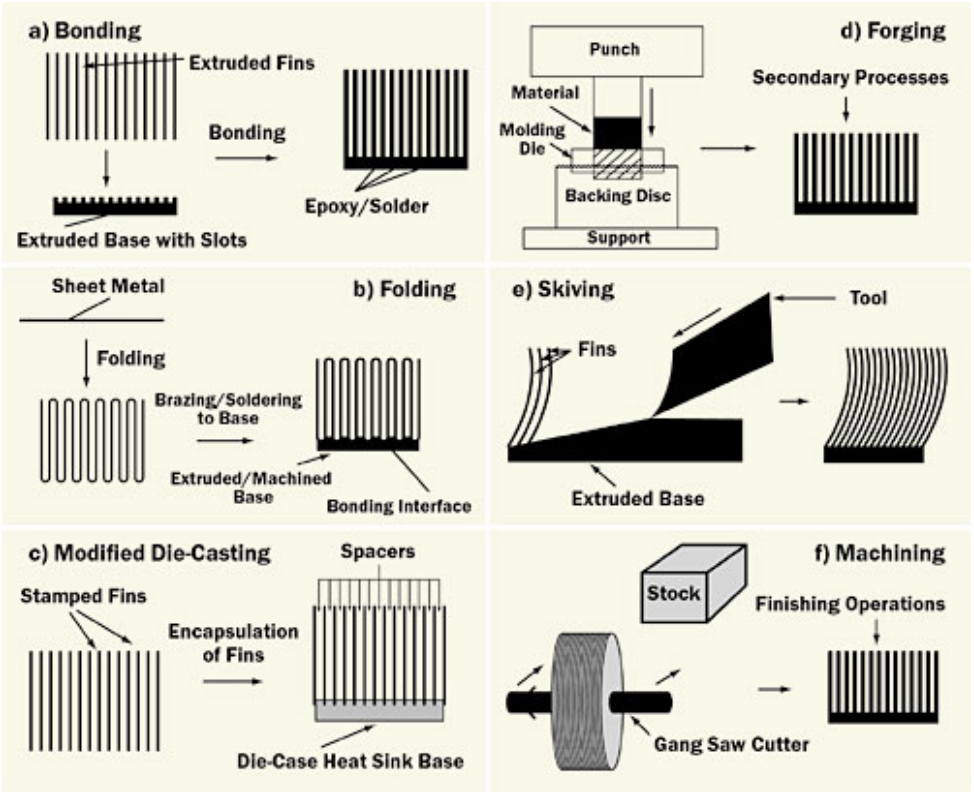


Figura 3 – Artigo de (COOLING, 2007) citando diferente métodos de fabricação de dissipadores de calor

1.1.3 Inovações nos dissipadores de calor

Metodologias mais complexas e ferramentas mais poderosas geram projetos cada vez mais complexos e performantes de dissipadores de calor. Um artigo de (SIEMENS, 2021) descreve certas etapas de um projeto de um dissipador de calor por convecção natural de geometria não convencional.

O projeto se utiliza de simulações CFD (*computational fluid dynamics*) e termais acopladas utilizando o método de volumes finitos. Essas simulações permitem que haja uma estimação precisa das trocas convectivas ao se simular o sólido e o fluido fluidicamente/-termicamente. Como mencionado na subseção 1.1.1, tais geometrias complexas podem ser fabricadas com o auxílio de fabricação aditiva. Nesse artigo foi utilizada a impressão 3D do tipo feixe de elétrons (SIEMENS, 2021).

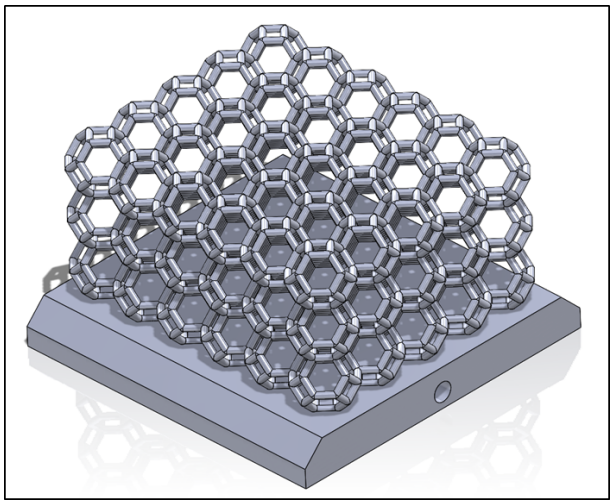


Figura 4 – Dissipador de calor por convecção natural simulado utilizando softwares comerciais de simulações CFD/Termais acopladas

Fonte: (SIEMENS, 2021)

1.2 OBJETIVO DO TRABALHO

O relatório visa detalhar o desenvolvimento de um trabalho de análise paramétrica de um dissipador de calor numa placa de processamento (ver seção 1.1.1). Para esse estudo, será utilizado um ferramental contendo diversos instrumentos de pré-processamento, simulação e pós-processamento. Um dos objetivos do trabalho foi de desenvolver ao máximo minhas competências em programação, sendo assim, considerando o tempo de desenvolvimento do projeto como 6 meses, todas as ferramentas que puderam ser implementadas em código próprio nesse tempo foram assim feitas. Ferramental utilizado no desenvolvimento do projeto:

a) Geração do modelo CAD: *FreeCAD*

b) Geração da malha: *Gmsh*

c) Solução do sistema em elementos finitos: *Python em código próprio*

d) Resolução do sistema linear: *Biblioteca Numpy do Python*

e) Pós-processamento/geração de dados: *Código próprio em Python*

f) Visualização dos dados: *Paraview*

1.3 RELEVÂNCIA DO ESTUDO

Além do desenvolvimento do aluno no contexto de projeto final de um curso de engenharia mecânica, o projeto tem também como objetivo indicar possíveis pontos de melhoria para o desenvolvimento de difusores de calor mais eficientes e performantes.

2 FUNDAMENTAÇÃO TEÓRICA

2.1 TRANSFERÊNCIA DE CALOR

Ösizik (OZISIK, 1985) entende a ciência de transferência de calor como o estudo da taxa de trocas térmicas em um sistema. A energia transferida pelo fluxo de calor não podendo ser medida diretamente, se faz necessário estabelecer relações dessa quantidade imensurável a uma outra mensurável. A quantidade mensurável em questão é a temperatura, uma grandeza facilmente mensurável e de fácil compreensão. Essa grandeza se torna então de grande interesse para engenheiros e cientistas, sendo ela necessária para determinar as transferências de calor em determinado sistema.

As transferências de calor são geralmente divididas em três grupos: Condução, Convecção e Radiação. É importante notar que essa divisão é feita para fins didáticos e que na realidade as transferências de calor (e por consequência a distribuição de temperatura) serão controladas por efeitos combinados dos três grupos. As subseções a seguir explicam em detalhes as três formas de transferência de calor e seus equacionamentos.

2.1.1 Condução

A condução é uma forma de transferência de calor de um ponto de alta temperatura (e portanto alta agitação molecular) para um ponto de baixa temperatura (e portanto baixa agitação molecular) por meio da transferência dessa energia cinética por impacto molecular (no caso de fluidos em repouso) ou transferência de elétrons (caso de metais).

Vale notar que metais são bons condutores térmicos justamente por como se dão as ligações do tipo metálica. Entende-se que os diferentes átomos metálicos entram em ligação onde os elétrons possuem alto nível de liberdade de movimentação dentro da malha metálica.

A equação empírica que modela a transferência de calor é atribuída ao físico e matemático francês Joseph Fourier, apesar de ter sido também anunciada por Jean-Baptiste Biot em observações experimentais. A equação indica que dado um gradiente de temperatura T presente numa área A e numa determinada direção x, o fluxo de calor q_x por condução é proporcional a este gradiente segundo a equação 2.1. A constante de proporcionalidade presente na equação é dependente do material e da temperatura e é o que se chama de condutividade térmica.

$$q_x = -kA\frac{dT}{dx} \tag{2.1}$$

2.1.2 Convecção

A convecção se dá quando existe um diferencial de temperatura entre um fluido em movimento e um elemento parado (seja ele um sólido ou um fluido parado). Se essa movimentação for impulsionada por um mecanismo externo como uma bomba, ventilador ou soprador, consideramos o fenômeno de *convecção forçada*. Por outro lado, essa movimentação pode se dar por movimentos de ascenção ou descensão devido a diferenças de densidade causados pelos diferenciais de temperatura, o que configuraria um fenômeno de *convecção natural*.

Um exemplo de convecção natural é a troca que acontece no nível do asfalto em dias quentes. Com a radiação solar, o asfalto aquece desproporcionalmente ao ar, gerando um grande diferencial de temperatura. O asfalto quente por sua vez aquece o ar nas camadas mais baixas, que por sua vez vê uma diminuição do valor da sua densidade e tende então a subir, renovando o ar em contato com o asfalto.

A equação proposta por Isaac Newton para a modelagem da convecção como forma de troca de calor é a equação 2.2. A equação diz que a taxa de transferência de calor entre os dois componentes (elemento parado e fluido em movimento) é proporcional à área A na qual essa troca ocorre e à diferença de temperatura entre o fluido T_f e a superfície T.

$$\dot{Q} = hA(T_f - T) \tag{2.2}$$

Sobre a equação 2.2, vale notar duas coisas:

a) A equação desta foram é escrita para o objeto parado, ou seja, quando $\dot{Q}$ é negativo, o objeto está transferindo calor para o fluido

b) A constante de proporcionalidade h (unidades $W/m^2.K$) é em geral difícil de se obter com precisão e para fins de engenharia é normalmente aproximada de acordo com o tipo de convecção (forçada ou natural), o regime de escoamento, a geometria e características do fluido

2.1.3 Radiação

Radiação é o meio de transferência de calor onde a energia transferida se dá por meio de radiação térmica. Esta radiação térmica se dá na forma de ondas eletromagnéticas segundo a teoria eletromagnética de Maxwell ou na forma de fótons discretos segundo a hipótese de Planck. Dentre as três formas de transferência de calor apresentadas nesse relatório, é a única que não precisa de um meio massivo para se propagar, podendo se propagar no vácuo absoluto.

Essa forma de transferência de calor é habitualmente observada em fenômenos onde o sol esquenta outros objetos, como o exemplo citado na subseção 2.1.2 com o sol esquentando o asfalto.

A equação que modela o fenômeno de transferência de calor por troca de radiação é a lei de Stefan-Boltzmann, presente na equação 2.3. A equação indica que um corpo a temperatura T_1, possuindo uma emissividade ε com área de troca A troca uma quantidade de calor Q com um corpo de temperatura T_2.

$$Q = -\varepsilon A\sigma(T_1^4 - T_2^4) \tag{2.3}$$

2.2 MÉTODO DE ELEMENTOS FINITOS

O estudo de engenharia é repleto de fenômenos que podem ser descritos por equações diferenciais ordinarias ou parciais de difícil ou impossível solução analítica. Antes do advento de simulações numéricas, tais casos eram estudados por meio de experimentos ou de modelos analíticos simplificados. O problema logo surge pois os modelos analíticos simplificados com frequência não conseguem descrever a complexidade dos fenômenos da vida real e se utilizar de experimentos para cada estudo seria economicamente inviável.

Do cenário descrito acima fica clara a vantagem de se utilizar de métodos numéricos para a resolução de tais equações. Diversos métodos existem para a resolução de variados tipos de fenômenos, mas essa seção foca somente no método de elementos finitos (MEF), pois é o método utilizado para a discretização espacial no estudo descrito no relatório.

Segundo Lewis (LEWIS; NITHIARASU; SEETHARAMU, 2000), o método foi inicialmente desenvolvido para o estudo de tensão em estruturas complexas de fuselagem de avião. O método considera que o domínio do sistema pode ser dividido em múltiplas regiões pequenas e interconectadas nas quais podemos aproximar os comportamentos das variáveis para funções de forma conhecida (linear ou quadrático, por exemplo). O método reduz então um problema contínuo que possui infinitas variáveis para um problema com um número finito de pontos chamados *nódulos*. Um conjunto de *nódulos* forma um *elemento* e o domínio é delimitado por uma região determinada por *contorno*. Essas definições estão a mostra na figura 5.

Em Lewis (LEWIS; NITHIARASU; SEETHARAMU, 2000) é indicado um passo a passo para a utilização do método de elementos finitos:

1. **Discretização do domínio contínuo**: o domínio é dividido em múltiplos elementos interconectadas e não sobrepostos. A discretização pode ser feita diferentes formatos. Possíveis exemplos são triângulos e quadriláteros para domínios em duas dimensões.

2. **Seleção das funções de interpolação ou forma**: essas funções definem como varia a função de campo em um elemento.

3. **Formulação das equações dos elementos**: determinação das matrizes de equação $[K_e]$ e os vetores de carga f_e.

