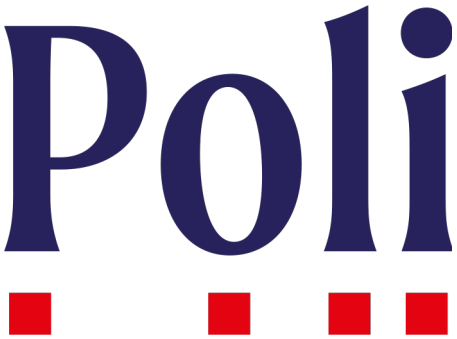


MÉTODO DOS ELEMENTOS FINITOS NA TRANSFERÊNCIA DE CALOR DE UM MÓDULO REGULADOR DE TENSÃO RESFRIADO POR UM DISSIPADOR DE CALOR

Gabriel Affonso Costa Waehneldt

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, Ph.D

Rio de Janeiro

Agosto de 2023

UNIVERSIDADE FEDERAL DO RIO DE JANEIRO

Departamento de Engenharia Mecânica

DEM/POLI/UFRJ

MÉTODO DOS ELEMENTOS FINITOS NA TRANSFERÊNCIA DE CALOR DE UM MÓDULO REGULADOR DE TENSÃO RESFRIADO POR UM DISSIPADOR DE CALOR

Gabriel Affonso Costa Waehneldt

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. Lavinia Maria Sanabio Alves Borges, D.Sc.

_____

Prof. Marcelo Amorim Savi, D.Sc.

RIO DE JANEIRO, RJ – BRASIL

AGOSTO DE 2023

Affonso Costa Waehneldt, Gabriel

Método dos Elementos Finitos na Transferência de Calor de um Módulo Regulador de Tensão Resfriado por um Dissipador de Calor/ Gabriel Affonso Costa Waehneldt. – Rio de Janeiro: UFRJ/Escola Politécnica, 2023.

XIV, 99 p.: il.; 29, 7cm.

Orientador: Gustavo Rabello dos Anjos, Ph.D

Projeto de Graduação – UFRJ/ Escola Politécnica/ Curso de Engenharia Mecânica, 2023.

Referências Bibliográficas: p. 73 – 73.

1. Elementos Finitos. 2. Transferência de Calor. 3. Modulo Regulador de Tensão. I. Ph.D, Gustavo Rabello dos Anjos,. II. Universidade Federal do Rio de Janeiro, UFRJ, Curso de Engenharia Mecânica. III. Método dos Elementos Finitos na Transferência de Calor de um Módulo Regulador de Tensão Resfriado por um Dissipador de Calor.

A minha família que nunca mediu esforços para me apoiar em tudo.

Agradecimentos

Agradeço a meus pais, Jones e Vera, que ao longo de toda minha vida me apoiaram em tudo. Eles foram compreensivos quando precisei, me deram forças e sempre estavam presentes por mim. Sem eles, não teria chegado ao fim da minha graduação. Agradeço também aos muitos alunos com que tive contato na UFRJ, amigos, conhecidos e colegas de aula. Cada um de vocês tiveram uma contribuição, grande ou pequena, no meu avanço na vida acadêmica. Agradeço aos meus amigos de fora da faculdade que me ajudaram muito nesses anos.

Agradeço também a todos os professores que compõe o Departamento de Engenharia Mecânica por terem me guiado nos meus primeiros passos para me tornar um Engenheiro Mecânico. Agradeço em especial o Professor e meu orientador Gustavo, por ter apresentado de maneira tão prática e interessante o método de elementos finitos em suas aulas durante a pandemia. Esse primeiro contato com o tema foi essencial para eu conhecer e desenvolver interesse nos métodos numéricos e por ter escolhido isso como tema do meu trabalho de conclusão de curso.

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

MÉTODO DOS ELEMENTOS FINITOS NA TRANSFERÊNCIA DE CALOR DE UM MÓDULO REGULADOR DE TENSÃO RESFRIADO POR UM DISSIPADOR DE CALOR

Gabriel Affonso Costa Waehneldt

Agosto/2023

Orientador: Gustavo Rabello dos Anjos, Ph.D

Programa: Engenharia Mecânica

O projeto consiste no desenvolvimento de um programa em linguagem Python capaz de calcular soluções aproximadas para o estado térmico dos componentes termicamente mais relevantes em módulos reguladores de tensão quando esses estão alimentando eletricamente um processador trabalhando em seu limite. Esses componentes mais críticos são comumente arrefecidos com dissipadores de calor e o programa engloba-os na simulação.

Esse cálculo é realizada por meio do Método dos Elementos Finitos em que se obtém a solução da equação de calor tridimensional, tendo como condição de contorno a convecção no dissipador e tendo geração de calor interna nos módulos reguladores.

Esse programa proporciona uma alternativa para se realizar um dimensionamento preliminar da alimentação elétrica de processadores uma vez que ele aceita diferentes cargas do processador, números de fases e geometrias de dissipadores.

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

FINITE ELEMENT METHOD FOR HEAT TRANSFER IN VOLTAGE REGULATOR MODULE COOLED BY HEAT SINK

Gabriel Affonso Costa Waehneldt

August/2023

Advisor: Gustavo Rabello dos Anjos, Ph.D

Department: Mechanical Engineering

The project consists of developing a program in Python language capable of calculating approximate solutions for the thermal state of the most thermally relevant components in voltage regulator modules when they are electrically feeding a processor working at its limit. These most critical components are commonly cooled with heatsinks and the program includes them in the simulation.

This calculation is carried out using the Finite Element Method, in which the solution of the three-dimensional heat equation is obtained, having as a boundary condition the convection in the heatsink and having internal heat generation in the regulator modules.

This program provides an alternative for carrying out a preliminary dimensioning of the voltage regulator modules of processors, since it accepts different processor loads, number of phases and geometries of heatsinks.

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

Em muitas situações, é vantajoso se realizar simulações para que se conheça uma aproximação do estado térmico de uma dada peça ou sistema em cenários distintos antes de fabrica-lo. Um procedimento comum é se realizar uma modelagem físico-matemática do problema e assim se alcançar um resultado aproximado da realidade. Contudo, em muitos casos, a solução analítica do problema é difícil ou impossível e, nessas situações, se faz necessário o uso de um modelo numérico. Há vários métodos diferentes, sendo os mais comumente utilizados o de Diferenças Finitas, Volumes Finitos e Elementos Finitos. Este último foi o método escolhido para o projeto em função de ser altamente consolidado em seu uso na engenharia, como em problemas de transferência de calor, além de possuir ampla literatura.

Módulos reguladores de tensão, também conhecidos como *VRM*, estão presentes em placas eletrônicas, cumprindo o papel de fornecer tensões e correntes adequadas as necessidades e especificações dos componentes que alimentam. Apesar de serem eficientes quando operados dentro de uma determinada faixa de corrente, as pequenas dimensões de seus componentes fazem com que seus estados térmicos sejam pontos relevantes no projeto de placas eletrônicas para computadores. A recente escalada nas correntes solicitadas por processadores torna o arrefecimento desses componentes ainda mais relevante. Um projeto mal dimensionado pode leva-los a operar fora das especificações térmicas dos fabricantes e consequentemente serem danificados, além de danificarem os *chips* que estão alimentando.

A finalidade do projeto é o desenvolvimento de uma ferramenta capaz de simular o estado térmico de componentes termicamente relevantes de um módulo regulador

de tensão, quando estes estão sendo arrefecidos por dissipadores orientados verticalmente em convecção natural. Também está no escopo do projeto o código ser capaz de ter flexibilidade na geração da geometria dos dissipadores, aceitar números de fases distintos no *VRM* e também diferentes solicitações por parte do processador que está sendo alimentado.

O desenvolvimento de um código para solução por MEF da equação de calor tridimensional se deu por meio da linguagem de programação *Python* que foi dividido em diferentes módulos para maior organização, legibilidade e facilidade de modificação. Comparações entre os resultados do programa e um *software* comercial são usados para validação, além de um caso analítico para teste de convecção.

A geração de calor por parte dos componentes termicamente relevantes dos módulos reguladores de tensão são estimados conforme gráficos fornecidos pelo fabricante desses produtos. Já o coeficiente de convecção aplicado a condição de contorno nos dissipadores de calor é baseado em equações calibradas em estudos experimentais [6].

Capítulo 2

Revisão Bibliográfica

2.1 Transferência de Calor

O calor, segundo Incropera[7], é a energia em trânsito devido a uma diferença de temperatura no espaço. A ciência da termodinâmica lida com a relação do Calor com outras formas de energia, diferentemente da transferência de calor que, para Ozisik[8], estuda a taxa com que essa energia térmica está se movimentando. Quantificar os fluxos de calor é algo necessário para um grande número de aplicações típicas de engenharia. Para projetos de sistemas que tem por função transferir calor é necessário quantificar como se darão os fluxos para ser possível um correto dimensionamento.

A transferência de calor é usualmente dividida em 3 tipos, a condução, a convecção e a radiação. Esses fenômenos não ocorrem separadamente, mas atuam simultaneamente transferindo calor através e entre sistemas. Contudo, quando o efeito de algum tipo de transferência de calor é comparativamente pequeno é razoável despreza-lo quando se está realizando uma modelagem do sistema.

Será apresentada uma breve revisão de cada um dos 3 tipos de transferência de calor em conjunto com suas respectivas descrições matemáticas.

2.1.1 Condução

A condução pode ser definida como o processo onde ocorre transferência de energia das moléculas mais energizadas para as menos energizadas por meio das interações e ligações que elas possuem. A taxa de transferência de calor é intimamente

ligada ao grau de interação molecular, variando substancialmente com o estado e o material. A equação que descreve a condução é chamada de lei de Fourier, que simplificada para uma única dimensão assume a forma:

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

Contudo, em casos reais, a condução ocorre em todas as 3 dimensões e em modelos mais complexos não há possibilidade de simplificação sem perda de eficácia da modelagem. A equação de Fourier generalizada para todas as dimensões têm a forma:

$$q_{cond} = -k\nabla T = -k\left(i\frac{\partial T}{\partial x} + j\frac{\partial T}{\partial y} + k\frac{\partial T}{\partial z}\right) \tag{2.2}$$

Sendo $q_x[W/m^2]$ o fluxo de calor por área e $k[W/mK]$ a condutividade térmica. O valor da condutividade varia com o tipo de material e com a temperatura do mesmo, uma vez que as interações moleculares variam em função de ambas.