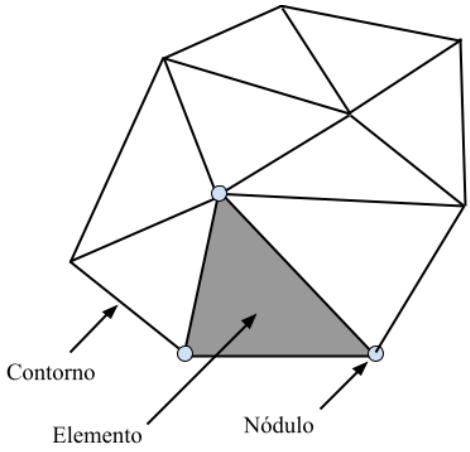


Figura 5 – Malha típica para o Método de Elementos finitos

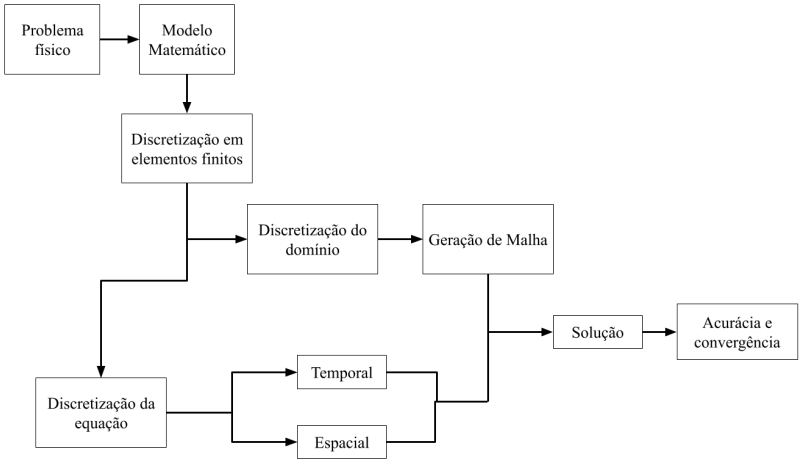


Figura 6 – Fluxograma para a resolução de um problema físico usando o MEF

4. **Montagem das equações dos elementos**: esse passo permite a determinação de um sistema linear (da forma da equação 2.4) a ser montado de acordo com as equações individuais de cada elemento, podendo assim representar o comportamento global do sistema.

5. **Resolução do sistema linear**: o sistema linear resultante da forma da equação 2.4 pode ser resolvido para obter os valores do campo de interesse (temperatura, por exemplo) nos nódulos.

6. **Cálculo de quantidades secundárias**: dado o valor do campo nos nódulos, é possível realizar o cálculo de quantidades secundárias. Por exemplo, tendo o valor do campo temperatura num determinado domínio, é possivel determinar o vetor gradiente no mesmo domínio.

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

2.3 MÉTODO DAS DIFERENÇA FINITAS

Conforme dito na seção 2.2, o método de elementos finitos é utilizado para a resolução espacial dos problema descrito. Porém, para a resolução temporal, o método utilizado será o método de diferenças finitas (MDF). Segundo LeVeque (LEVEQUE, 2004), o método consiste em aproximar derivadas contínuas (equação 2.5) para derivadas numéricas (equação 2.6).

$$\frac{du}{dx} = \lim_{h \to 0} \frac{u(x+h) - u(x)}{h} \tag{2.5}$$

$$D_+ u(x) \equiv \frac{u(x+h) - u(x)}{h} \tag{2.6}$$

A hipótese principal do método das diferenças finitas é que para um valor de h suficientemente pequeno, a equação 2.7 vale.

$$D_+ u(x) \approx \frac{du}{dx} \tag{2.7}$$

Assim como no método de elementos finitos (ver 2.2), o objetivo do método das diferenças finitas é transformar o problema físico contínuo em um problema matemático discreto que pode ser resolvido pela forma de um sistema linear.

2.4 CONDIÇÃO DE CONTORNO

Segundo Lewis (LEWIS; NITHIARASU; SEETHARAMU, 2000), três tipos de condições de contorno podem ser definidos:

a) Condição de Contorno de Dirichlet:
   Para uma dada região de contorno Γ_D, a condição de contorno de Dirichlet prescreve o valor de cada ponto $p \in \Gamma_D$. A definição matemática pode ser escrita como na equação 2.8. Esta formulação é comumente utilizada para contornos nos quais a temperatura é considerada fixa.

$$T(p) = T_0 \; para \; p \in \Gamma_D \tag{2.8}$$

b) Condição de Contorno de Neumann:
   Para uma dada região de contorno Γ_N, a condição de contorno de Neumann prescreve o valor do produto escalar entre o vetor gradiente de temperatura e o vetor área normal da superfície de cada ponto determinada $p \in \Gamma_N$. A definição matemática pode ser escrita como na equação 2.9.

$$\frac{\partial T(p)}{\partial n} = C_0 \; para \; p \in \Gamma_N \tag{2.9}$$

A formulação da condição de contorno de Neumann é comumente utilizada para contornos nos quais o fluxo de calor é considerado fixada, dada a definição do fluxo de calor numa determinada direção n segundo a equação 2.10

$$q_n(p) = -\kappa \frac{\partial T(p)}{\partial n} = -\kappa C_0 \; para \; p \in \Gamma_N \tag{2.10}$$

c) Condição de Contorno de Robin:
Para uma dada região de contorno Γ_R, a condição de contorno de Robin prescreve o valor de uma operação linear entre a derivada e o valor da função de cada ponto determinada $p \in \Gamma_N$ numa determinada direção n. A definição matemática pode ser escrita segundo a equação 2.11.

$$a_0 \frac{\partial T(p)}{\partial n} + b_0 T(p) = C_0 \; para \; p \in \Gamma_N \tag{2.11}$$

A formulação da condição de contorno de Robin é comumente utilizada para contornos nos quais a transferência de calor é dada por convecção, dada a definição do numa determinada direção n segundo a equação 2.12.

$$\dot{Q} = mc_p \frac{\partial T(p)}{\partial t} = hA(T_{ext} - T(p)) \therefore mc_p \frac{\partial T(p)}{\partial t} + hAT(p) = hAT_{ext} \tag{2.12}$$

2.5 POSSÍVEIS MÉTRICAS PARA O ERRO

Para um mesmo resultado numérico determinado, existem múltiplas métricas para a determinação da acurácia desse calculo em relação a uma referência. Para as equações 2.13, 2.14 e 2.15, serão usados T_{num} e T_{ref} como as estruturas de dados que armazenam respectivamente a solução numérica e a solução de referência e Ω como o domínio do problema.

a) Erro Médio Absoluto (EMA):

$$EMA(T_{num}, T_{ref}) = \frac{\sum\limits_{i \in \Omega} |T_{num,i} - T_{ref,i}|}{N_\Omega} \tag{2.13}$$

b) Raíz do Erro Quadrático Médio (REQM):

$$REQM(T_{num}, T_{ref}) = \sqrt{\frac{\sum\limits_{i \in \Omega} (T_{num,i} - T_{ref,i})^2}{N_\Omega}} \tag{2.14}$$

c) Erro Máximo(EM):

$$EM(T_{num}, T_{ref}) = max|T_{num,i} - T_{ref,i}|_{i\in\Omega} \tag{2.15}$$

2.6 PROCESSADOR

O processador (em inglês CPU - *Central Processing Unit*) é o dispositivo eletrônico responsável pelo controle de todos os outros componentes do computador. Segundo Marcelo Schuh (SCHUH, 2017), um processador pode ser dividido nas seguintes partes (ver figura 7):

1. Die: componente no qual efetivamente ocorre o processamento da lógica computacional. Numa análise térmica, também é considerado o componente no qual se gera calor

2. PCB (*Printed Circuit Board*): componente responsável pela ligação do Die e a Placa Mãe conectando o processador ao resto da máquina

3. IHS (*Integraded Heat Spreader*): é a tampa metálica responsável por conduzir o calor do núcleo para o dissipador diminuindo assim a temperatura do núcelo. Tem como função secundária proteger mecanicamente o núcleo.

4. TIM (*Thermal Interface Material*): material intersticial entre o IHS e o Die e o Dissipador de Calor e IHS (ver figura 8)

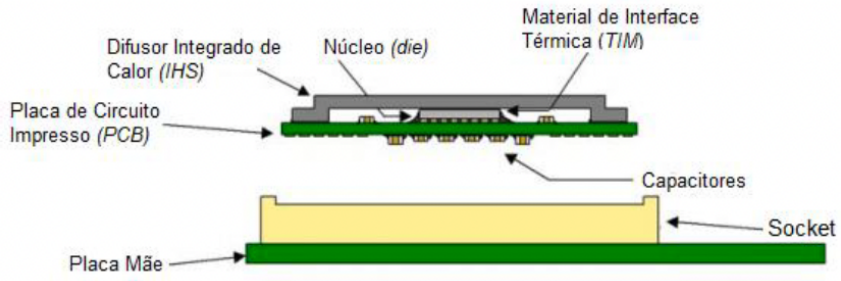


Figura 7 – Esquemático de um processador típico

Fonte: (SCHUH, 2017)

Associado ao dissipador de calor está um ventilador. Este dispositivo permite que as trocas por convecção no dissipador sejam feitas no regime de convecção forçada, aumentando assim o fluxo total de trocas e permitindo assim um arrefecimento mais eficiente.

Os diferentes processadores disponíveis no mercado variam principalmente em termos de arquitetura do processador. Essa arquitetura indica a composição do processador

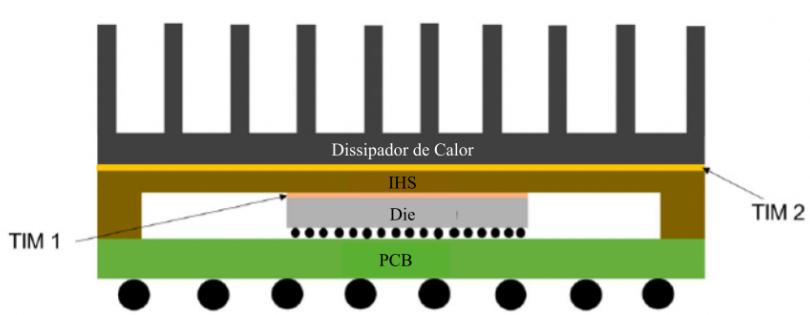


Figura 8 – Esquemático para a localização do TIM

Fonte: (CHOWDHURY et al., 2021)

(número de núcleos, tamanho da GPU, etc). Por exemplo, um número maior de núcleos permite um maior número de operações simultâneas executadas pelo processador. O Die utilizado nesse trabalho para a análise térmica tem a arquitetura Coffee Lake, apresentada na figura 9.

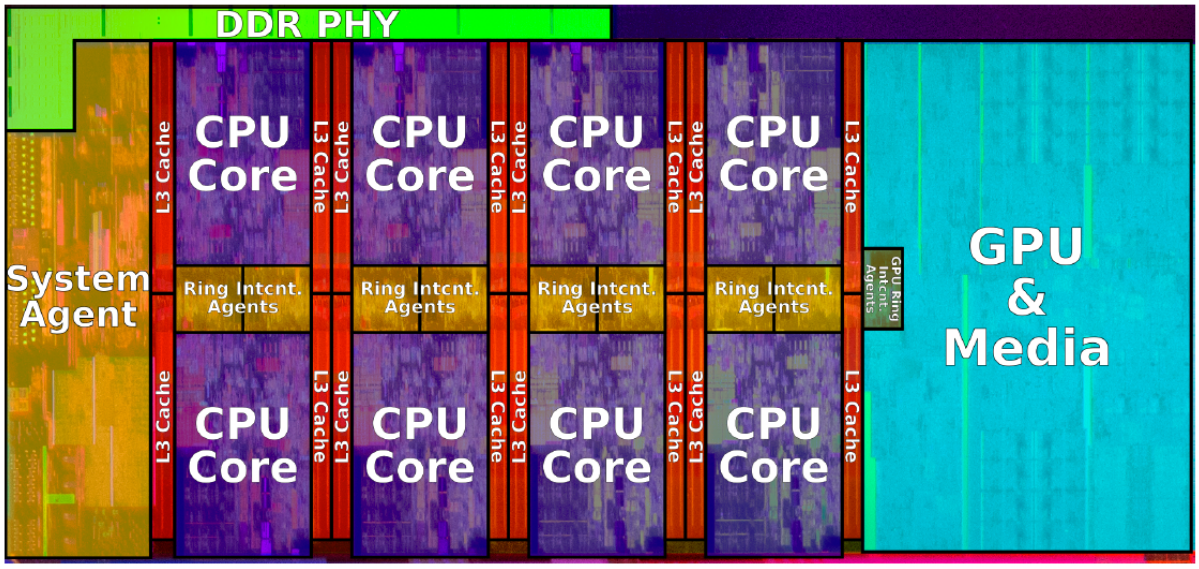


Figura 9 – Arquitetura do Die Coffee Lake

Fonte: (INTEL, 2022b)