2.1.2 Convecção

Quando há velocidade relativa entre uma superfície e um fluido além de uma diferença de temperatura entre eles, ocorre a transferência de calor denominada convecção. Nele existe a superposição de 2 fenômenos distintos, a transferência de calor por meio da ligação e iteração entre moléculas, além do próprio movimento das mesmas no fluido. Quando o movimento ocorre em função de uma força externa causada por um ventilador, por exemplo, a convecção é dita forçada. Quando ele ocorre em função do empuxo a convecção é chamada de natural ou livre. O fenômeno do empuxo é originado das diferenças de densidade que são caudados por variações de temperatura ao longo do fluido.

Sobre a convecção, ainda existe uma diferenciação entre a troca de calor que se dá entre a superfície e o fluido. Quando o gás ou líquido só apresenta uma mudança de temperatura mantendo-se a mesma fase, a troca de calor é chamada de sensível. Quando há troca de fase ela é chamada de latente. Segue a equação da Lei de Resfriamento de Newton que descreve o fenômeno:

$$q_{conv} = h(T_S - T_\infty) \tag{2.3}$$

Sendo h o coeficiente de convecção em W/m^2K, $T_S[K]$ a temperatura de parede, $T_\infty[K]$ a temperatura do meio fluido e $q_{conv}[W/m^2]$ o fluxo de calor por convecção.

O valor do coeficiente de convecção varia de uma série de variáveis como geometria da superfície, temperatura de superfície e meio entre outros e, em muitos casos, não pode ser calculado analiticamente.

2.1.3 Radiação

Segundo Incropera[7], a radiação é a energia que é transportada por ondas eletromagnéticas (ou fótons) dos corpos com temperatura não nula, estando o corpo no estado sólido, líquido ou gasoso. Esse processo é equacionado pela lei de Stefan-Boltzman:

$$q_{rad} = \varepsilon \sigma T_S^4 \tag{2.4}$$

Sendo que q_{rad} é o fluxo de calor por radiação de um corpo, σ é a constante de Stefan-Boltzmann ($5,67 \times 10^{-8} W/m^2K^4$) e ε é a emissividade (uma propriedade do material).

2.1.4 Equação de Calor

Para este trabalho é necessário a determinação da distribuição de temperatura no espaço e no tempo do sistema em estudo. Para isso, será derivada uma equação de calor em coordenadas cartesianas proveniente da lei de conservação de energia em um volume de controle[1].

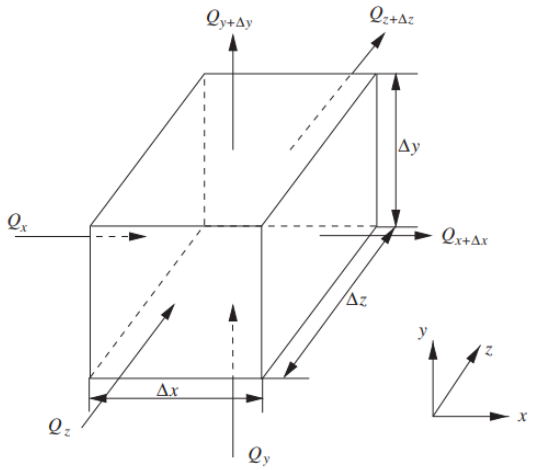


Figura 2.1: Volume de controle para análise da condução de calor.[1]

Com base na figura 2.1, para que haja a conservação de energia térmica em um estado permanente, o calor entrando e saindo do volume de controle precisa ser igual.

$$Q_x + Q_y + Q_z = Q_{x+dx} + Q_{y+dy} + Q_{z+dz} \tag{2.5}$$

No estado transiente, outras considerações precisam ser feitas, como a possibilidade de mudança de temperatura no sistema. Isso fica representado na equação na forma de taxa de mudança no armazenamento de energia térmica:

$$\rho c_p(\Delta x \Delta y \Delta z)\frac{\partial T}{\partial t} \tag{2.6}$$

A geração de calor interna é outro fator não considerado na equação 2.5, sendo que este termo assume a forma de $Q_{gen}(\Delta x \Delta y \Delta z)$.

Adicionando a geração de calor no termo esquerdo da equação 2.5 ao lado da entrada de calor no volume de controle e a 2.6 no termo direito, temos uma equação de calor transiente com geração de calor interna[1]:

$$Q_x + Q_y + Q_z + Q_{gen}(\Delta x \Delta y \Delta z) = \rho c_p(\Delta x \Delta y \Delta z)\frac{\partial T}{\partial t} + Q_{x+dx} + Q_{y+dy} + Q_{z+dz} \tag{2.7}$$

Contudo, trabalhar com os termos de entrada e saída de calor no volume de controle não é prático, sendo melhor o uso da diferença entre entrada e saída em cada eixo. Para isso, será usado uma expansão em série de taylor para abrir Q_{x+dx}:

$$Q_{x+dx} = Q_x + \frac{\partial Q_x}{\partial x}\Delta x \tag{2.8}$$

Sendo o procedimento análogo para as outras duas direções.

Substituído a 2.8 na 2.7 e simplificando-a, temos:

$$-\frac{\partial Q_x}{\partial x}\Delta x - \frac{\partial Q_y}{\partial y}\Delta y - \frac{\partial Q_z}{\partial z}\Delta z + Q_{gen}(\Delta x \Delta y \Delta z) = \rho c_p(\Delta x \Delta y \Delta z)\frac{\partial T}{\partial t} \tag{2.9}$$

O fluxo de calor total Q em cada direção pode ser expresso também como o fluxo em área q e posteriormente esse último termo pode ser substituído usando a equação 2.1 da condução:

$$Q_x = (\Delta y \Delta z)q_x = -k_x(\Delta y \Delta z)\frac{\partial T}{\partial t} \tag{2.10}$$

Sendo a expressão análoga para as demais direções.

Substituindo 2.10 em Q_x, Q_y e Q_z na 2.9, dividindo pelo volume e considerando o material em estudo isotrópico (k é igual em todas as direções), temos que a 2.9 assume a forma de:

$$-k\left(\frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2} + \frac{\partial^2 T}{\partial z^2}\right) + Q_{gen} = \rho c_p \frac{\partial T}{\partial t} \tag{2.11}$$

O operador Laplaciano em coordenadas cartesianas é definido por:

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

Com o operador, podemos expressar a equação de calor 2.11 de maneira mais compacta:

$$-k\nabla^2 T + Q_{gen} = \rho c_p \frac{\partial T}{\partial t} \tag{2.12}$$

2.2 Módulos Reguladores de Tensão

O avanço e desenvolvimento de microprocessadores, segundo Chen[2], levou a uma gradual redução na tensão e aumento das correntes que precisam ser fornecidas a esses componentes. A demanda por uso de frequências mais altas e litografias mais

finas, gera essa tendência. Para isso, é necessário que as altas tensões fornecidas pela fonte sejam convertidas de maneira eficiente para valores adequados aos *chips* atuais. Os módulos reguladores de tensão cumprem esse papel e eles geralmente são compostos de conversores Buck.

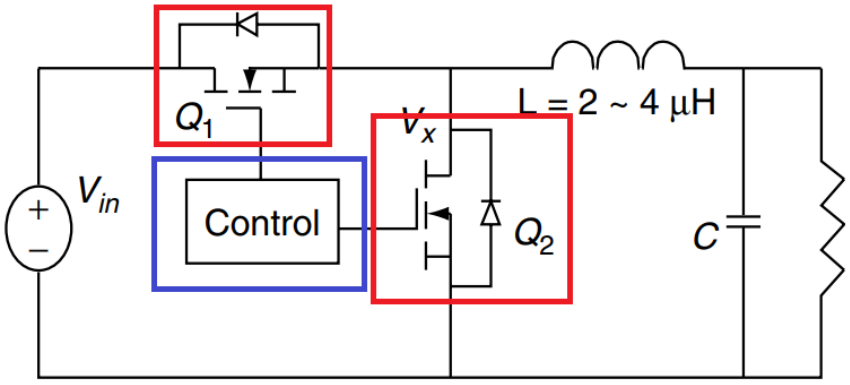


Figura 2.2: Diagrama de um conversor Buck síncrono. Em vermelho os transistores *MOSFET* e em azul o controlador.[2]

Os conversores Buck funcionam como uma fonte chaveada, abrindo e fechando o circuito permitindo a passagem de corrente para o componente eletrônico em uma frequência alta o suficiente que se tem uma redução de tensão com um aumento de corrente. Esse chaveamento é feito por 2 transistores do tipo *MOSFET*, destacados em vermelho na figura 2.2. Nesses conversores também há a presença de indutores que tem a função de suavizar as variações de tensão e torna-la mais estável para o correto funcionamento dos *chips* que estão sendo alimentados.

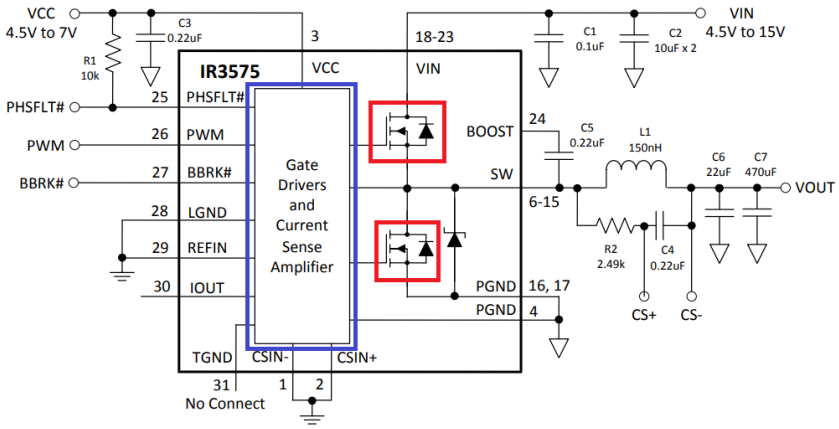


Figura 2.3: Transistores integrados ao controlador, modelo IR3575. Em vermelho os transistores *MOSFET* e em azul o controlador.[3]

Outro componente presente é o controlador dos transistores que na figura 2.2 está destacado em azul. Seguindo a tendência de miniaturização e consolidação de componentes eletrônicos é comum a utilização de produtos que integram os 2 transistores e seus respectivos controladores em um único componente. Um exemplo dessa integração pode ser visto no produto IR3575 na imagem 2.3. Este produto será o componente utilizado na simulação numérica neste trabalho e seu nome será frequentemente intercambiado com os dos *MOSFET's* que o integram.

Os *MOSFET's* são os maiores geradores de calor de um conversor Buck e, por isso, necessitam de maior atenção em seu arrefecimento sendo geralmente acoplados a dissipadores de calor. O IR3575, primariamente composto de *MOSFET's*, tem a sua eficiência de conversão variando em função da corrente e tensão com que operam.

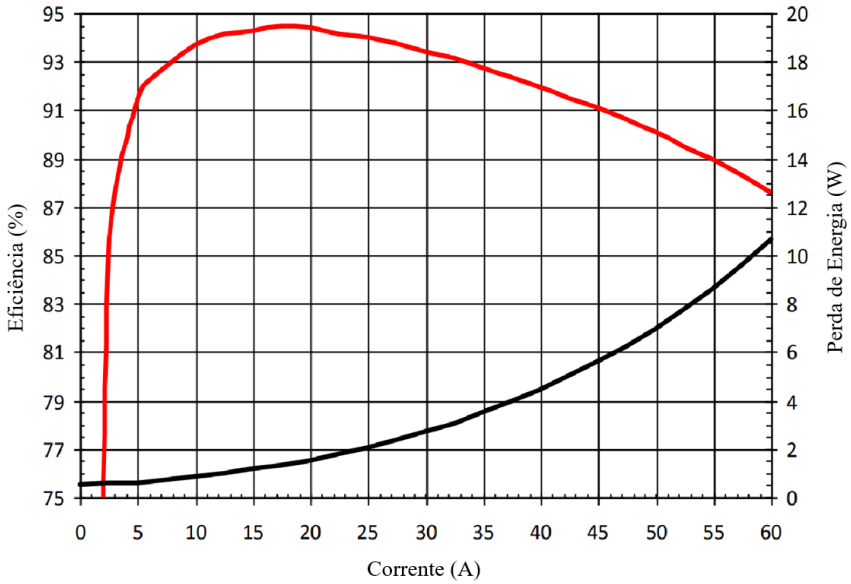


Figura 2.4: Eficiência e perda de energia do *MOSFET* IR3575 em função da corrente quando em 1.2 Volt.[3]

Observando a figura 2.4, é possível perceber que há uma região ótima de operação dos *MOSFET's* quanto a eficiência, entre 15 e 20 Amperes. Ao exceder essa faixa, a eficiência decresce gradativamente, aumentando substancialmente a geração de calor interna.

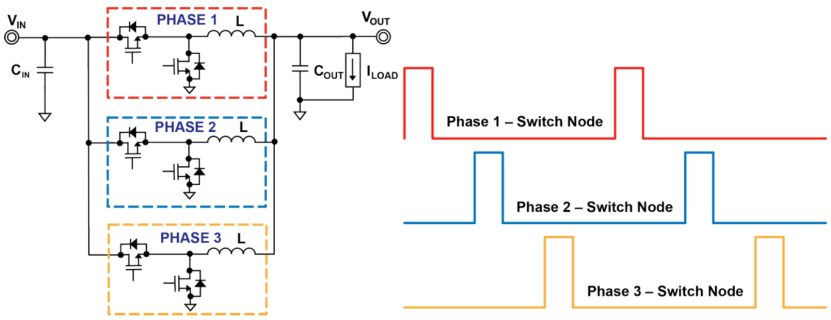


Figura 2.5: Módulo regulador de tensão trifásico de conversores Buck síncronos.[4]

Um único conversor Buck, para não exceder a temperatura de operação do *MOSFET*, não pode suprir toda a corrente que muitos dos micro processadores requerem. Por isso, é comum a utilização de vários desses conversores, formando um modulo de regulação de tensão multifásico e dividindo a corrente solicitada igualmente entre elas. Um exemplo de um módulo regulador de tensão trifásico pode ser visto na figura 2.5.

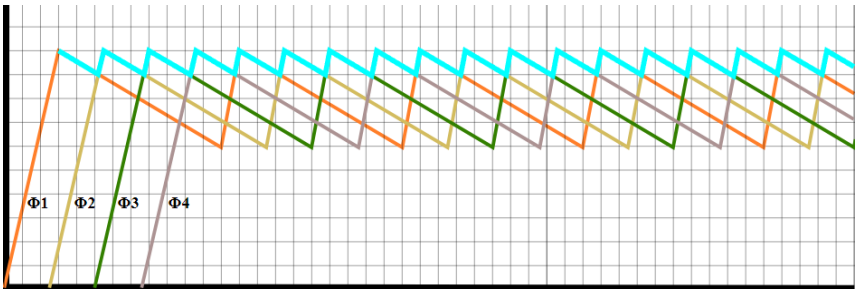


Figura 2.6: Imagem exemplificando a maior estabilidade da tensão de saída (em azul) quando comparada a uma única fase (demais cores). Eixo x corresponde ao tempo e eixo y a tensão.

Outro efeito da configuração multifásica de um *VRM* é a maior estabilidade da tensão de saída que se consegue obter. Com esse efeito, não se torna necessário operar os conversores Buck em uma frequência demasiadamente elevada para se ter uma saída suficientemente estável para alimentação adequada dos processadores.

2.3 Demanda Elétrica de Processadores

Os processadores são componentes eletrônicos que possuem uma demanda por energia que varia conforme sua carga de trabalho. A sua alimentação elétrica é fre-

quentemente feita por um *VRM* que tem como função modificar a tensão e corrente recebida para valores que se adéquem as suas especificações e demandas ao longo do tempo.

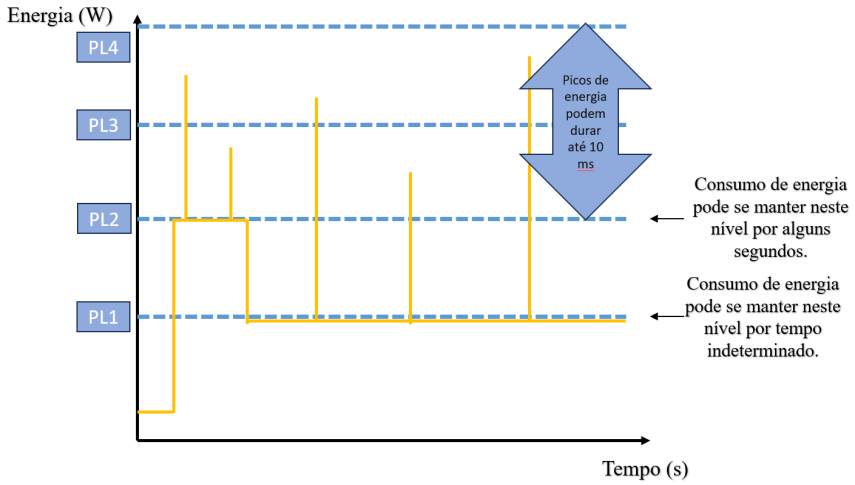


Figura 2.7: Comportamento de limite de demanda energética em processadores Intel de 12ª geração.

Fonte: Elaborado pelo Autor

Os processadores modernos, como os da 12ª geração da Intel, possuem tipicamente 4 limites em sua demanda energética, como pode ser visto na figura 2.7. São eles o *Power Limit 1* ou *PL1* que pode ser sustentado pelo componente indefinidamente e o *PL2* que é atingido por tempo limitado. Os outros 2 limites são mantidos por não mais de 10 milissegundos e com isso não são relevantes no presente projeto.

2.4 Método de Elementos Finitos

Nesta seção será apresentado a história do desenvolvimento do Método de Elementos Finitos (MEF), bem como será realizada uma breve explicação das suas etapas e funcionamento.

2.4.1 História do Desenvolvimento do Método

Segundo Logan [9], a origem do Método de Elementos Finitos remonta do início da década de 1940 quando Hrennikoff em 1941 e McHenry em 1943 utilizaram elementos de barras e vigas (unidimensional) para resolver problemas de tensão em

sólidos. Ainda em 1943 foi publicado um artigo de Courant onde ele propunha resolver as tensões de forma variacional. No mesmo trabalho ele introduz o conceito de funções de forma sobre sub-regiões triangulares fazendo com que toda ela tenha uma solução numérica aproximada. Levy também realizou contribuições nesse ramo em 1947 e em 1953, mas a dificuldade de se resolver os cálculos necessários para se aplicar os métodos desenvolvidos até então tornavam pouco prático os seus usos. Foi só com o trabalho de Turner que em 1956 se começou a utilizar elementos bidimensionais, na forma de triângulos e retângulos, para se calcular tensão. O procedimento conhecido como método matricial de rigidez, capaz de obter a matriz global de rigidez da estrutura, foi definido por ele.

Com a introdução de computadores digitais mais poderosos na segunda metade de 1950, tornou-se mais viável a utilização de métodos numéricos com requerimentos computacionais mais altos, fazendo com que a área fosse mais estudada e desenvolvida. Algum tempo depois, no início dos anos 1960, o MEF se tornou mais capaz com a introdução de elementos tetraédricos, o que possibilitou a análise de estruturas tridimensionais. Outro importante desenvolvimento ocorreu novamente com o trabalho de Turner, que em 1960 conseguiu com que o método de elementos finitos conseguisse trabalhar com estruturas que sofriam grandes deformações (anteriormente só se considerava deformações elásticas dos materiais). Ele também introduziu análises térmicas ao método.

2.4.2 Etapas do Método de Elementos Finitos

Para a aplicação do Método de Elementos Finitos, o sistema que precisa ser modelado é dividido (discretizado) na forma de elementos, também chamados de células. Eles podem ser uni, bi ou tridimensional, dependendo da simulação, além de terem diferentes formas, como mostra as imagens 2.8 e 2.9. Por sua vez, os elementos são compostos de nós e arestas, sendo que o número e organização dos mesmos vai variar para cada tipo de célula.

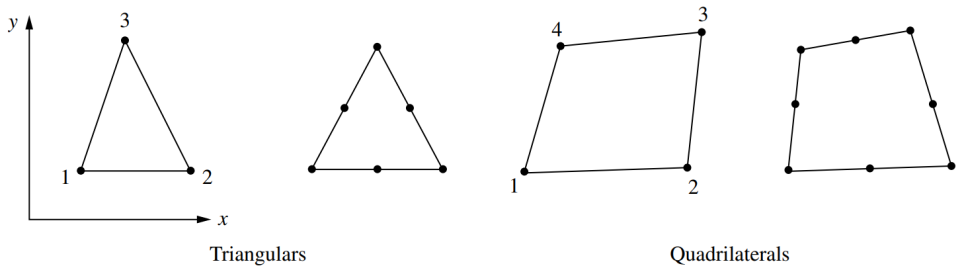


Figura 2.8: Diferentes tipos de elementos bidimensionais[1].