2.7 ESTIMATIVA DO COEFICIENTE DE TRANSFERÊNCIA DE CALOR CONVECTIVA

Considerando um trocador de calor com as dimensões expressas na figura 10, existem aproximações analíticas para calcular o coeficiente h de troca de calor convectiva a partir

de modelos do número de Nusselt. Vale lembrar que o número de Nusselt é o número adimensional que indica a relação entre as trocas convectivas de um fluido em movimento com um objeto parado e as trocas advectivas internas ao fluido. A relação entre o número de Nusselt e o coeficiente h no dissipador de calor está expressa na equação 2.16.

$$Nu = \frac{hL}{k_{ar}} \tag{2.16}$$

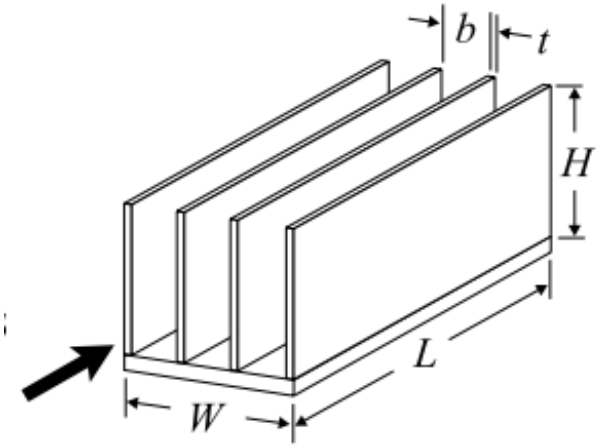


Figura 10 – Dimensões de um dissipador de calor escritas de maneira literal

P. Teertstra (TEERTSTRA; YOVANOVICH; CULHAM, 1999) afirma que o coeficiente de Nusselt ideal de um dissipador de calor pode ser calculado a partir do número de Reynolds do escoamento e do número de Prandtl do fluido segundo a equação 2.17.

$$Nu_i = \left[\left(\frac{Re_b^* Pr}{2} \right)^{-3} + \left(0.664 \sqrt{Re_b^*} Pr^{1/3} \left(1 + \frac{3.65}{\sqrt{Re_b^*}} \right)^{1/2} \right)^{-3} \right]^{-1/3} \tag{2.17}$$

Vale lembrar que o número de Prandtl é definido (OZISIK, 1985) como a relação entre a difusividade de momento e a difusividade térmica segundo a equação 2.18.

$$Pr = \frac{c_p \mu}{k} \tag{2.18}$$

O coeficiente de Nusselt também pode ser corrigido se utilizando de parâmetros geométricos do dissipador e de parâmetros materiais do fluido e do dissipador segundo a equação 2.19

$$Nu_b = Nu_i \frac{tanh \sqrt{2Nu_i \frac{k_f}{k} \frac{H}{b} \frac{H}{t} \left(\frac{t}{L} + 1 \right)}}{\sqrt{2Nu_i \frac{k_f}{k} \frac{H}{b} \frac{H}{t} \left(\frac{t}{L} + 1 \right)}} \tag{2.19}$$

3 METODOLOGIA

3.1 DESCRIÇÃO DO DOMÍNIO

Para a modelagem do problema físico de interesse, é preciso inicialmente definir quais elementos vão ser modelados para a solução por elementos finitos. Para a análise descrita nesse trabalho, serão modelados o PCB, o Die, o ar ao redor do Die e o IHS diretamente acoplado ao dissipador de calor segundo a imagem 11.

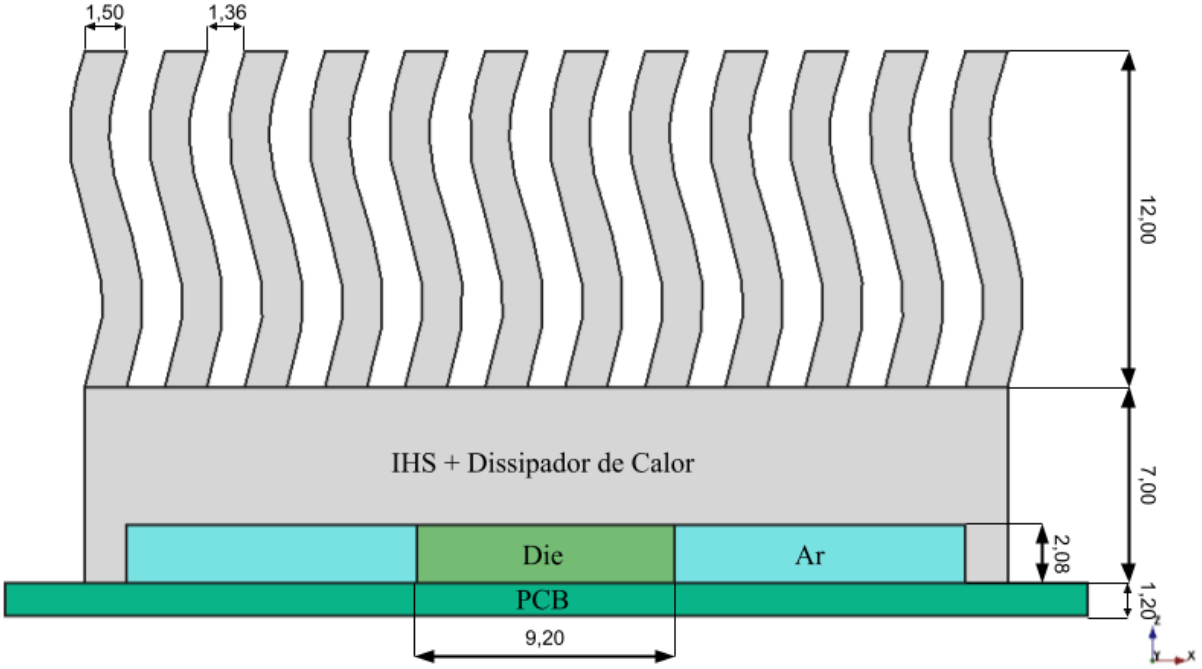


Figura 11 – Esquemático do domínio de simulação (todas as medidas estão em milímetros)

3.2 HIPÓTESES SIMPLIFICADORAS

Sendo um sistema altamente complexo, são necessárias certas hipóteses simplificadoras para a modelagem do sistema em elementos finitos. As hipóteses tomadas foram as seguintes:

- O sistema pode ser descrito de maneira satisfatoria segundo o esquemático da imagem 11
- As transferências de calor para as partes abaixo do PCB (seguindo a orientação da imagem 11) são desprezíveis em relação àquelas que acontecem no conjunto IHS/Disspador de Calor

- Entre os sistemas Die/Ar/PCB/IHS, as trocas de calor são somente condutivas, podendo assim desprezar as trocas radiativas e as possíveis trocas condutivas e advectivas do ar dentro do IHS (consideramos o ar parado e sem grande gradiente de temperatura)
- As trocas convectivas se dão somente nas partes internas das aletas do dissipador de calor (mostrado em mais detalhes na seção 3.7)
- As trocas convectivas mencionadas acima são regidas por um coeficiente de troca térmica convectiva h constante no contorno descrito
- O ar que realiza as trocas convectivas nas aletas do dissipador de calor possui temperatura uniforme de $25°C$
- A fonte de calor do sistema é considerada constante e uniforme no Die
- O ar entre o IHS e o PCB é considerado isolante, visto que sua condutividade térmica representa 2,4 % da condutividade térmica do IHS
- O contato entre o Die e o IHS e entre o IHS e o Dissipador de calor são considerados perfeitos

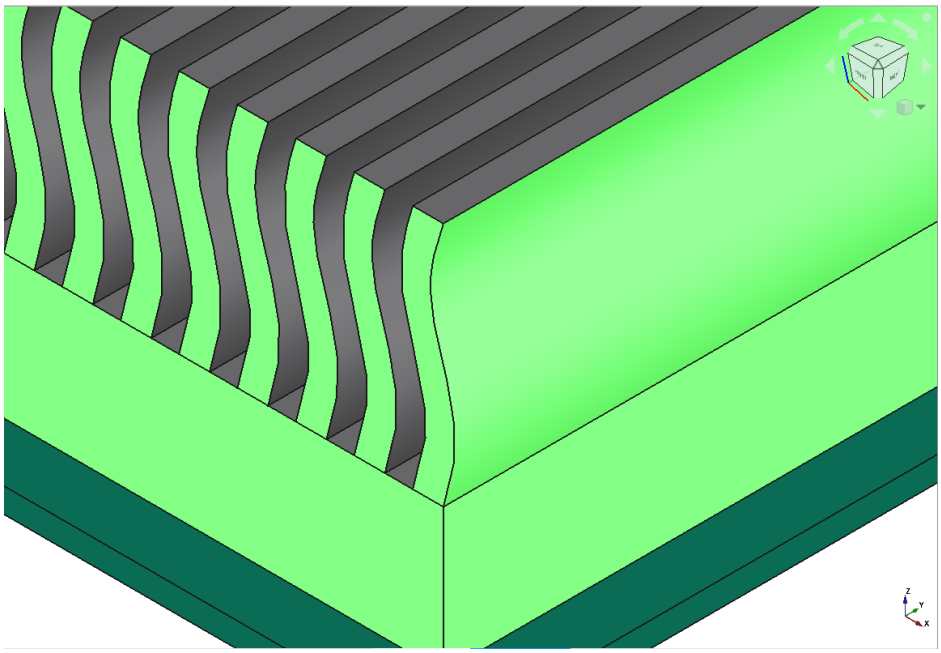


Figura 12 – As regiões selecionadas (em verde) são as regiõe do contorno do IHS que possuem condição de contorno do tipo Neumann

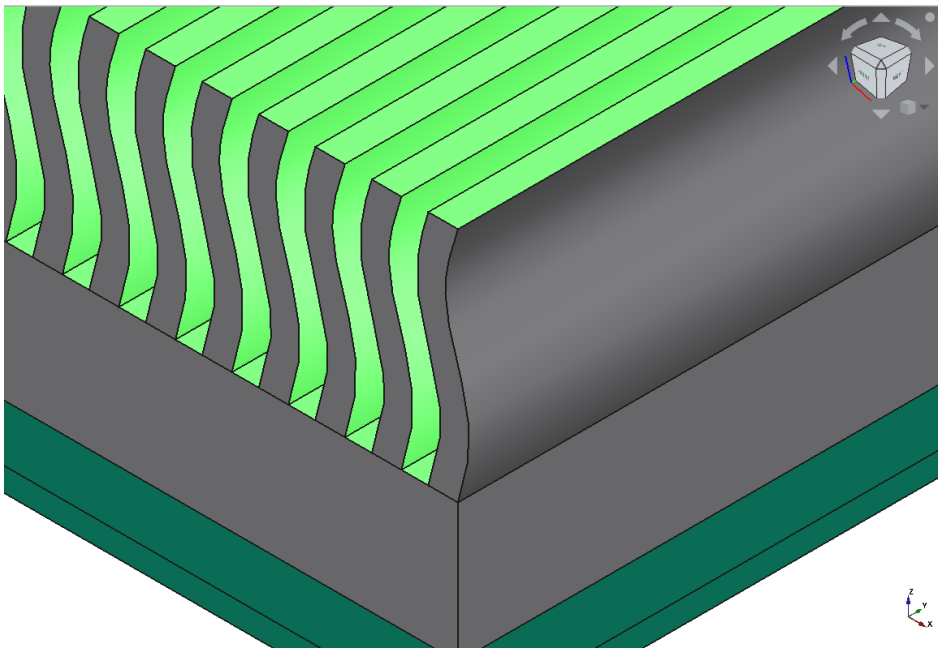


Figura 13 – As regiões selecionadas (em verde) são as regiõe do contorno do IHS que possuem condição de contorno do tipo Robin

3.3 FORMULAÇÃO NUMÉRICA DO PROBLEMA

Seguindo as hipóteses listadas na seção 3.2, o sistema de interesse será modelado só com transferências de calor condutivas internas e trocas convectivas em certas regiões do contorno. Sendo assim, a equação que rege o fenômeno no domínio é a equação 3.1 onde α, o coeficiente de difusividade térmica, está definido na equação 3.2.

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

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

Uma das operações realizadas ao se utilizar o método de elementos finitos é a conversão da equação da equação de sua forma forte (equação 3.1) para a sua forma fraca (equação 3.3). O lema fundamental do cálculo variacional (LEITAO, 2001) diz que uma solução que satisfaz a forma fraca da equação tem que satisfazer sua forma forte.

$$\int_\Omega \omega \left(\frac{\partial T}{\partial t} - \alpha \nabla^2 T - \frac{Q}{\rho c_v} \right) d\Omega = 0 \tag{3.3}$$

Integrando por partes a parte da equação que contém o operador laplaciano, chegamos a equação 3.4, onde Γ representa o contorno do domínio. Essa nova equação implica na

$$\int_\Omega \omega \alpha \nabla^2 T d\Omega = \int_\Gamma \omega \alpha \nabla T d\Gamma - \int_\Omega \alpha \nabla T \nabla w d\Omega \tag{3.4}$$

$$\int_{\Omega} \frac{\partial T}{\partial t} d\Omega - \int_{\Gamma} \omega \alpha \nabla T d\Gamma + \int_{\Omega} \alpha \nabla T \nabla w d\Omega - \int_{\Omega} \omega \frac{Q}{\rho c_v} d\Omega = 0 \tag{3.5}$$

No método implementado ao decorrer deste relatório foi utilizado o método de Galerkin, o que implica nas equações 3.6, 3.7 e 3.8. Nessas equações, T_i, ω_i e Q_i representam respectivamente os valores de temperatura, função peso e geração de calor no nódulo i.

$$T(x, y, z, t) \approx \sum N_i(x, y, z) T_i(t) \tag{3.6}$$

$$w(x, y, z, t) \approx \sum N_i(x, y, z) \omega_i \tag{3.7}$$

$$Q(x, y, z, t) \approx \sum N_i(x, y, z) Q_i(t) \tag{3.8}$$

Considerando as equações 3.6, 3.7 e 3.8, podemos escrever as equações 3.9 e 3.10, onde Ω_e representa o domínio referente ao elemento e.

$$\int_{\Omega} \omega \frac{\partial T}{\partial t} d\Omega \approx \sum_i \sum_j \frac{\partial T_i}{\partial t} \int_{\Omega_e} N_i N_j d\Omega \tag{3.9}$$

$$\int_{\Omega} \omega Q d\Omega \approx \sum_i \sum_j Q_i \int_{\Omega_e} N_i N_j d\Omega \tag{3.10}$$

Os termos $\int_{\Omega_e} N_i N_j d\Omega$ presentes nas equações 3.9 e 3.10 resultam no que é reconhecido como matriz de massa $M_{i,j}$.

Analisando os termos relativos ao laplaciano da temperatura, temos a equação 3.10.

$$\int_{\Gamma} \omega \alpha \nabla T d\Gamma - \int_{\Omega} \alpha \nabla T \nabla \omega d\Omega \approx \sum_i \sum_j \alpha \omega_j T_i \left(\int_{\Gamma_e} N_j \nabla N_i d\Gamma - \int_{\Omega_e} \nabla N_i \nabla N_j d\Omega \right) \tag{3.11}$$

O termo entre parênteses da equação 3.11 resulta na matriz de rigidez $K_{i,j}$. Segundo o lema fundamental do cálculo variacional, o termo ω pode ser retirado da equação. Considerando então as equações precedentes e a equação 3.12, podemos então escrever as equações 3.13 e 3.14.

$$T_i = \beta T_i^n + (1 - \beta) T_i^{n+1} \tag{3.12}$$

$$M_{i,j} \frac{(T_i^{n+1} - T_i^n)}{\Delta t} - K_{i,j} T_i - M_{i,j} \frac{Q_i}{\rho c_v} = 0 \tag{3.13}$$

$$\left(\frac{M_{i,j}}{\Delta t} - \beta K_{i,j}\right) T_i^{n+1} = \left(\frac{M_{i,j}}{\Delta t} + (1-\beta) K_{i,j}\right) T_i^n + M_{i,j} \frac{Q}{\rho c_v} \tag{3.14}$$

O parâmetro β na equação 3.12 define o método de discretização temporal que será utilizado: se $\beta = 0$, o método utilizado é o implícito; se $\beta = 0,5$, está sendo utilizado método de Crank-Nicholson; e se $\beta = 1$, o método em questão é o explícito.

As matrizes presentes na equação 3.14 são feitas a partir da montagem de matrizes elementares k^e e m^e apresentadas nas equações 3.15 e 3.16 para um tetraedro de volume V como o da figura 14.

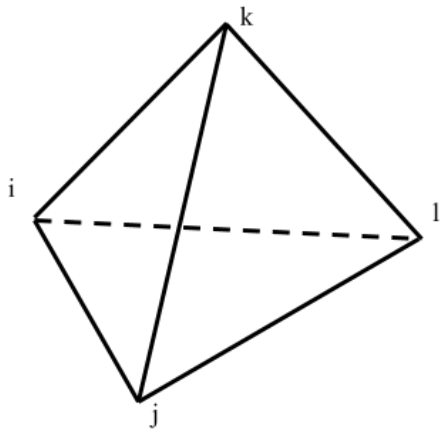


Figura 14 – Tetraedro de vértices i, j, k *e* l

$$m^e = \frac{1}{6V} \begin{bmatrix} 2 & 1 & 1 & 1 \\ 1 & 2 & 1 & 1 \\ 1 & 1 & 2 & 1 \\ 1 & 1 & 1 & 2 \end{bmatrix} \tag{3.15}$$

$$k^e = V \boldsymbol{B}^T \boldsymbol{B} \tag{3.16}$$

Onde B pode ser escrito na forma da equação 3.17, onde os parâmetros b, c *e* d são determinados a partir da equação 3.18. O volume do tetraedro pode ser determinado a partir da equação 3.19

$$B = \frac{1}{6V} \begin{bmatrix} b_i & b_j & b_k & b_l \\ c_i & c_j & c_k & c_l \\ d_i & d_j & d_k & d_l \end{bmatrix} \tag{3.17}$$

$$(p,q) \in [i,j,k,l]^2 \begin{cases} a_p + b_p x_q + c_p y_q + d_p z_q = 1, se\ p = q \\ a_p + b_p x_q + c_p y_q + d_p z_q = 0, se\ p \neq q \end{cases} \tag{3.18}$$

$$V = \frac{1}{6} det \begin{pmatrix} 1 & x_i & y_i & z_i \\ 1 & x_j & y_j & z_j \\ 1 & x_k & y_k & z_k \\ 1 & x_l & y_l & z_l \end{pmatrix} \tag{3.19}$$

3.4 PARÂMETROS DE SIMULAÇÃO

Como verificado na seção 3.3, a formulação numérica do problema depende de diversos parâmetros numéricos, sejam eles materiais, térmicos ou geométricos. Essa seção serve para detalhar os valores tomados para cada um desses parâmetros.

3.4.1 Parâmetros Materiais

Os parâmetros materias para cada um dos materiais simulados e do ar seguem na tabela

Material (*Componente*)	Massa específica $[kg/m^3]$	Calor Específico $[J/kg.K]$	Condutividade térmica $[J/m.s.K]$
Silica (*Die*)	2650	712	$1,25$
FR-4 (*PCB*)	1850	1100	$(xy)\ 0,81 \quad (z)0,29$
Cobre (*IHS*)	8960	380	401
Ar	$1,225$	1000	$0,003$

Tabela 1 – Tabela de parâmetros de massa específica, calor específico e condutividade térmica dos materiais usados na modelagem do sistema em questão

Fontes: (ZHU et al., 2018), (NWES, 2020)

3.4.2 Coeficiente de troca térmica convectiva

A primeira coisa a se fazer para a determinação do coeficiente de troca térmica convectiva é estabelecer um ponto de operação. Segundo o manual (WAKEFIELD-VETTE,), a velocidade do ar em um dissipador de calor costuma variar de $100 ft/minuto$ a $1000 ft/minuto$. Fazendo a conversão para o sistema métrico, temos $0,5 m/s$ a $5 m/s$ de velocidade do ar entre as aletas do dissipador. Para a análise desse relatório, será utilizado um valor padrão de $3 m/s$ para essa velocidade. A fórmula que permite calcular o número de Reynolds está na equação 3.20.

$$Re = \frac{Ub}{\nu} \tag{3.20}$$

Considerando a velocidade citada anteriormente, o valor de $b = 1,44mm$ e a viscosidade cinemática do ar como $\nu = 1,48.10^{-5}m^2/s$, chegamos ao valor de numéro de Reynolds de $Re = 291,89$, que comparando aos resultados de transição laminar/turbulenta para dutos, seria considerado um escoamento laminar.

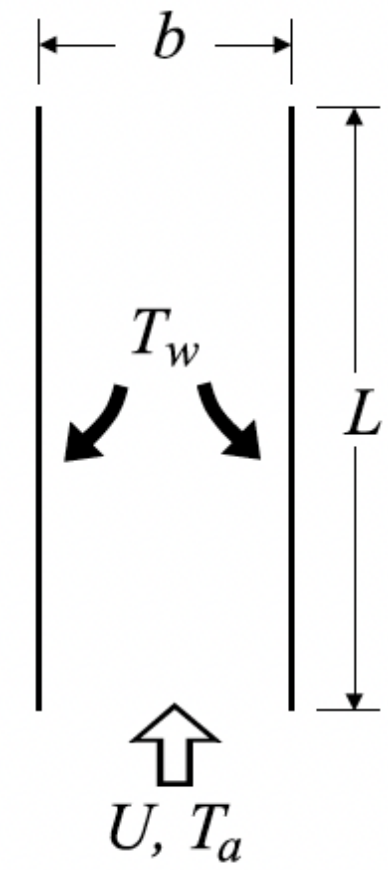


Figura 15 – Esquemático do escoamento do ar frio ao redor das aletas quentes do dissipador de calor

Para esse valor de Reynolds, podemos então calcular o número de Nusselt ideal pela fórmula na equação 2.17, chegando ao resultado de $Nu_i = 11,14$ e $Nu_b = 10,87$, chegando a um coeficiente de troca térmica convectiva de $h = 8,65W/m^2.K$

3.4.3 Parâmetros Geométricos

Esse estudo paramétrico se dá sobre duas variáveis geométricas das aletas: n e A. Esses dois parâmetros controlam a forma da linha neutra da aleta a partir da equação

$$x = H.A.sin\Big(\frac{n\pi z}{H}\Big); z \in [0, H] \tag{3.21}$$

O estudo paramétrico será do tipo varredura, ou seja, vão ser avaliados pares ordenados (n, A) segundo a figura 17.

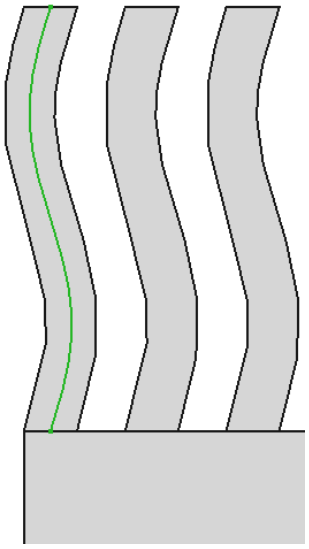


Figura 16 – Linha neutra (em verde) da aleta do dissipador de calor

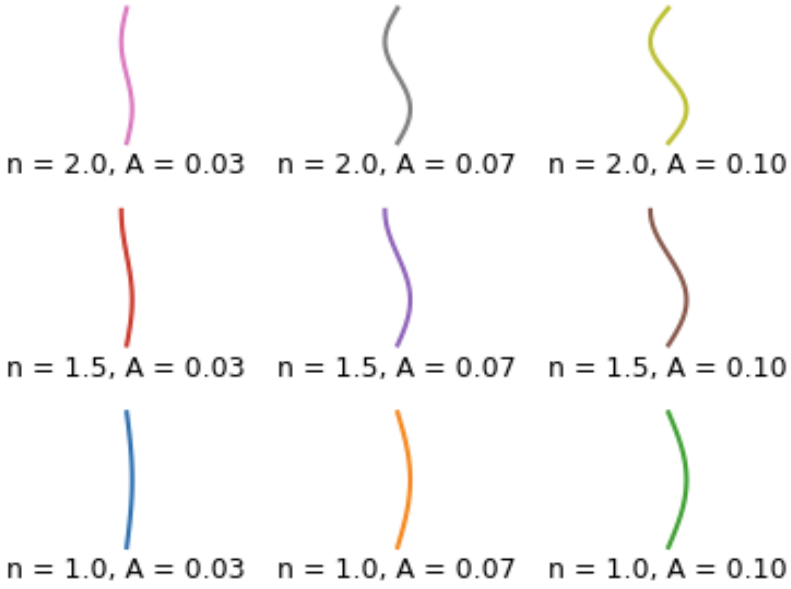


Figura 17 – Linhas neutras das aletas para cada par ordenado n, A para o estudo paramétrico do tipo varredura

3.5 CONTROLE DE VERSÃO

Para a organização do trabalho de modelagem/pré-processamento/simulação, foi necessária a criação de uma tabela de controle de versões. Essa tabela organiza todos os pares ordenados em formas de índice para futura referência. A tabela 2 descreve o índice correspondente de cada par ordenador (n, A) e dá como informação adicional o volume e a área de troca convectiva do dissipador de calor representado pela malha tetraédrica. Essas informações serão utilizadas na seção 5.2.

Número de versão	n	A	Volume do dissipador $[mm^3]$	Área de troca do dissipador $[mm^2]$
0	0.0	0.00	$1,286.10^4$	$2,494.10^2$
1	1.0	0.03	$1,284.10^4$	$2,501.10^2$
2	1.0	0.07	$1,281.10^4$	$2,524.10^2$
3	1.0	0.10	$1,269.10^4$	$2,581.10^2$
4	1.5	0.03	$1,282.10^4$	$2,502.10^2$
5	1.5	0.07	$1,276.10^4$	$2,521.10^2$
6	1.5	0.10	$1,143.10^4$	$2,521.10^2$
7	2.0	0.03	$1,279.10^4$	$2,519.10^2$
8	2.0	0.07	$1,269.10^4$	$2,566.10^2$
9	2.0	0.10	$1,141.10^4$	$2,219.10^2$

Tabela 2 – Tabela para a organização dos pares ordenados em forma de índice. Como informação adicional temos o volume e a área de troca do dissipador representado pela malha.