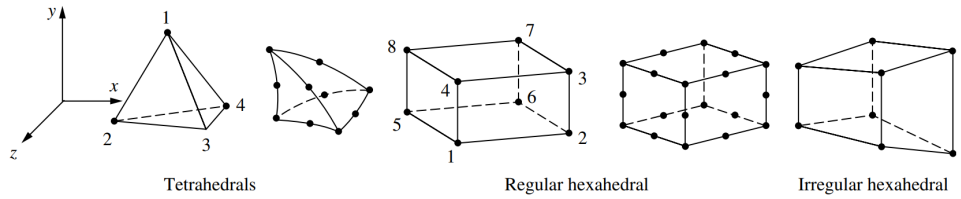
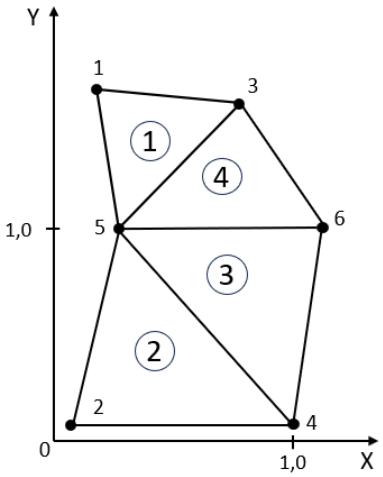


Figura 2.9: Diferentes tipos de elementos tridimensionais[1].

A união das células, como pode ser visto à esquerda na imagem 2.10, ocorre por meio dos nós, uma vez que os mesmos fazem parte de diferentes elementos. Da união desses elementos se forma a malha que discretiza a geometria do modelo. Para se corretamente definir uma malha é preciso se conhecer a posição de cada nó e de que maneira eles estão conectados para formar os elementos. Esse último é organizado através de uma matriz de conectividade, conhecida como IEN. Nela, cada linha da matriz corresponde a um elemento da malha e cada valor ao índice dos nós que a compõe. Um exemplo disso pode ser visto à direita na imagem 2.10, onde se tem a IEN e a posição dos nós para a malha triangular, visualizada na mesma figura.



ELEM	v_1	v_2	v_3
1	1	5	3
2	2	4	5
3	5	4	6
4	5	6	3

VER	X	Y
1	0,2	1,8
2	0,1	0,1
3	0,8	1,7
4	1,0	0,1
5	0,25	1,0
6	1,1	1,0

Figura 2.10: Visualização de malha composta de elementos triangulares, bem como sua IEN e posicionamento dos nós.

Fonte: Elaborado pelo Autor

O início do Método de Elementos Finitos se dá na modificação da equação diferencial que descreve o problema (forma forte) para a sua forma fraca (variacional) equivalente no domínio em questão. A transformação é feita por meio da multiplicação de uma função peso $w(x, y, z)$ pela equação diferencial $f''(x, y, z)$ e pela integração no domínio do problema:

$$\int_{\Omega} w(x, y, z) f''(x, y, z)\, d\Omega \tag{2.13}$$

O próximo passo consiste em aplicar a integração por partes pelo teorema de Green para redução da ordem da equação 2.13:

$$\int_{\Omega} w f''\, d\Omega = \int_{\Gamma} w f'\, d\Gamma - \int_{\Omega} w' f' \tag{2.14}$$

A etapa seguinte no MEF corresponde a transformar as funções diferenciais continuas presentes na forma fraca em discretas. Uma das maneiras de se alcançar isso é pela utilização do método de resíduos ponderados, sendo ele capaz de encontrar funções que aproximam a solução exata que se está tentando substituir em um determinado intervalo. Ele faz isso através de um somatório de funções de forma N, também chamado de funções interpoladoras [1], multiplicado com amplitudes arbitrárias a, para cada elemento da malha:

$$f(x,y,z) = \sum_{i=1}^{n} N_i(x,y,z)a_i \tag{2.15}$$

$$w(x,y,z) = \sum_{j=1}^{n} N_j(x,y,z)a_j \tag{2.16}$$

As funções de forma, segundo Nithiarasu [1], são definidas para terem valor unitário em seu respectivo nó e terem valor nulo nos demais nós do elemento. A função interpoladora entre pontos é geralmente polinomial para maior facilidade de derivação e integração da mesma. Nesse trabalho foi escolhido uma função linear pelo seu custo computacional mais baixo.

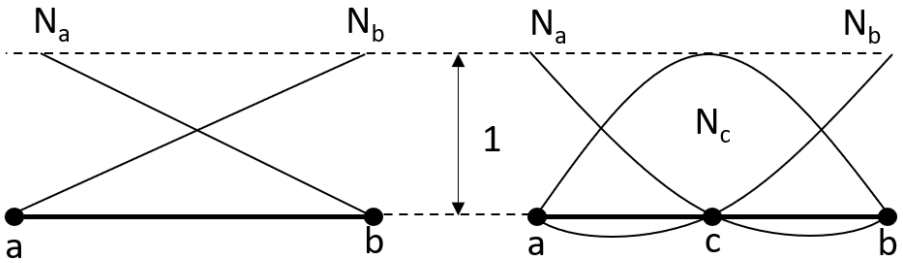


Figura 2.11: Função de forma linear e quadrática.

Fonte: Elaborado pelo Autor

Para que o erro da aproximação seja pequeno, as amplitudes a são escolhidas de maneira que as funções discretas tenham o valor próximo ao das funções continuas nos nós do elemento.

O método de Galerkin é um tipo de método de resíduos ponderados, sendo ele o mais utilizado em MEF e também sendo o escolhido para esse trabalho. Nele se tem que N_j é igual a N_i.

Com a substituição das aproximações feitas em 2.15 e 2.16 na equação 2.14, chega-se em um estágio da aplicação do MEF onde se pode reconhecer as matrizes elementares m^e (matriz de massa elementar) e k^e (matriz elementar de rigidez). Para o caso dos elementos tetraédricos utilizados nesse problema, essas matrizes têm dimensão 4x4 e segundo Nithiarasu[1] tem a seguinte composição:

$$m^e = \frac{V^e}{20}\begin{bmatrix} 2 & 1 & 1 & 1 \\ 1 & 2 & 1 & 1 \\ 1 & 1 & 2 & 1 \\ 1 & 1 & 1 & 2 \end{bmatrix}$$

Sendo que V^e é o volume do elemento e uma forma de se calcular esse valor está presente no apêndice A.

A matriz elementar de condutividade térmica pode ser calculada como:

$$k^e = V^e k B^T B$$

Sendo k a condutividade térmica do material e B é uma matriz que depende das coordenadas dos nós e também está definida no Apêndice A.

Neste estágio da execução do MEF se tem um sistema formado pelas matrizes elementares $Ax = b$. O próximo passo corresponde a executar o procedimento de *assembling* para se formar as matrizes globais e assim o sistema englobar todo o domínio do problema.

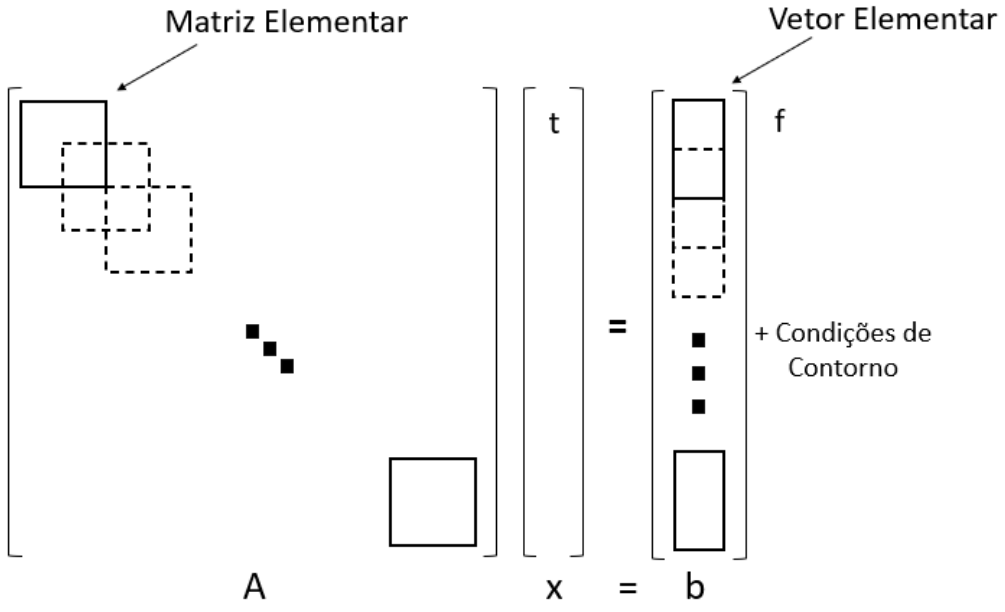


Figura 2.12: Montagem das matrizes elementares em matrizes globais.

Fonte: Elaborado pelo Autor

Para a solução do sistema de matrizes globais é necessário a aplicação das condições de contorno para a transferência de calor que podem ser de Dirichlet,

Neumann ou Robin. Esse último é necessário para a simulação de convecção e consequentemente a única aplicada neste trabalho. O restante do contorno do modelo (os *MOSFET's* IR3575 e a parte inferior da placa do dissipador) não possui nenhuma transferência de calor ou temperatura fixada, sendo então adiabático. No MEF isso é equivalente a não definir nenhuma condição de contorno. O procedimento será explicado durante o Capítulo 3.

A condição de contorno para a sua aplicação no MEF necessita de uma malha, sendo ela uma dimensão inferior ao do problema em questão. No caso de um modelo discretizado com malha tetraédrica, o contorno consiste em elementos triangulares.

A matriz de massa de um elemento triangular é definida como[1]:

$$m_{cc}^{e} = \frac{A_{cc}^{e}}{12} \begin{bmatrix} 2 & 1 & 1 \\ 1 & 2 & 1 \\ 1 & 1 & 2 \end{bmatrix}$$

sendo A_{cc}^{e} a área de cada elemento triangular no contorno.