3.6 CONFIGURAÇÃO DO PROBLEMA

A configuração do problema se dá por meio da edição de três arquivos externos ao código *python*:

1. **Arquivo de malha**: o arquivo de malha é gerado pelo *software Gmsh* a partir do modelo *CAD* gerado pelo *FreeCAD*. Para as simulações de dissipador de calor, a malha utilizada é de refino 1 (ver seção 4.1). Os volumes e regiões de contorno têm de ser nomeados de maneira única para futura identificação pelo código usando a função *physical group name*.

2. **Base de dados materiais**: o arquivo *json* foi criado para armazenar os dados materiais como massa específica, condutividade térmica e calor específico para cada material utilizado na modelagem do sistema. Vale notar que o modelo utilizado para a condutividade térmica é ortotrópico, logo cada material recebe três valores numéricos para a configuração da condutividade térmica para os eixos x, y e z.

3. **Condições de simulação**: seguindo os nomes configurados no arquivo de malha, é criado um arquivo *json* para a configuração das condições de simulação para cada uma das regiões nomeadas. O arquivo é dividido em domínio e contorno e as configurações de condição de simulação são de fonte de calor para as regiões de domínio e variam para as regiões de contorno segundo o tipo de condição de contorno (ver seção 3.7).

3.7 CONDIÇÕES DE CONTORNO

As condições de contorno para o sistema divididos por componente são:

- PCB:
    1. condição de contorno de Neumann para todos os contornos, com prescrição de $q = 0$.
    2. Não há geração de calor dentro do domínio.
- Die:
    1. condição de contorno de Neumann para todos os contornos, com prescrição de $q = 0$.
    2. O domínio inteiro recebe uma fonte de calor que soma 65 W homogeneamente distribuida (FERREIRA, 2022).
- IHS:
    1. condição de contorno de Neumann para os contornos externos (regiões selecionadas na figura 12), com prescrição de $q = 0$.
    2. condição de contorno de Robin para os contornos internos das aleras (regiões selecionadas na figura 13), com prescrição de transferência de calor por convecção com $h = 8,65 W/m^2.K$ e $T_{\text{inf}} = 25^\circ\ C$.
    3. Não há geração de calor dentro do domínio.

3.8 DIMENSÕES DO MODELO

A figura 18 mostra as cotas do modelo de dissipador de calor (todas as medidas estão em mm), os valores vêm do modelo definido por (FERREIRA, 2022).

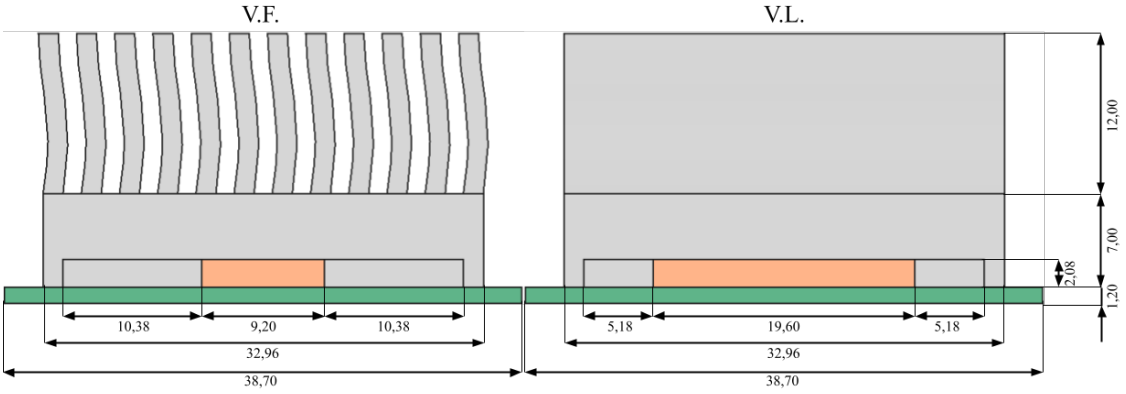


Figura 18 – Cotas em mm do modelo utilizado

3.9 ANÁLISE DE MALHA E VALIDAÇÃO DO MODELO

A análise de malha será feita por dois aspectos: a partir de uma validação do método por solução manufaturada num caso simplifcado e a partir de uma análise de convergência de malha no caso de interesse.

3.9.1 Validação por Solução Manufaturada

O princípio do método de solução manufaturada é de atribuir uma função arbitrária à função $T(x, y, z, t)$, substituir esse valor na equação a ser resolvida e descobrir a função de controle que satisfaz a equação. No caso da lei de Fourier o princípio é achar a função Q que satisfaz a equação 3.22.

$$Q_{manu} = \rho c_v \left(\frac{\partial T_{manu}}{\partial t} - \alpha \nabla^2 T_{manu} \right) \tag{3.22}$$

É imprescindível para a validação do código por solução manufaturada que as condições de contorno sejam também impostas de acordo com a função T_{manu}, ou seja, T_{manu} deve ser implementado nas equações de contorno que podem ser do tipo Dirichlet, Neumann ou Robin (ver equações 2.8, 2.9 e 2.11)

Após definida a função Q_{manu}, a equação deve ser resolvida pelo método de preferência (no caso desse relatório se utiliza o MEF) para achar uma solução numérica T_{num}. Para calcular a acurácia do resultado, foi utilizado como métrica o erro quadrático médio (EQM), definido na equação 2.14.

3.9.2 Análise de convergência de malha

O princípio da análise de convergência de malha é partir de uma malha mais grosseira, e refiná-la em passos consecutivos e analisar a mudança relativa que ocorreu em determinada métrica ao refinar. Logo, se definirmos como T^n_{case} e $T^n_{Dia,max}$ como as temperaturas de case e máxima do Die respectivamente no passo de tempo n, definimos a mudança relativa devida ao refino da malha na equação 3.23.

$$\Delta T^n_{case} = \frac{T^n_{case} - T^{n-1}_{case}}{T^n_{case}} \tag{3.23}$$

3.10 ANÁLISE DO RESULTADO

A análise do resultado se dá a partir de duas medidas que serão determinadas para cada passo de tempo da solução da equação: a temperatura média e a temperatura máxima no Die, T_{max} e T_{avg}, respectivamente. Como referência, será tomada a recomendação que a temperatura no Die não exceda $100°C$ (INTEL, 2022a).

4 RESULTADOS E DISCUSSÃO

4.1 VALIDAÇÃO POR SOLUÇÃO MANUFATURADA

Para a validação por solução manufaturada, a função escolhida está descrita na equação 4.1. Seguindo a equação 3.22, podemos determinar a função de Q que satisfaz a equação que rege o fenômeno. A solução para Q está descrita na equação 4.2, supondo um material para o qual todos os parâmetros materiais ρ, c_v e k são iguais a 1, cada um em sua respectiva unidade no SI.

$$T_{manu} = (x^2 + y^2 + z^2)e^{-t} \tag{4.1}$$

$$Q_{manu} = -(x^2 + y^2 + z^2 + 6)e^{-t} \tag{4.2}$$

Em seguida definimos um cubo de aresta $1m$ como domínio para a simulação numérica. Utilizando o *software GMSH*, é possível criar a malha para este cubo. A malha original (será referenciada a partir desse ponto como malha de refino 0) possui 273 nódulos e 622 elementos entre elementos tetraédricos e triangulares.

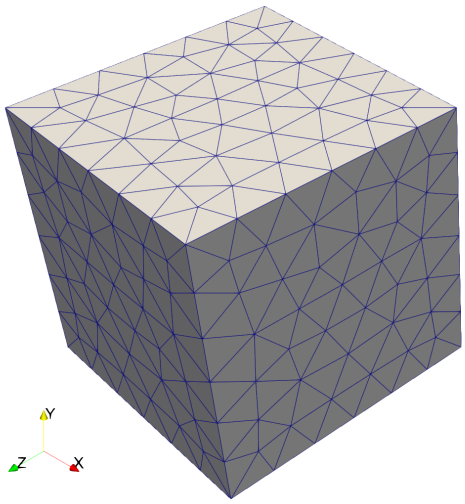


Figura 19 – Modelo numérico do cubo com 273 nódulos (malha de refino 0)

O *software* possui a opção *Refine By Splitting* que define um novo nódulo no ponto médio entre dois pontos vizinhos. Repetindo esse comando para todo o domínio, o comando gera então uma malha com elementos de tamanho igual à metade do elemento original. Este procedimento está explicado visualmente na figura 20. Para os propósitos desta seção, definiremos a malha original com a malha de refino 0 e a cada vez que utilizarmos o comando de refino esse nível de refino aumenta de 1.

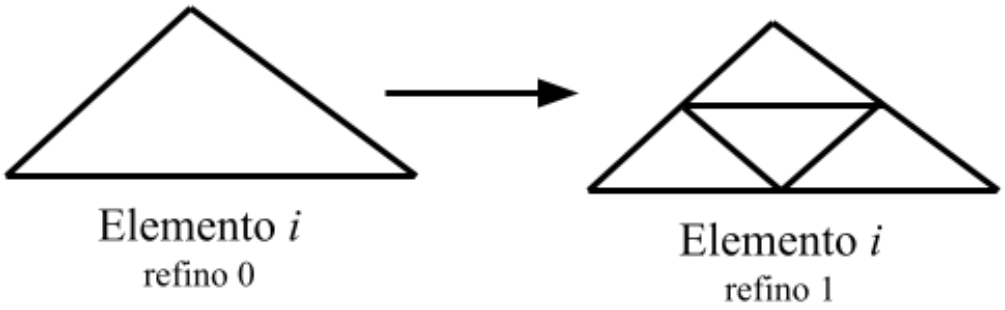


Figura 20 – Explicação visual do comando de refino (*refine by splitting*)

É possível então escrever uma rotina em *python* para repetir a análise para cada nível de refino da malha. Ao fazer essa análise, é possível se deparar com a primeira limitação do projeto (que serão descritas em detalhe na seção 5.4): o computador utilizado para as simulações numéricas é um computador pessoal e não conseguiu realizar a análise de malha para malhas de refino superior a 2.

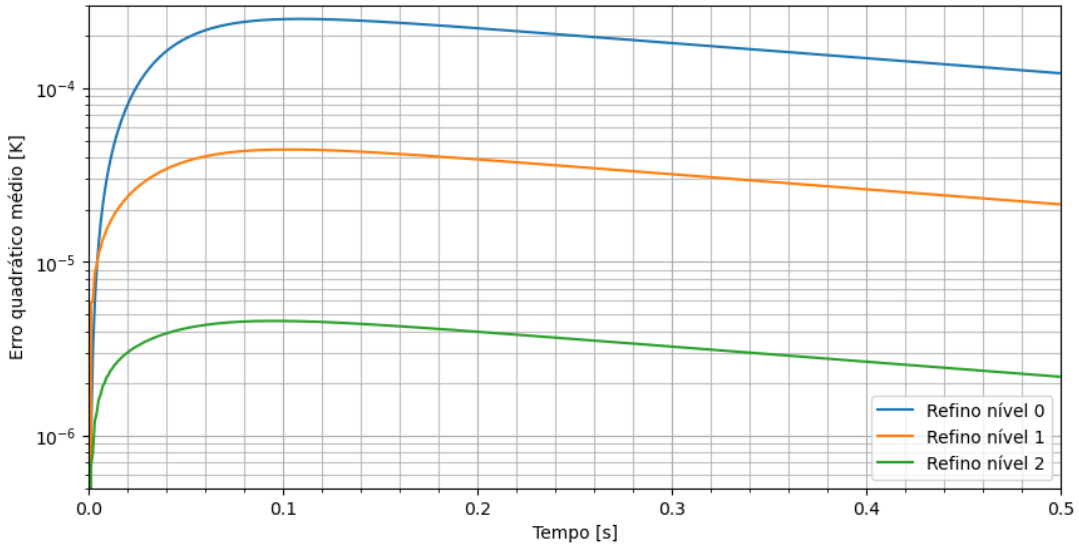


Figura 21 – Resultado do erro quadrático médio por passo de tempo em relação à solução manufaturada

A figura 21 mostra a evolução do erro quadrático médio por passo de tempo em relação à solução manufaturada dividido por diferentes níveis de refino da malha. É possível ver diretamente que, como se pode esperar, o refino da malha gera resultados com níveis mais baixos de erro consistentemente. Para os níveis de refino 0, 1 e 2 temos que os valores máximos do erro quadrático médio são de $2,5.10^{-4}$, $4,4.10^{-5}$ e $4,6.10^{-6}$ respectivamente.