Com a aplicação das condições de contorno na matriz global se pode, por fim, resolver o sistema resultante do tipo $Ax = b$ e encontrar a variável do problema, nesse caso a temperatura para todo o domínio.

Capítulo 3

Metodologia

3.1 Modelagem do Sistema *MOSFET* Dissipador para a Alimentação de um Processador

A simulação térmica usando MEF do sistema em estudo consiste na situação de maior geração de calor possível nos *MOSFET's* do *VRM* quando os mesmos estão alimentando um determinado processador. Para isso, é simulado que o *CPU* está em seu *PL1* por tempo indeterminado e logo em seguida adentra o *PL2* pelo maior tempo possível. Essa situação garante que os *MOSFET's* estão sendo simulados no caso de maior solicitação pelo processador (*PL2*) e já partindo da máxima temperatura atingida ao trabalhar em *PL1* por tempo indeterminado. O diagrama da simulação pode ser visto a seguir:

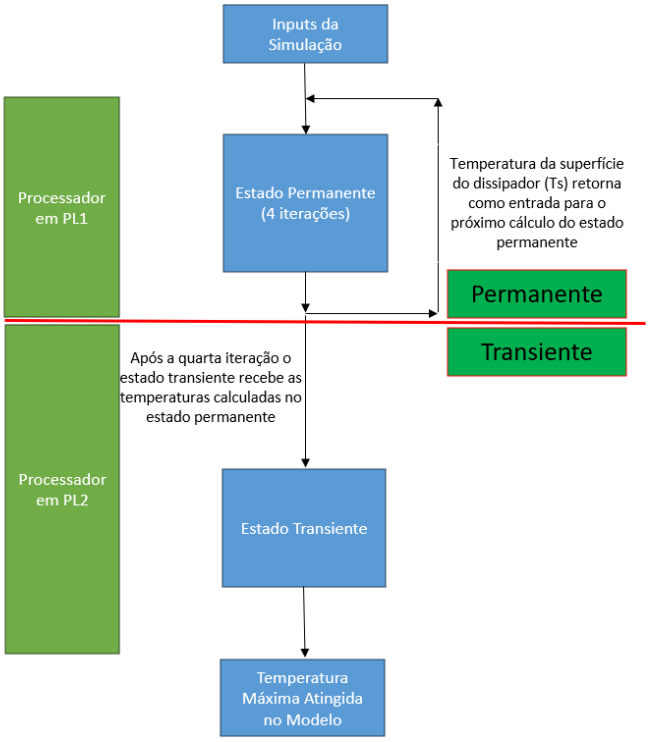


Figura 3.1: Diagrama simplificado da simulação.

Fonte: Elaborado pelo Autor

Como pode ser observado no diagrama da figura 3.1, a fase permanente é computada 4 vezes antes de a temperatura resultante no sistema ser passada para o cálculo da fase transiente. A temperatura de superfície (T_s) é um fator importante para a definição do coeficiente de convecção e como antes de se realizar a simulação permanente não se conhece essa temperatura, o primeiro cálculo é feito supondo-a. Isso potencialmente gera um erro considerável e, por isso, a fase permanente é recalculada com todos os mesmos parâmetros, mas usando-se a T_s encontrada no cálculo anterior. Esse processo é repetido mais 2 vezes, garantindo que a T_s e consequentemente h estejam com valores mais precisos na simulação.

3.1.1 Geometria e Demais Parâmetros

O sistema *MOSFET* dissipador neste trabalho é definido por alguns parâmetros geométricos, de materiais, de demanda elétrica do processador, número de fases e temperatura ambiente. Todas essas características do modelo podem ser alteradas pelo usuário por meio da interface gráfica do programa. Esse aspecto do código é

gerido pelo módulo "Executar", sendo que cada um dos inputs do usuário e quais são seus efeitos no modelo serão apresentados posteriormente na seção 3.3 em conjunto com o restante do código.

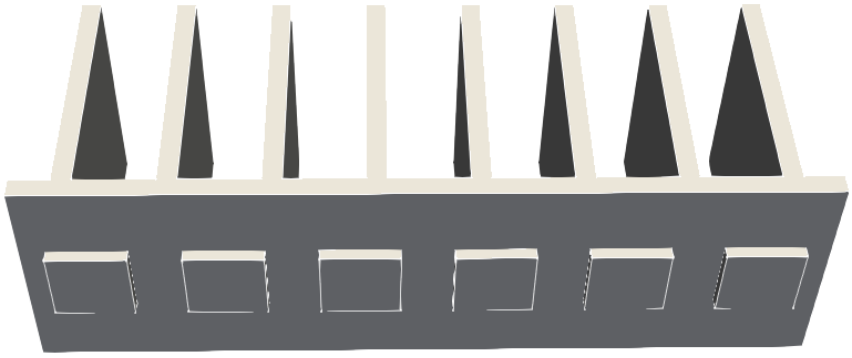


Figura 3.2: Geometria do sistema *MOSFET* acoplado ao dissipador para o caso de referência. Visualização gerada no Software *Paraview*.

Fonte: Elaborado pelo Autor

3.1.2 Hipóteses

Foram adotadas algumas hipóteses e simplificações para a simulação do sistema:

- A placa eletrônica onde os *MOSFET's* estão situados não fazem parte da simulação. Com isso esses componentes só trocam calor com o dissipador;
- A interface entre *MOSFET* e dissipador é considerada perfeita, não sendo simulado a pasta térmica que geralmente faz a união entre os componentes.
- O *MOSFET* IR3575, usado na simulação, é considerado como inteiramente feito de silício;
- O material do dissipador e *MOSFET* são considerados isotrópicos e suas propriedades não variam com a temperatura. No entanto, as propriedades do ar variam na simulação, com o intuito de melhor capturar o efeito da convecção ao longo do aquecimento das paredes do dissipador;
- A tensão do processador é considerada constante em 1,2 volt; além de que a corrente por ele solicitada é dividida igualmente entre as fases do *VRM*;

- A convecção só é aplicada nas aletas e na parte superior da placa do dissipador. As laterais e parte inferior da placa, bem como as partes expostas ao ar dos *MOSFET's* são considerados adiabáticos;

- A posição em que o dissipador está instalado é vertical. A direção das aletas também é vertical.

- Para o cálculo do coeficiente h é desprezado qualquer movimento relativo do ar por fatores externos a simulação. Deste modo, a convecção é considerada puramente natural.

- O efeito da radiação é considerado desprezível.

3.1.3 Geração de Calor

A geração de calor que ocorre nos *MOSFET's* depende da tensão (estipulada em 1,2 Volt) e, principalmente, com a corrente solicitada pelo *CPU*. Esse último varia com a demanda por energia do processador e pelo número de fases do *VRM* que o alimenta:

$$A_f = \frac{W_{pro}}{n_f V_{pro}} \tag{3.1}$$

Sendo A_f a corrente em Amperes por *MOSFET*, W_{pro} a potência consumida em Watts pelo *CPU*, V_{pro} a tensão em Volts solicitada pelo processador e n_f o número de fases do *VRM*. Com a corrente pode-se calcular a geração de calor por meio do gráfico de perda e eficiência energética do *MOSFET* IR3575 3.3.

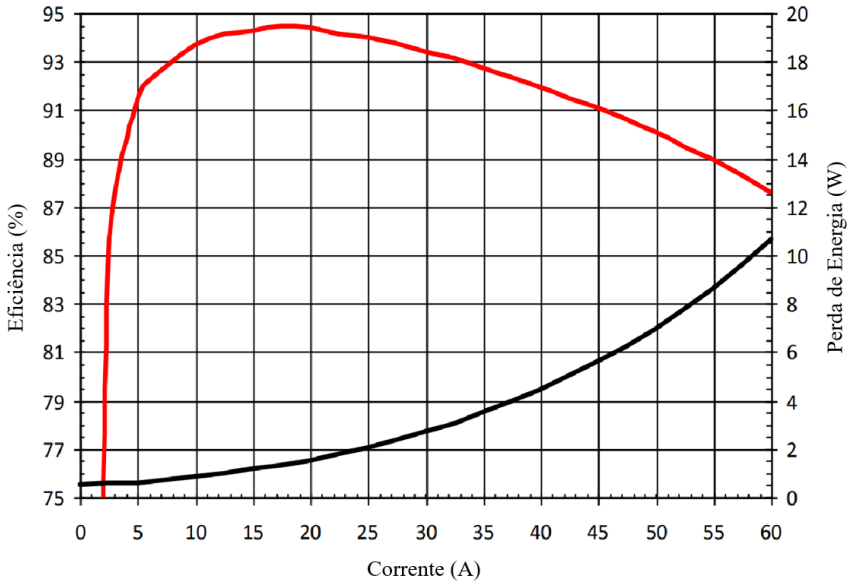


Figura 3.3: Eficiência (em vermelho) e perda de energia (em preto) do *MOSFET* IR3575 em função da corrente quando em 1.2 Volts.[3]

3.1.4 Condições de Contorno

A condição de contorno de convecção aplicada ao sistema em estudo pode ser visualizada na imagem 3.5, consistindo nas aletas e superfície superior da placa de base.

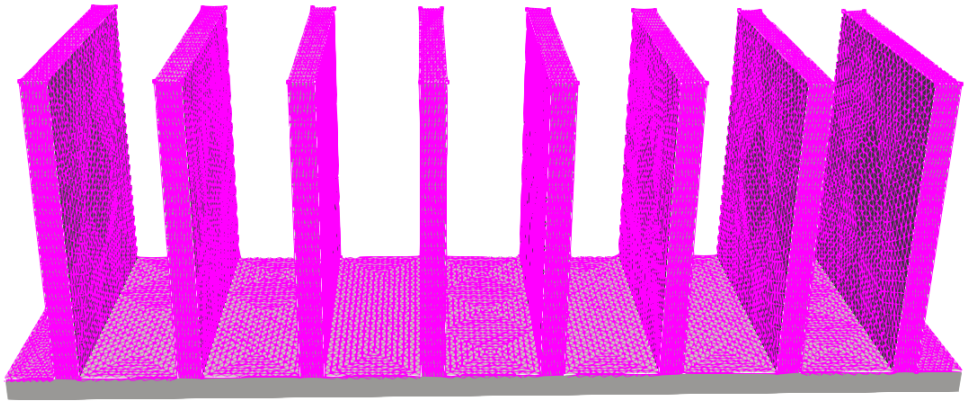


Figura 3.4: Malha do contorno destacada onde a condição de convecção está sendo aplicada. Visualização gerada no *software Paraview*. Vista Superior.