4.2 ANÁLISE DA EVOLUÇÃO DA TEMPERATURA

Para a avaliação da evolução da temperatura, será usado como referência o elemento 0 (ver tabela 2), o dissipador com aletas retas. A figura 22 mostra a evolução da temperatura máxima e média no *die*.

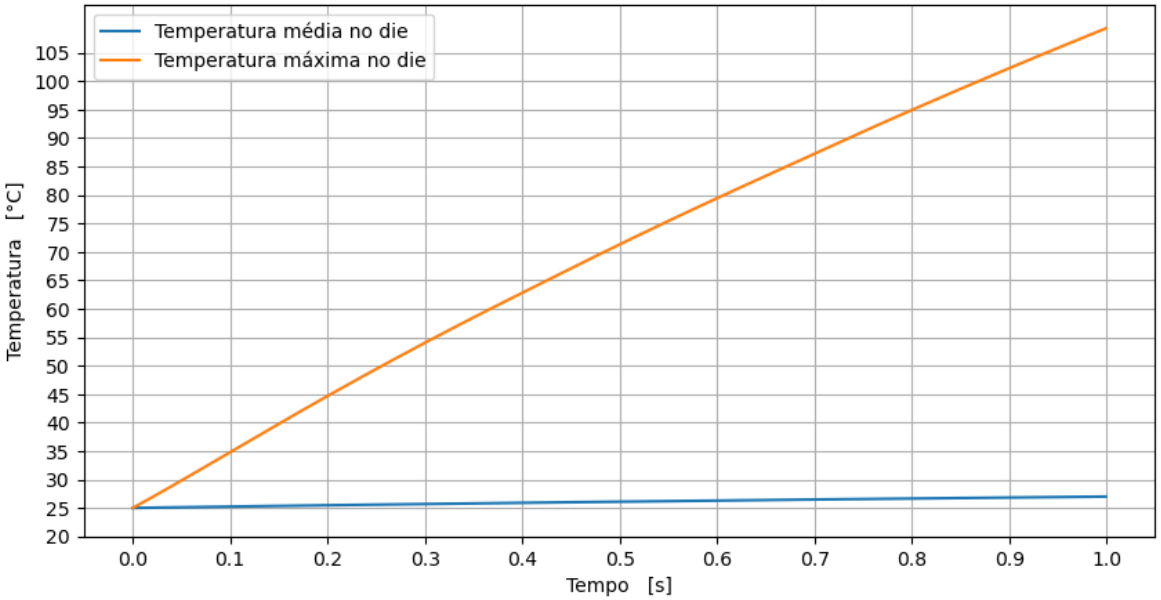


Figura 22 – Evolução da temperatura máxima e média no *die* para o dissipador versão 0 (aletas retas)

Do gráfico 22, é possível chegar a algumas conclusões:

1. Em $t = 1.0\ s$, o *die* ainda não teve tempo para se estabilizar termicamente
2. A variação de temperatura máxima no *die* é muito maior (cerca de 40 vezes) que a variação de temperatura média no mesmo. Isso pode indicar que a temperatura máxima se manifesta em relativamente poucos pontos no domínio no *die*.

Visto que o objetivo primário deste trabalho é de avaliar a influência dos parâmetros geométricos, o fato do *die* não ter estabilizado em temperatura não é um problema. A hipótese aqui tomada é que duas versões diferentes podem ser comparadas a partir da temperatura máxima final do *die* a $t = 1.0\ s$ e que quanto mais performante o dissipador, menor será essa temperatura.

4.3 ANÁLISE DA INFLUÊNCIA DOS PARÂMETROS GEOMÉTRICOS NA TRANSFERÊNCIA DE CALOR

Como determinado na seção anterior, as versões do dissipador determinados pelos pares ordenados (n, A) (ver tabela 2) serão comparados a partir da temperatura máxima do *die* a $t = 1.0\ s$.

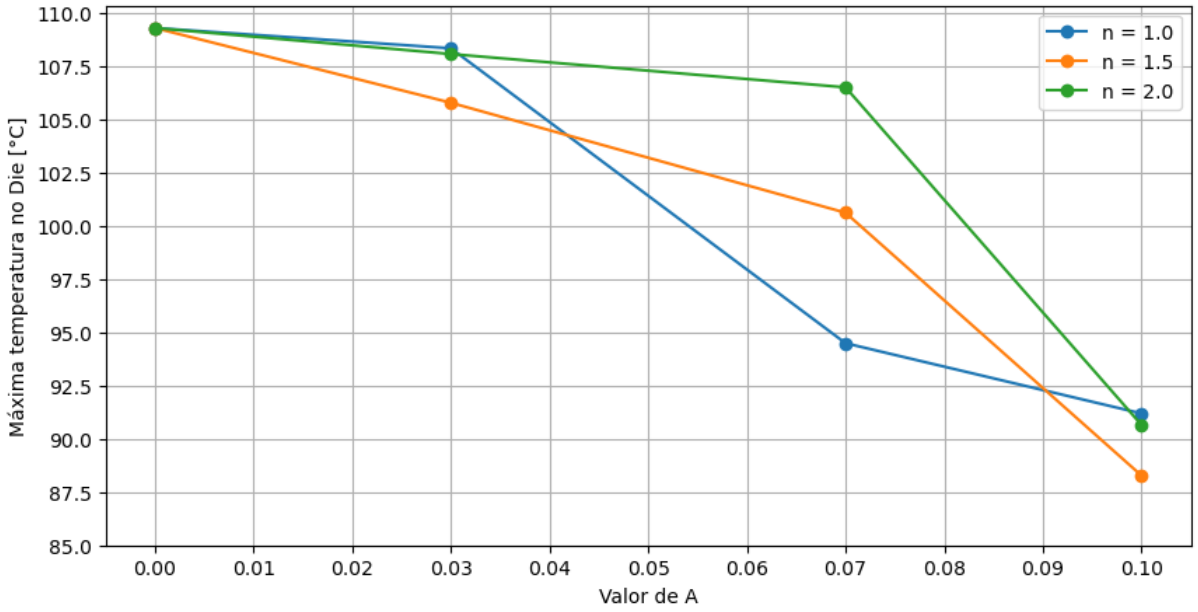


Figura 23 – Máxima temperatura no die para diferente confiugrações (n, A)

Da figura 23, podemos tirar algumas conclusões:

- O efeito do aumento do valor de A tende a melhorar o desempenho do dissipador para um mesmo valor de n.
- Contrário ao que se poderia esperar, o aumento do valor de n não parece melhorar o desempenho do dissipador.
- Como poderia se esperar de um fenômeno complexo, o aumento do valor de A não parece ter um efeito independente do valor de n. Basta ver os pontos de $n = 1.0$ e $A = 0.03$ e $n = 2.0$ e $A = 0.03$ e o efeito do aumento do valor de A nesses casos.

Vale notar que na figura 23, versões do dissipador de mesmo valor para n se encontram numa linha de mesma cor. O dissipador de versão 0 (aletas retas) é considerado como uma hipotética versão onde $A = 0$ e n qualquer, podendo ser agregado em qualquer uma das linhas.

5 CONCLUSÃO

5.1 ORGANIZAÇÃO DO PROGRAMA

O código desenvolvido em *python* se aproveitou da orientação a objeto oferecida pela linguagem. Isso quer dizer que as grandezas e definições do problema são armazenadas em objetos. A organização do código está presente na imagem 24, que apresenta os passos seguidos pela metodologia implementada, com as caixas de traço pontilhado representando arquivos externos previamente configurados pelo usuário.

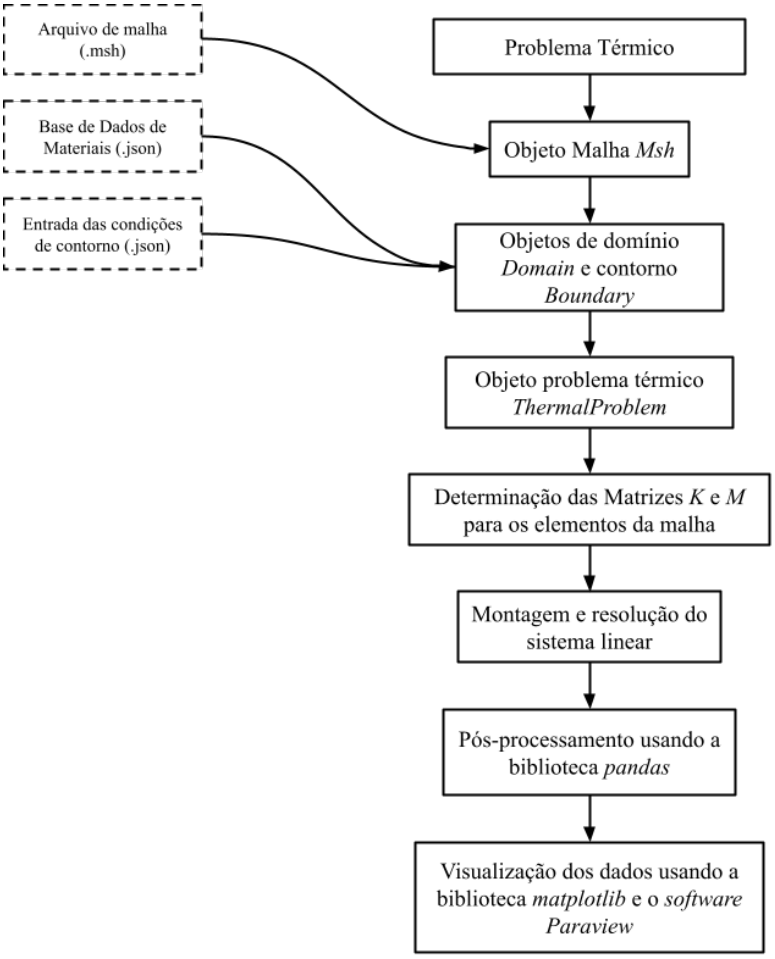


Figura 24 – Fluxograma de organização da metodologia de simulação com os objetos *python* explicitados

A metodologia foi feita da maneira mostrada na imagem 24 para garantir a flexibilidade do código, podendo simular diferentes condições com a simples mudança dos arquivos externos de malha, dados de materiais ou condições de contorno (a configuração desses arquivos está descrita na seção 3.6). Para o caso típico do dissipador de calor por exemplo, o fluxo de trabalho para simular um novo problema térmico é:

1. Geração do arquivo *CAD* utilizando o *software FreeCAD*, exportando um arquivo *step*.
2. Geração da malha utilizando *Gmsh*, refino da malha e configuração dos nomes das regiões de todos os volumes e das regiões do contorno onde há convecção, exportando um arquivo *msh*.
3. Configuração do arquivo *json* para configurar as condições de simulação, configurando a fonte de calor no Die, o coeficiente de transferência de calor convectivo e a temperatura ambiente.
4. Configuração do arquivo *json* de base de dados materiais.
5. Organização de todos os arquivos citados acima em uma pasta única.
6. Execução do código *python* previamente implementado que realiza o fluxo de trabalho descrito na figura 24.

5.2 SÍNTESE DOS RESULTADOS

Número de versão	n	A	Diferença na temperatura máxima em relação a versão 0 (aletas retas) [°C]
0	0.0	0.00	0
1	1.0	0.03	$-0,950$
2	1.0	0.07	$-14,626$
3	1.0	0.10	$-17,975$
4	1.5	0.03	$-3,263$
5	1.5	0.07	$-8,672$
6	1.5	0.10	$-20,876$
7	2.0	0.03	$-1,223$
8	2.0	0.07	$-2,771$
9	2.0	0.10	$-18,552$

Tabela 3 – Tabela de sintetização dos resultados com informações sobre o valor de n, valor de A e a diferença na temperatura máxima em relação à versão 0.

A tabela 3 mostra a consolidação dos resultados mostrados na figura 23. Novamente, é possível verificar um efeito que condiz com a intuição de que o aumento de A aumenta o intervalo entre a temperatura máxima na versão de interesse e a versão 0.

Por outro lado, como sempre é o caso quando se trata simulações numéricas, os resultados obtidos não podem ser tomados como verdade sem outros tipos de validação, apesar da validação do método pela solução manufaturada (ver seção 4.1). Estudos como os de

convergência de malha devem ser feitas, principalmente devido ao problema descrito na seção 5.4 de criação de malhas em elementos finos como as aletas do dissipador de calor.