Fonte: Elaborado pelo Autor

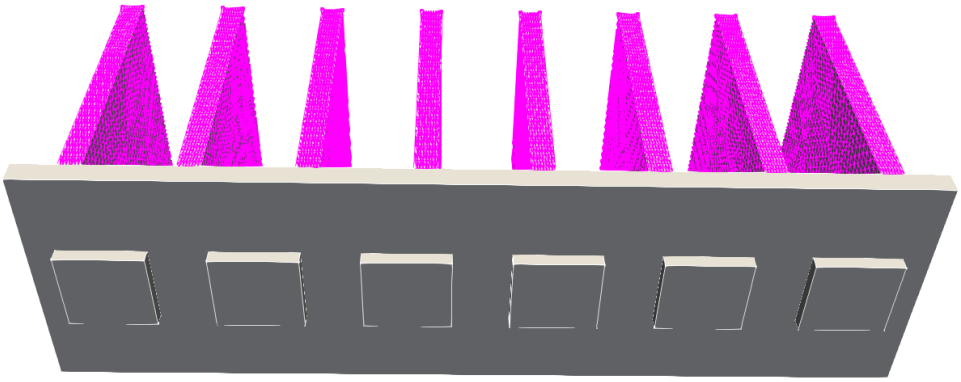


Figura 3.5: Malha do contorno destacada onde a condição de convecção está sendo aplicada. Visualização gerada no *software Paraview*. Vista Inferior.

Fonte: Elaborado pelo Autor

A definição do coeficiente de convecção h, como foi visto na equação 2.3, é necessária para o cálculo desse tipo de transferência de calor. Em função da geometria intricada dos dissipadores, encontrar uma solução analítica para o valor do coeficiente torna-se complexo. Por isso, optou-se por uma equação calibrada por dados experimentais para o cálculo do número de Nusselt (Nu) [5]:

$$Nu = 0.086 Ra^{0.229} \left(\frac{S}{L}\right)^{0.455} \left(\frac{H}{L}\right)^{-0.0112} \left(\frac{t}{L}\right)^{-1.082} n^{-0.119} \tag{3.2}$$

Sendo Ra o número de Rayleigh, $L[m]$ a largura da placa, $W[m]$ o comprimento da mesma, $H[m]$ e $t[m]$ a altura e espessura das aletas, $S[m]$ a distância entre aletas e n o número delas. A validade dessa equação está condicionada ao posicionamento do dissipador e da orientação das aletas, ambas necessitando estar na vertical, como na imagem abaixo:

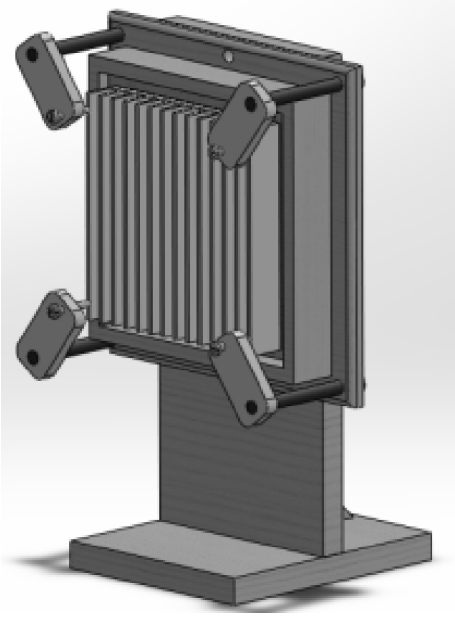


Figura 3.6: Exemplo de um dissipador posicionado verticalmente e com suas aletas também em orientação vertical[5].

Após o cálculo do Nu é possível determinar um $h[W/m^2K]$ para a simulação por meio da equação 3.3 [5].

$$h_{med} = \frac{Nu \cdot k_f}{L} \tag{3.3}$$

Sendo o $k_f[W/mK]$ a condutividade térmica do fluido, neste trabalho o ar. A equação 3.3 em conjunto com a 3.2 não são capazes de definir um h ponto a ponto na superfície do dissipador, só calculando um h médio. Outra limitação está na necessidade de Ra estar em uma determinada faixa para a equação 3.2 ser considerada válida ($2,9 \times 10^5 < Ra < 4,6 \times 10^6$). O cálculo do número de Rayleigh pode ser efetuado por meio da equação [8]:

$$Ra = \frac{g\beta L^3(T_w - T_\infty)}{\upsilon^2} Pr \tag{3.4}$$

Tendo que Pr é o número de Prandall do fluido, T_w e T_∞ a temperatura da parede e do ambiente em Kelvin respectivamente, g $[m/s^2]$ a aceleração da gravidade, β o coeficiente de expansão do fluido e υ $[m^2/s]$ a viscosidade cinemática do mesmo.

A temperatura do fluido com que o dissipador está trocando calor impacta diretamente no coeficiente de convecção e consequentemente na efetividade do dissipador em arrefecer os *MOSFET's*. Isso fica claro ao se observar que há parâmetros físicos

do fluido nas equações 3.3 e 3.4, sendo que essas características variam conforme a temperatura do mesmo (T_{filme}) que é calculada por uma média entre T_w e T_∞. Como a temperatura de T_w não é conhecida inicialmente, se computa o estado permanente 4 vezes para que na última iteração T_{filme} esteja com um valor mais próximo da realidade.

No restante do modelo não foi aplicado condição de contorno nenhuma, o que equivale a essas superfícies não trocarem calor com o ambiente.

3.2 Método de Elementos Finitos

Na seção 2.4 foi apresentado os passos de maneira simplificada e generalizada para a aplicação do MEF. Aqui será exposto às etapas para o uso do método para, especificamente, o caso em estudo.

3.2.1 Geração da Malha

Para a aplicação do Método de elementos finitos é necessário que o modelo do problema que será analisado esteja discretizado. Por isso, um passo preliminar ao MEF consiste na geração de malha para o modelo.

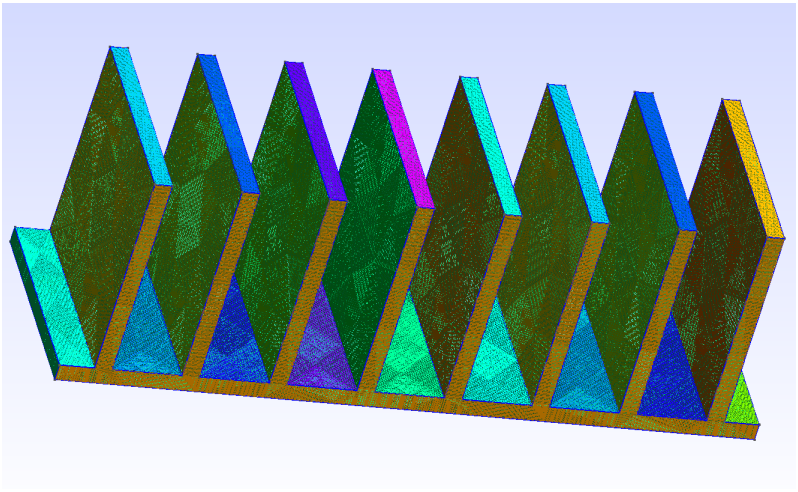


Figura 3.7: Malha gerada pelo código para o sistema *MOSFET* dissipador em estudo.

Essa tarefa é feita pelo Módulo Malha do programa, que é responsável por interpretar as entradas inseridas pelo usuário e construir a geometria e discretiza-la.

Uma apresentação mais detalhada desse e dos demais módulos que compõe o código será feita mais adiante neste capítulo.

3.2.2 Forma Forte e Fraca da Equação

A equação diferencial que descreve o problema, em sua forma forte, consiste na equação de calor que foi deduzida na 2.12:

$$\nabla^2 kT + Q_{gen} = \rho c_p \frac{\partial T}{\partial t} \tag{3.5}$$

A equação acima é válida em um domínio $\Omega \subset \Re$ e sujeita a condição de contorno natural:

$$\nabla kT = h(T_\infty - T) \text{ em } \Gamma \tag{3.6}$$

Como o caso que será calculado possui 2 materiais distintos, temos que o k varia espacialmente. Por isso, o coeficiente de condução térmica deixa de ser uma constante e não pode ser retirado da derivada. O próximo passo consiste na conversão para a forma fraca, ou variacional. Isso é feito por meio da integração no domínio do problema e pela multiplicação por uma função peso:

Sendo $T(\Omega) \in V \mid w(\Omega) \in W$,

$$\int_\Omega w\rho(\Omega)c_p(\Omega)\frac{\partial T}{\partial t}\, d\Omega - \int_\Omega w\nabla^2 k(\Omega)T\, d\Omega - \int_\Omega wQ_{gen}\, d\Omega = 0 \tag{3.7}$$

Sendo V o espaço de funções admissíveis,

$$V = \left(T(\Omega), \int_0^1 \frac{dT}{d\Omega} d\Omega < \infty\right) \tag{3.8}$$

e W o espaço de funções peso,

$$W = \left(w(\Omega), \int_0^1 \frac{dw}{d\Omega} d\Omega < \infty\right) \tag{3.9}$$

Para a redução da ordem do termo $w\nabla^2 kT$ é nele aplicado o teorema de Green:

$$\int_\Omega w\nabla^2 k(\Omega)T\, d\Omega = \int_{\Gamma_{conv}} w\nabla k(\Gamma)T\, d\Gamma - \int_\Omega \nabla k(\Omega) w \cdot \nabla T\, d\Omega \tag{3.10}$$

Substituindo a 3.7 na 3.6, temos que:

$$\int_\Omega w\rho(\Omega)c_p(\Omega)\frac{\partial T}{\partial t}\,d\Omega - \int_{\Gamma_{conv}} w\nabla k(\Gamma)T\,d\Gamma + \int_\Omega \nabla k(\Omega)w\cdot\nabla T\,d\Omega - \int_\Omega wQ_{gen}\,d\Omega = 0 \tag{3.11}$$

3.2.3 Método de Resíduos Ponderados e Montagem das Matrizes

A próxima etapa para a aplicação do MEF é a da transformação das funções diferenciais continuas que temos na forma fraca da 3.11 em discretas. Para isso, é aplicado o método de resíduos ponderados de Galerkin em T, Q e w:

$$T(x,y,z) = \sum_{i=1}^{n} N_i(x,y,z)T_i \tag{3.12}$$

$$Q(x,y,z) = \sum_{i=1}^{n} N_i(x,y,z)Q_i \tag{3.13}$$

$$w(x,y,z) = \sum_{j=1}^{n} N_j(x,y,z)w_j \tag{3.14}$$

Substituindo na equação 3.11:

$$\int_\Omega \sum_{i=1}^{n}\sum_{j=1}^{n} \rho(\Omega)c_p(\Omega)N_iN_j\frac{dT_i}{dt}\,d\Omega - \int_{\Gamma_{conv}} w\nabla k(\Gamma)T\,d\Gamma + \\ + \int_\Omega \sum_{i=1}^{n}\sum_{j=1}^{n} \nabla k(\Omega)N_i\cdot\nabla N_jT_i\,d\Omega - \int_\Omega \sum_{i=1}^{n}\sum_{j=1}^{n} N_iN_jQ_i\,d\Omega = 0 \tag{3.15}$$

As substituições no segundo termo, correspondente a condição de contorno, serão realizadas mais adiante.

A seguir, podemos fazer a seguinte manipulação na expressão presente na 3.15:

$$\int_\Omega \sum_{i=1}^{n}\sum_{j=1}^{n} N_iN_j = \sum_i\sum_j \int_{\Omega^0} N_iN_j + \sum_i\sum_j \int_{\Omega^1} N_iN_j + ... + \sum_i\sum_j \int_{\Omega^{ne}} N_iN_j \tag{3.16}$$

$$\int_\Omega \sum_{i=1}^{n}\sum_{j=1}^{n} N_iN_j = m_0^e + m_1^e + ...m_{ne}^e = M \tag{3.17}$$

Ao se realizar a manipulação para se ter um somatório de integrais cujo domínio abrange somente um elemento da malha cada, pode-se encontrar as matrizes de massa elementares para o problema. Ao se realizar o *assembly* com todos os elementos, se encontra a matriz global de massa (M).

Para encontrar a matriz global de rigidez (K), um procedimento similar é realizado:

$$\int_{\Omega} \sum_{i=1}^{n} \sum_{j=1}^{n} \nabla k(\Omega) N_i \cdot \nabla N_j \, d\Omega = \int_{\Omega} k(\Omega) \left(\frac{\partial N_i}{\partial x} \frac{\partial N_j}{\partial x} + \frac{\partial N_i}{\partial y} \frac{\partial N_j}{\partial y} \right) d\Omega \tag{3.18}$$

$$\int_{\Omega} k(\Omega)(x_{ij} + y_{ij}) \, d\Omega = K \tag{3.19}$$

Apesar de k não ser constante espacialmente em todo o domínio, ele é invariável dentro de cada elemento da malha. Por isso, como a matriz K é montada se resolvendo as integrais com domínios por elemento, foi retirado o k da derivada como se o mesmo fosse constante na 3.18.

Substituindo as matrizes globais M e K na 3.15 resulta em:

$$M\rho c_p \frac{dT_i}{dt} - \int_{\Gamma_{conv}} w \nabla k(\Gamma) T \, d\Gamma + KT_i - MQ_i = 0 \tag{3.20}$$

$$M\rho c_p \frac{dT_i}{dt} + KT_i = \int_{\Gamma_{conv}} w \nabla k(\Gamma) T \, d\Gamma + MQ_i \tag{3.21}$$

3.2.4 Condição de Contorno de Convecção

Para se aplicar o método de resíduos ponderados no termo da condição de contorno é necessário primeiro substituir ∇kT pela expressão que descreve o fenômeno da convecção:

$$\nabla kT = h(T_{\infty} - T) \tag{3.22}$$

Substituindo na 3.21:

$$M\rho c_p \frac{dT_i}{dt} + KT_i = \int_{\Gamma_{conv}} wh(T_{\infty} - T) \, d\Gamma + MQ_i \tag{3.23}$$

$$M\rho c_p \frac{dT_i}{dt} + KT_i + \int_{\Gamma_{conv}} whT = \int_{\Gamma_{conv}} whT_\infty \, d\Gamma + MQ_i \tag{3.24}$$

Aplicando o método de resíduos ponderados aos termos referentes a convecção:

$$M\rho c_p \frac{dT_i}{dt} + KT_i + hM_{cont}T_i = hM_{cont}T_\infty + MQ_i \tag{3.25}$$

Sendo M_{cont} encontrado da mesma maneira que a Matriz de Massa M foi anteriormente. A diferença entre às duas consiste no domínio, sendo que a primeira engloba a superfície de contorno de convecção e a última o modelo inteiro. Como M_{cont} abrange o domínio de uma superfície, os elementos usados para calcular as suas matrizes elementares para o *assembly* são triangulares. Para simplificar a 3.25, podemos definir:

$$M_{conv} = hM_{cont} \tag{3.26}$$

$$M\rho c_p \frac{dT_i}{dt} + KT_i + M_{conv}T_i = M_{conv}T_\infty + MQ_i \tag{3.27}$$

$$M\rho c_p \frac{dT_i}{dt} + (K + M_{conv})T_i = M_{conv}T_\infty + MQ_i \tag{3.28}$$

3.2.5 Diferenças Finitas e Forma Final do Sistema

O problema em estudo neste trabalho possui etapas não só permanentes, como transientes. Com isso, a derivada no tempo na equação 3.28 não pode ser igualada a zero para a realização da simulação. Por isso, a próxima etapa para a aplicação do MEF consiste em discretizar temporalmente pelo Método de Diferenças Finitas (MDF) o termo $\frac{dT_i}{dt} = T_i'(t)$.

O MEF consiste em um método numérico que realiza aproximações em derivadas por uma fórmula encontrada por meio da Série de Taylor. Para o caso da derivada temporal de interesse, temos que:

$$\frac{dT_i}{dt} = \frac{T_i'(t + \Delta t) - T_i'(t)}{\Delta t} \tag{3.29}$$

Sendo Δt o passo temporal do programa, em segundos.

Substituindo a 3.29 na 3.28 resulta em:

$$M\rho c_p \frac{T_i'(t + \Delta t) - T_i'(t)}{\Delta t} + (K + M_{conv})T_i = M_{conv}T_\infty + MQ_i \tag{3.30}$$

Desse modo, temos um sistema $Ax = b$ e a sua solução torna conhecida a temperatura nos *MOSFET's* e dissipador.

3.3 Algoritmo Computacional

O *software* desenvolvido neste trabalho tem como missão resolver por MEF o estado térmico de um sistema *MOSFET* acoplado a dissipador com vários parâmetros disponíveis para modificação pelo usuário. A simulação para ser executada necessita tanto do cálculo do estado permanente quanto do transiente. Desse modo, o caso estudado possui um grau de complexidade alto e o código necessita ser também similarmente complexo. Para facilitar o desenvolvimento e compreensão, o programa foi dividido em 5 módulos, cada um lidando com um aspecto bem definido da simulação por MEF. O funcionamento de cada módulo e seu papel no código será apresentado ao longo da seção 3.3.

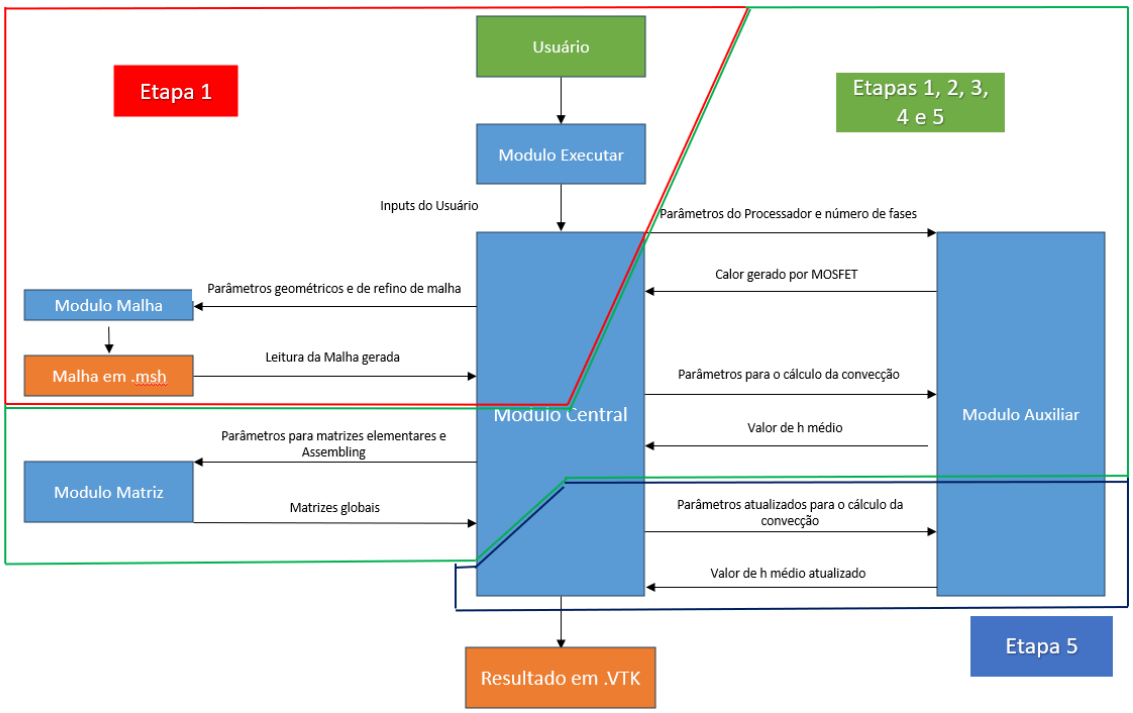


Figura 3.8: Diagrama de funcionamento do código. Os módulos estão destacados, bem como quais deles são utilizados em cada etapa do *software*.

Fonte: Elaborado pelo Autor

A execução do código é dividida em 5 etapas, as 4 primeiras sendo a solução

do caso permanente e a final sendo do caso transiente. Esse último caso parte das temperaturas encontradas para o modelo na quarta etapa, no caso permanente. Há certas diferenças entre quais módulos são utilizados e o funcionamento do código entre certas etapas:

- Etapa 1, permanente. Nela ocorre o envio das entradas ao Módulo Central, a geração da malha e das matrizes globais, além da aplicação do solver, se encontrando uma temperatura preliminar para o modelo no caso permanente. Nas próximas etapas não haverá mais a necessidade de se ler as entradas, nem se gerar uma nova malha.