5.3 RETORNO DE EXPERIÊNCIA

Como descrito na seção 1.2, um dos objetivos do trabalho descrito nesse relatório foi de desenvolver ferramentas em *python* para a modelagem do fenômeno térmicos usando o método de elementos finitos (FEM). No fim, o projeto resultou na criação de múltiplas ferramentas (disponíveis nos anexos) e no domínio de outras ferramentas comerciais:

- Domínio da ferramenta *FreeCAD* para a geração do modelo *CAD*, junto com a criação de *macros* para a agilização e parametrização da criação desses modelos.
- Domínio da ferramenta *Gmsh* para a criação da malha relativa a um modelo com múltiplos elementos e diferentes condições de contorno.
- Criação de objetos *ThermalProblem, Domain* e *Boundary* em *python* para o armazenamento das definições do problema térmico, domínio e contorno, respectivamente.
- Criação de um padrão de nomeação das regiões de domínio e contorno que pode ser processado em um código *python* para a montagem das matrizes K *e* M e a subsequente montagem do sistema linear.
- Desenvolvimento de um código automatizado para o estudo de malha: o código itera em múltiplos arquivos de malha de diferentes refino e um código de pós-processamento para a visualização dos dados.
- Criação de arquivos *json* para a definição da base de dados materiais e das condições de simulação.
- Desenvolvimento de um código que pode iterar em todas as versões do dissipador de calor e executar a mesma análise. Esse mesmo código precisa ser capaz de exportar arquivos *csv* e *vtk* para a leitura dos dados pela biblioteca *pandas* e o *software Paraview*, respectivamente.

Além dos pontos supracitados, esse projeto foi capaz de fornecer um maior entendimento da complexidade de código que existe por trás de softwares comerciais de simulação, mesmo quando se trata da implementação de certas funcionalidades como a frequência de salvamento de arquivos de resultado, por exemplo.

5.4 LIMITAÇÕES E SUGESTÕES PARA TRABALHOS FUTUROS

As principais limitações na implementação deste projeto vêm principalmente do tempo hábil curto para a execução do mesmo e da falta de poder computacional devido ao fato de todos os códigos aqui apresentados terem sido executados utilizando um computador pessoal. Dentre as **limitações** estão duas principais:

1. **Número de elementos na malha**: o número de elementos na malha (ou o refino da mesma) é um parâmetro considerado crítico para a qualidade do resultado oferecido pela metodologia. Por outro lado, esse mesmo parâmetro é altamente limitado pelo poder computacional da máquina utilizada. Esse parâmetro é especialmente importante na simulação de disspadores de calor pois este possui aletas consideradas finas (cerca de 1.5 mm) e uma malha com poucos elementos pode representar de maneira não satisfatória o gradiente de temperatura presente no interior dessas aletas e também a curvatura das mesmas.

2. **Ordem dos elementos finitos**: a metodologia implementada neste projeto conta somente com elementos de ordem 1 (as funções de interpolação entre os nódulos são lineares). Um elemento de ordem superior poderia gerar resultados mais acurados porém não foram implementados devido ao curto tempo hábil.

Conectado às limitações supracitadas e a algumas hipóteses simplificadoras, algumas sugestões para **trabalhos futuros** são:

1. Desenvolvimento de uma análise similar àquela representada neste relatório com um refino maior ou de ordem superior da malha utilizando um computador de maior poder computacional

2. Análise de um dissipador de calor modelando também as pastas térmicas entre o *die* e IHS ou entre o IHS e o dissipador de calor 7

3. Simulação CFD/termal acoplada para a estimação mais precisa do coeficiente de transferência de calor convectiva entre o ar e o dissipador de calor

REFERÊNCIAS

CHOWDHURY, A. S. M. R. et al. A comparative study of thermal aging effect on the properties of silicone-based and silicone-free thermal gap filler materials. **Materials**, 2021. Disponível em: https://doi.org/10.3390/ma14133565.

COOLING, E. **Design For Manufacturability Of Forced Convection Air Cooled Fully Ducted Heat Sinks**. 2007. Disponível em: https://www.electronics-cooling.com/2007/08/design-for-manufacturability-of-forced-convection-air-cooled-fully-ducted-heat-sinks/.

FAWAZ, A. et al. Topology optimization of heat exchangers: A review. **Energy**, n. 252, 2022.

FERREIRA, G. P. Criação de um código próprio em python para simular a transferência de calor em um processador computacional em diferentes arranjos de cargas térmicas usando o método de elementos finitos. 2022.

GENC, A. et al. A review of the emi effect on natural convection heatsinks. **IETE Journal of Research**, 2021.

INTEL. **Information about Temperature for Intel® Processors**. 2022. Disponível em: https://www.intel.com/content/www/us/en/support/articles/000005597/processors.html.

INTEL. **Technical resources: Intel Core Processors**. 2022. Disponível em: https://www.intel.com.br/content/www/br/pt/products/docs/processors/core/coretechnical-resources.html.

LEITAO, A. Cálculo variacional e controle Ótimo. 2001. Disponível em: https://impa.br/wp-content/uploads/2017/04/23_CBM_01_02.pdf.

LEVEQUE, R. J. **Finite Difference Methods for Differential Equations**. Washignton: University of Washington, 2004. 261 p.

LEWIS, R. W.; NITHIARASU, P.; SEETHARAMU, K. **Fundamentals of Finite Element Method for Heat and Fluid Flow**. 8. ed. London: Wiley, 2000.

NWES. **FR4 Thermal Properties to Consider During Design**. 2020. Disponível em: https://www.nwengineeringllc.com/article/fr4-thermal-properties-to-consider-during-design.php.

OZISIK, M. N. **Heat transfer: A basic approach**. [S.l.]: McGraw-Hill, Inc., 1985. 661 p.

SCHUH, M. O. Caracterização e análise térmica de um micrprocessador de alta perfomance via método numérico. 2017.

SIEMENS. **How to Design a Heat Sink for Additive Manufacturing**. 2021. Disponível em: https://blogs.sw.siemens.com/simcenter/how-to-design-a-heat-sink-for-additive-manufacturing/.

TEERTSTRA, P.; YOVANOVICH, M.; CULHAM, J. Analytical forced convection modeling of plate fin heat sinks. p. 18, 1999. Disponível em: http://www.mhtl.uwaterloo.ca/paperlib/present/fc_hs1/fc_hs1.pdf.

WAKEFIELD-VETTE. Simplified method for estimating heatsink thermal resistance for forced convection applications.

ZHU, W. et al. Thermal conductivity of amorphous sio2 thin film: A molecular dynamics study. **Scientific Reports**, v. 8, n. 10537, 2018. Disponível em: https://www.nature.com/articles/s41598-018-28925-6:.

ANEXOS

ANEXO A – BASE DE DADOS DE PARÂMETROS MATERIAIS


```
{
        "alu": {
                "rho": 2698.9,
                "cp": 900,
                "k": [210, 210, 210]
        },
        "silicon": {
                "rho": 2650,
                "cp": 712,
                "k": [1.25, 1.25, 1.25]
        },
        "fr4": {
                "rho": 1850,
                "cp": 1100,
                "k": [0.81, 0.81, 0.29]
        },
        "copper": {
                "rho": 8960.0,
                "cp": 380,
                "k": [401, 401, 401]
        },
        "air": {
                "rho": 1.225,
                "cp": 1000,
                "k": [0.03, 0.03, 0.03]
        },
        "base": {
                "rho": 1,
                "cp": 1,
                "k": [1, 1, 1]
        }
}
```

ANEXO B – ARQUIVO *JSON* PARA CONFIGURAÇÃO DE PARÂMETROS DE SIMULAÇÃO


```
{
        "domain": {
                "copper_heatSink": {
                        "Q": 0
                },
                "silicon_die": {
                        "Q": 0.1733030169e9
                },
                "fr4_pcb": {
                        "Q": 0
                },
                "base_meshStudy": {
                        "Q": 0
                }
        },
        "bc": {
                "copper_fins_rob_01": {
                        "h": 8.460,
                        "Ta": 25
                },
                "base_msh_dir_01": {
                        "T": 25
                }
        }
}
```

ANEXO C – CÓDIGO *PYTHON* PARA VALIDAÇÃO DO MÉTODO POR SOLUÇÃO MANUFATURADA

```
import numpy as np
import meshio
from tools import *
from tqdm import tqdm
from os import environ
import matplotlib.pyplot as plt

environ['OMP_NUM_THREADS'] = '8'

Nt = 500
dt = 1e-3

beta = 0.5 #Crank-Nicholson

# Manufactured function
manu_func = lambda x, y, z, t: (x*x + y*y + z*z)*np.exp(-t)
# Heat Source function
q_func = lambda x, y, z, t: -(6 + x*x + y*y + z*z)*np.exp(-t)

for ii in range(3):
    fpath = './Mesh_study/cube_' + str(ii) + '.msh'

    ThermalProblem = ProblemClass(fpath)
    ThermalProblem.addDomainsAndBoundaries()

    Msh = ThermalProblem.Msh
    X, Y, Z = Msh.X, Msh.Y, Msh.Z

    IEN_boundary = Boundary_IEN(Msh, ThermalProblem.boundaries, 'Tri')
    IEN_domain = Domain_IEN(Msh, ThermalProblem.domains, 'Tet',
                                        IEN_boundary)

    IEN_domain.IENFunc()

    unique_bc = []
    unique_domain = []

    for boundary in ThermalProblem.boundaries.values():
        if len(unique_bc) == 0:
            unique_bc = np.unique(np.reshape(boundary['IEN'], (1, -1)))
        else:
            unique_bc = np.unique(np.concatenate(unique_bc, np.reshape(
```

```
                                                boundary['IEN'], (1, -1)
                                                )))

for domain in ThermalProblem.domains.values():
    if len(unique_domain) == 0:
        unique_domain = np.unique(np.reshape(domain['IEN'], (1, -1))
                                                )
    else:
        unique_domain = np.unique(np.concatenate(unique_domain, np.
                                                reshape(domain['IEN'], (
                                                1, -1))))

unique_bc = set(unique_bc)
unique_domain = set(unique_domain)
unique_core = unique_domain - unique_bc


A = IEN_domain.K.copy()
b = IEN_domain.M@IEN_domain.q_vec.copy()

IEN_domain.replace_lines(A, b)

Npoints = A.shape[0]

T = np.zeros((Nt, Npoints))
Error = np.zeros((Nt, Npoints))

for point in unique_domain:
    x, y, z = X[point], Y[point], Z[point]
    T[0][point] = manu_func(x, y, z, 0)

A1 = (beta*IEN_domain.K.copy() + IEN_domain.M.copy()/dt)
A2 = (-(1 - beta)*IEN_domain.K.copy() + IEN_domain.M.copy()/dt)

for point in unique_bc:
    A1[point] = np.zeros(Npoints)
    A1[point][point] = 1

A1Inv = np.linalg.inv(A1)

q_vec = np.zeros(Npoints)

for i in tqdm(range(1, Nt)):
    t = i*dt

    for point in unique_core:
        x, y, z = X[point], Y[point], Z[point]
```

```
            q_vec[point] = q_func(x, y, z, t)

        bTemp = A2@T[i-1] + IEN_domain.M.copy()@q_vec

        for point in unique_bc:
            x, y, z = X[point], Y[point], Z[point]
            bTemp[point] = manu_func(x, y, z, t)
        T[i] = A1Inv@bTemp

        Error[i] = abs(T[i] - np.array([manu_func(X[i], Y[i], Z[i], t)
                                        for i in range(Npoints)]))

    msh = meshio.read(fpath)

    for i in tqdm(range(Nt)):
        point_data = {"Temperature": T[i], "Error": Error[i]}

        # Export to VTK
        meshio.write_points_cells(
            './Mesh_study/VTK/' + str(ii) + '/meshStudy-V' + str(ii) + '
                                                -malha-00' + str(i) + '.
                                                vtk',
            msh.points,
            msh.cells,
            point_data = point_data
        )

    time_list = np.linspace(0, (Nt - 1)*dt, Nt)
    error_rms_list = [sum(Error[i]**2/Npoints) for i in range(Nt)]

    # Export to CSV
    exportable = np.transpose([time_list, error_rms_list])
    np.savetxt('./Mesh_study/CSVs/data_' + str(ii) + '.csv', exportable,
                                        delimiter=',')
```

ANEXO D – CÓDIGO *PYTHON* PARA A ITERAÇÃO NAS VERSÕES DO DISSIPADOR DE CALOR

```python
import numpy as np
import meshio
from tools import *
from tqdm import tqdm
from os import environ
import matplotlib.pyplot as plt
import json

environ['OMP_NUM_THREADS'] = '7'

Nt = 1000
dt = 1e-3
writeInterval = 20

beta = 0.5 #Crank-Nicholson

for ii in range(13):
    fpath = './Versions/V' + str(ii) + '/heatSink-V' + str(ii) + '.msh'

    ThermalProblem = ProblemClass(fpath)
    ThermalProblem.addDomainsAndBoundaries()

    Msh = ThermalProblem.Msh

    IEN_boundary = Boundary_IEN(Msh, ThermalProblem.boundaries, 'Tri')
    IEN_domain = Domain_IEN(Msh, ThermalProblem.domains, 'Tet',
                                        IEN_boundary)

    IEN_domain.IENFunc()

    A = IEN_domain.K.copy()
    b = IEN_domain.M@IEN_domain.q_vec.copy()

    IEN_domain.replace_lines(A, b)

    Npoints = A.shape[0]

    unique_die = []

    for jj, domain in enumerate(ThermalProblem.domains.values()):
        ienelems = np.unique(np.reshape(domain['IEN'], (1, -1))[0])
        if jj == 0:
```

```
            unique_die = np.unique(ienelems)
        else:
            conc = np.hstack((unique_die, ienelems))
            unique_die = np.unique(conc)

    die_size = len(unique_die)

    T = np.zeros((Nt, Npoints))
    T[0] = np.full(Npoints, 25)

    A1 = (beta*IEN_domain.K.copy() + IEN_domain.M.copy()/dt)
    A2 = (-(1 - beta)*IEN_domain.K.copy() + IEN_domain.M.copy()/dt)

    A1Inv = np.linalg.inv(A1)

    for i in tqdm(range(1, Nt)):
        bTemp = A2@T[i-1] + b
        T[i] = A1Inv@bTemp

    msh = meshio.read(fpath)

    for i in tqdm(range(0,Nt,writeInterval)):
        point_data = {"Temperature": T[i]}

        # Export to VTK for Paraview
        meshio.write_points_cells(
            './Versions/V' + str(ii) + '/VTK/heatsink-V' + str(ii) + '-
                                                malha-00' + str(i) + '.
                                                vtk',
            msh.points,
            msh.cells,
            point_data = point_data
        )

    print('Saved V%i to VTK folder.' % ii)

    time_list = np.linspace(0, (Nt - 1)*dt, Nt)
    avg_T_list = [sum(T[i][unique_die]/die_size) for i in range(Nt)]
    max_T_list = [max(T[i][unique_die]) for i in range(Nt)]

    # Export to CSV
    exportable = np.transpose([time_list, avg_T_list, max_T_list])
    np.savetxt('./Versions/V' + str(ii) + '/data_' + str(ii) + '.csv',
                                        exportable, delimiter=',')
    print('Saved data_%i to CSV file.' % ii)

    geo_data = {
```

```
        'domains':{},
        'boundaries': {}
    }

    for domain in ThermalProblem.domains.values():
        geo_data['domains'][domain['name']] = {
            'volume': domain['volume']
        }

    for boundary in ThermalProblem.boundaries.values():
        geo_data['boundaries'][boundary['name']] = {
            'surface': boundary['surface']
        }

    # Export geometrical data as JSON
    with open('./Versions/V' + str(ii) + '/geo_data_' + str(ii) + '.json
                                        ', 'w') as outfile:
        json.dump(geo_data, outfile)
```

ANEXO E – DEFINIÇÃO DA CLASSE *THERMALPROBLEM*

```
class ThermalProblem():
    def __init__(self, fpath, scale = 0.001):
        self.fpath = fpath

        self.domains = {}
        self.boundaries = {}

        self.Msh = Mesh(fpath, scale = scale)

    def retrieve_material_properties(self, material_name):
        file_path = './JSON/material_db.json'
        try:
            with open(file_path, "r") as json_file:
                data = json.load(json_file)

            try:
                material_properties = data[material_name]
                return material_properties
            except KeyError:
                print(material_name)
                print("Material not found")
                return {'cp': 1, 'k': 1, 'rho': 1}
        except FileNotFoundError:
            print("File not found.")
        except json.JSONDecodeError:
            print("Invalid JSON data in the file.")

    def retrieve_prescription(self, name, type):
        file_path = './JSON/input_conditions.json'
        try:
            with open(file_path, "r") as json_file:
                data = json.load(json_file)

            try:
                material_properties = data[type][name]
                return material_properties
            except KeyError:
                print(name)
                print("Material not found")
                return {'cp': 1, 'k': [1, 1, 1], 'rho': 1, 'alpha': [1,
                                                        1, 1]}
        except FileNotFoundError:
            print("File not found.")
```

```
        except json.JSONDecodeError:
            print("Invalid JSON data in the file.")

    def addDomainsAndBoundaries(self):
        msh = meshio.read(self.fpath)

        # setup boundaries and domains dict
        for key, [tag, dim] in msh.field_data.items():
            if dim == 2:
                self.boundaries[tag] = {
                    'name': key,
                    'dim': dim,
                    'material': key.split('_')[0],
                    'IEN': []
                }

                self.boundaries[tag]['bc'] = key.split('_')[2]
            elif dim == 3:
                self.domains[tag] = {
                    'name': key,
                    'dim': dim,
                    'material': key.split('_')[0],
                    'properties': self.retrieve_material_properties(key.
                                                        split('_')[0]),
                    'IEN': []
                }

                self.domains[tag]['properties']['k'] = np.array(self.
                                                    domains[tag]['
                                                    properties']['k'])
                self.domains[tag]['properties']['alpha'] = self.domains[
                                                    tag]['properties']['
                                                    k']/(self.domains[
                                                    tag]['properties']['
                                                    rho']*self.domains[
                                                    tag]['properties']['
                                                    cp'])
            else:
                print('Something went wrong. Check problem dimensions.')
                break


        # add IENs
        for i, elem in enumerate(msh.cell_data['gmsh:physical']):
            tag = elem[0]

            IEN_temp = msh.cells[i].data
```

```
            if msh.cells[i].dim == 2:
                if len(self.boundaries[tag]['IEN']) == 0:
                    self.boundaries[tag]['IEN'] = IEN_temp
                else:
                    self.boundaries[tag]['IEN'] = np.concatenate([self.
                                                    boundaries[tag][
                                                    'IEN'], IEN_temp
                                                    ])
            elif msh.cells[i].dim == 3:
                if len(self.domains[tag]['IEN']) == 0:
                    self.domains[tag]['IEN'] = IEN_temp
                else:
                    self.domains[tag]['IEN'] = np.concatenate([self.
                                                    domains[tag]['
                                                    IEN'], IEN_temp]
                                                    )
            else:
                print("Something went wrong. Check problem dimensions")
                break

        # add prescription for boundaries
        for key, val in self.boundaries.items():
            name = self.boundaries[key]['name']
            bc_tag = name.split('_')[3]

            if bc_tag != '00':
                self.boundaries[key]['prescription'] = self.
                                                retrieve_prescription
                                                (name, 'bc')
            else:
                bc_type = name.split('_')[2]
                if   bc_type == 'neu':
                    self.boundaries[key]['prescription'] = {'q': 0}
                elif bc_type == 'rob':
                    self.boundaries[key]['prescription'] = {'h': 0, 'Ta'
                                                    : 0}
                elif bc_type == 'dir':
                    self.boundaries[key]['prescription'] = {'T': 0}

        # add prescription for domains
        for key, val in self.domains.items():
            try:
                name = self.domains[key]['name']
                self.domains[key]['prescription'] = self.
                                                retrieve_prescription
                                                (name, 'domain')
```

```
            except KeyError:
                self.domains[key]['prescription'] = {'Q': 0}
```

ANEXO F – DEFINIÇÃO DA CLASSE *MESH*

```
class Mesh():
    def __init__(self, fpath, setCoordsBool = True, scale = 0.001) ->
                                                    None:
        self.fpath = fpath
        self.scale = scale

        if setCoordsBool:
            self.setCoords()

    def setCoords(self):
        msh = meshio.read(self.fpath)
        scale = self.scale

        X = msh.points[:,0]
        Y = msh.points[:,1]
        Z = msh.points[:,2]

        self.X = X*scale
        self.Y = Y*scale
        self.Z = Z*scale

    def exportToVTK(self, name, folder, fields):
        msh = meshio.read(self.fpath)

        point_data = {}

        for key, value in fields.items():
            point_data.update({key: value})

        # plot malha e pontos de contorno (export para vtk)
        meshio.write_points_cells(
            folder + name + '-malha.vtk',
            msh.points,
            msh.cells,
            point_data = point_data
        )

    def exportToVTK_Transient(self, name, folder, fields):
        msh = meshio.read(self.fpath)

        with meshio.xdmf.TimeSeriesWriter(self.fpath) as writer:
            writer.write_points_cells(msh.points, msh.cells)
            for t in [0.0, 0.1, 0.21]:
```

```
                writer.write_data(t, point_data={"Temperature": fields})

        point_data = {}

        for key, value in fields.items():
            point_data.update({key: value})

        # plot malha e pontos de contorno (export para vtk)
        meshio.write_points_cells(
            folder + name + '-malha.vtk',
            msh.points,
            msh.cells,
            point_data = point_data
        )
```

ANEXO G – DEFINIÇÃO DA CLASSE *BOUNDARY*

```
class Boundary_IEN():
    def __init__(self, mesh, boundaries, data_type) -> None:
        self.mesh = mesh
        self.data_type = data_type
        self.boundaries = boundaries

    def IENFunc(self, K, A, b) ->tuple:
        N = 3

        X = self.mesh.X
        Y = self.mesh.Y
        Z = self.mesh.Z

        for boundary in self.boundaries.values():
            boundary['surface'] = 0

        for boundary in tqdm(self.boundaries.values()):
            for IENelem in boundary['IEN']:
                for point in IENelem:
                    BCType = boundary['bc']
                    tri_matrix = [[X[idx], Y[idx], Z[idx]] for idx in
                                                        IENelem]
                    tri_area = 0.5*abs(np.linalg.det(tri_matrix))
                    boundary['surface'] += tri_area

                    if BCType == 'neu':
                        if b[point] is None:
                            b[point] = 0

                        b[point] += boundary['prescription']['q']*
                                                            tri_area
                    elif BCType == 'rob':
                        if b[point] is None:
                            b[point] = 0

                        h = boundary['prescription']['h']*tri_area
                        b[point] += h*boundary['prescription']['Ta']

                        K[point][point] += h

        for key, boundary in tqdm(self.boundaries.items()):
            for IENelem in boundary['IEN']:
                for point in IENelem:
```

```
                    if boundary['bc'] == 'dir':
                        if A[point] is None:
                            A[point] = np.zeros(A.shape[0])
                            b[point] = 0

                        A[point][point] = 1
                        b[point] = self.boundaries[key]['prescription']['
                                                                'T']

        for boundary in self.boundaries.values():
            print("%s: %.3e mm2" % (boundary['name'], boundary['surface'
                                                    ]*10**6))

        return A,b
```

ANEXO H – DEFINIÇÃO DA CLASSE *DOMAIN*

```
class Domain_IEN():
    def __init__(self, mesh, domains, data_type, Lower_IEN) -> None:
            self.mesh = mesh
            self.domains = domains
            self.data_type = data_type
            self.Lower_IEN = Lower_IEN

            self.K = np.zeros((mesh.X.shape[0], mesh.X.shape[0]), dtype
                                                    = float)
            self.M = np.zeros((mesh.X.shape[0], mesh.X.shape[0]), dtype
                                                    = float)


    def replace_lines(self, A, b):
        N = b.shape[0]

        A_replace = self.A_replace
        b_replace = self.b_replace

        for i in tqdm(range(N)):
            if b_replace[i] is not None:
                b[i] = b_replace[i]

            if A_replace[i] is not None:
                A[i] = A_replace[i]

    def IENFunc(self):
        N = 4

        X = self.mesh.X
        Y = self.mesh.Y
        Z = self.mesh.Z

        A_replace = np.full(X.shape[0], None)
        b_replace = np.full(X.shape[0], None)

        q_vec     = np.zeros(X.shape[0])

        for domain in self.domains.values():
            domain['volume'] = 0

        for domain in tqdm(self.domains.values()):
            for IENelem in domain['IEN']:
                tet_matrix = np.array([[1, X[IENelem[i]], Y[IENelem[i]],
```

```
                                        Z[IENelem[i]]] for
                                       i in range(N)])

tet_volume = (1/6)*np.linalg.det(tet_matrix)
domain['volume'] += tet_volume

[a_list, b_list, c_list, d_list] = np.linalg.inv(
                                        tet_matrix)

melem = (tet_volume/20)*np.array(([2, 1, 1, 1],
                                  [1, 2, 1, 1],
                                  [1, 1, 2, 1],
                                  [1, 1, 1, 2]))

kxelem = tet_volume*np.array([[b_list[i]*b_list[j] for i
                                        in range(N)] for j
                                       in range(N)])
kyelem = tet_volume*np.array([[c_list[i]*c_list[j] for i
                                        in range(N)] for j
                                       in range(N)])
kzelem = tet_volume*np.array([[d_list[i]*d_list[j] for i
                                        in range(N)] for j
                                       in range(N)])

k_vec = [kxelem, kyelem, kzelem]

kelem = kxelem*domain['properties']['alpha'][0] + kyelem
                                       *domain['properties'
                                       ]['alpha'][1] +
                                       kzelem*domain['
                                       properties']['alpha'
                                       ][2]

for ilocal in range(N):
    iglobal = IENelem[ilocal]

    for jlocal in range(N):
        jglobal = IENelem[jlocal]

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

for point in IENelem:
    q_vec[point] = domain['prescription']['Q']/(domain['
                                            properties']['
                                            rho']*domain['
                                            properties']['cp
```

```
                                                    '])


            A_replace, b_replace = self.Lower_IEN.IENFunc(self.K, q_vec,
                                                A_replace, b_replace)

            self.q_vec = q_vec
            self.A_replace = A_replace
            self.b_replace = b_replace

            for domain in self.domains.values():
                print("%s: %.3e mm3" % (domain['name'], domain['volume']*10
                                                        **9))
```