- Etapa 2, 3 e 4, permanente. Em cada uma dessas etapas se geram novas matrizes globais e se aplicam novamente o solver, se encontrando temperaturas para o modelo no caso permanente. O resultado encontrado com cada etapa é mais preciso que o anterior, uma vez que a temperatura de superfície é um parâmetro importante para a definição do coeficiente de convecção médio (h_{med}). Na primeira etapa, esse valor tem de ser suposto, gerando mais erro nas temperaturas obtidas. As etapas subsequentes podem utilizar a temperatura de superfície calculada anteriormente e com isso obtêm valores cada vez mais precisos.

- Etapa 5, transiente. Nesse estágio final, ocorre novamente a definição das matrizes globais como nos estágios anteriores, mas a aplicação do solver difere. Como o estado passa a ser transiente, o solver precisa ser aplicado pelo número de iterações do caso. Para cada uma delas o h_{med} é recalculado com as novas temperaturas encontradas e as matrizes são devidamente atualizadas. Vale ressaltar que a temperatura inicial do caso transiente corresponde ao resultado do caso permanente calculado na etapa 4. Por fim, os resultados são salvos em um arquivo.VTK.

3.3.1 Módulo Executar

O Módulo Executar é responsável pela interface gráfica do programa, permitindo que o usuário insira com facilidade as entradas para a simulação. A biblioteca utilizada para isso é a *tkinter* que permite a construção de interfaces simples. Após

a inserção de todas as entradas e o usuário apertar o botão "Rodar", o módulo é responsável por enviar esses dados ao Módulo Central, onde se inicia a simulação.

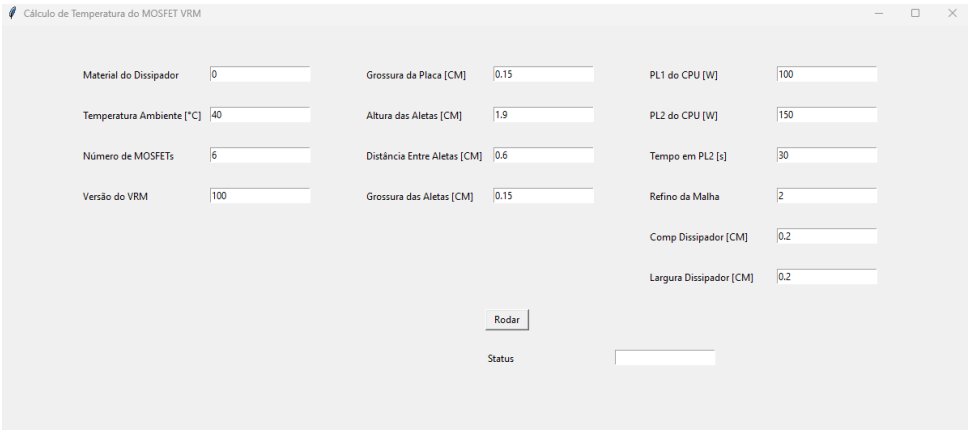


Figura 3.9: Interface gráfica do *software* desenvolvido com inputs inseridos para o caso de referência.

Fonte: Elaborado pelo Autor

Como pode ser visto na interface gráfica na figura 3.9, as entradas do programa são as seguintes:

- Material do dissipador, sendo que o número 0 representa o alumínio 1350 e o 1 representa o cobre 10100.
- Temperatura ambiente $[C]$, temperatura do ar no ambiente em que o dissipador se encontra.
- Número de *MOSFET's*, em outras palavras, número de fases do *VRM*. Como a distância entre *MOSFET's* é fixada em 0,4 cm, esse parâmetro também influencia no comprimento do dissipador.
- Versão do *VRM*, número que ficará no nome do arquivo .VTK resultante para que se possa reconhecer posteriormente os resultados de diferentes simulações.
- Grossura da placa [cm], espessura da placa base do dissipador.
- Altura das Aletas [cm]
- Distância entre aletas [cm], indiretamente define o número de aletas no dissipador, em conjunto com os parâmetros que definem o comprimento do dissipador.

- Grossura das Aletas [cm], também indiretamente define o número de aletas no dissipador, mas em menor grau.
- *PL1* do *CPU* [w], define a potência consumida pelo *CPU* capaz de ser mantida por ele por tempo indeterminado.
- *PL2* do *CPU* [w], define a potência consumida máxima do *CPU*, sendo capaz de mantê-la por tempo determinado.
- Tempo em *PL2*[s], consiste no tempo em segundos que o *CPU* é capaz de se manter em *PL2*.
- Refino da Malha, define o quão refinado será a malha gerada para o modelo. Valores como 0 correspondem a malha menos refinada possível e 4 como a malha mais refinada que pode ser computada em MEF em tempo hábil (por volta de 1 dia).
- Comp Dissipador [cm], corresponde ao quanto, em centímetros, o dissipador excede os *MOSFET's* que se encontram nas extremidades. Em conjunto com o número de *MOSFET's*, define o comprimento do dissipador.
- Largura Dissipador [cm], define o quão mais largo o dissipador é do que o MOSFET.

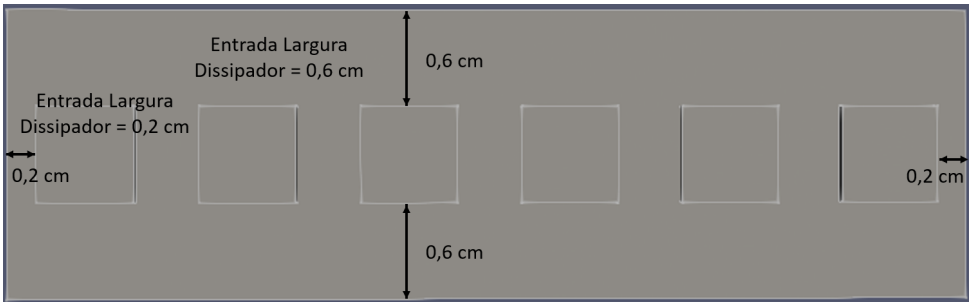


Figura 3.10: Efeito no tamanho do dissipador das entradas "Comp Dissipador [cm]"e "Largura Dissipador [cm]".

Fonte: Elaborado pelo Autor

3.3.2 Módulo Central

O módulo central, como pode ser visto na imagem 4.7, faz o papel de eixo principal do código, sendo o único interagindo com os demais módulos e onde o sistema que é encontrado com o MEF é propriamente solucionada. Também é onde se realiza a configuração para que o programa realize o cálculo do caso permanente por 4 vezes (para maior precisão) e logo em seguida se calcule o caso transiente.

Listing 3.1: Módulo Central - Controle de etapas do programa

```
cycle = 1
for i in range(0,5):
    if cycle==1: #No primeiro ciclo considera como Ts=75 graus celcius
        Ts=60
        teta = 1
    if cycle>=2: #Nos outros ciclos considera a temperatura real de um
        ponto na condicao de contorno de conveccao
        Ts=T[IENbound[1][0]]
    if cycle==1: #Calcula 4 vezes o caso permanente para maior
        precisao no Ts e consequentemente h_med
        mode = 0
    if cycle==5: #Ciclo 5 e o transiente, com maior carga no processador
        mode = 1
        it = 15
        dt = tPL2/it
        T = T_per
...
...
...
    cycle +=1
```

O Módulo, em sua maioria, reside dentro de um *loop* tipo *for*, sendo que cada laço representa uma etapa da simulação. O número de cada laço é registrado na variável *cycle*, que se inicia com 1 e termina o programa como 5. Os valores 1, 2, 3 e 4 correspondem aos 4 cálculos do caso permanente, sendo possível ver no trecho do código acima que o primeiro *loop* define a temperatura de superfície (T_s)

arbitrariamente. Isso prejudica a precisão do cálculo do coeficiente de convecção médio e consequentemente as temperaturas obtidas na simulação. Por isso, os 3 subsequentes cálculos do estado permanente calculam a exata mesma situação que o primeiro, mas com o valor T_s mais preciso obtido pela solução anterior.

A variável *mode* controla o estado da etapa da simulação, o valor 0 definindo como permanente e 1 como transiente. Como pode ser visto acima no código, o *cycle* 5 corresponde a etapa do estado transiente. Nela é definida que a temperatura inicial nos pontos do modelo do sistema é a mesma calculada com a última etapa permanente. Além disso, se estipula que o número de iterações (it) é fixo em 15 e que o passo (dt) depende do tempo máximo que o processador sustenta o seu $PL2$, no código chamado de $tPL2$

Listing 3.2: Trecho Módulo Central - Solver

```
for i in range(it):

    print ("Ciclo de solucao de Matrix:"+str(i+1))
    b = ((M)@(T/dt))*mode + MQ - ((K+M_cc)@((1-teta)*T)) + (M_cc@T_inf)

    T = spsolve(A,b) #resolvendo matriz esparsa
    if cycle==5:
        Ts=T[IENbound[1][0]]
        h_med_antigo=h_med #salvando o valor antigo de h_med
        h_med = misc.h_calc(H,n,W,L,S,Ts,T_inf,t) #recalculando h_med para
            cada nova temperatura no estado transiente
        M_cc=M_cc*(h_med/h_med_antigo) #modificando M_cc com o novo h_med
    A = (((M)/dt)*mode) + (teta*(K+M_cc)) #Recriando A para o proximo
        ciclo com o M_cc atualizado
    if cycle==5:
        T_per = T #Definido a temperatura do estado permanente como ponto
            de partida do estado transiente
```

Para cada uma das 5 etapas da simulação se aplica o solver para se solucionar as equações do modelo. Para isso, um solver de matrizes esparsas da biblioteca *scipy* é utilizado. Nas etapas permanentes, o número de iterações (it) é definido como 1

e, consequentemente, o *loop* contido no código acima tem somente um laço. Com isso, nessas situações, somente é montado o vetor b e aplicado o solver, se chegando então nas temperaturas em cada ponto do modelo para os casos permanentes.

Na etapa transiente, *it* tem valor superior a 1 acarretando em vários laços no *loop*, levando ao vetor b ser calculado e o solver ser aplicado pelo número de iterações do caso transiente. Outro ponto relevante é de que o valor de *cycle* é 5, o que acarreta em se adentrar na condicional *if cycle==5*. Isso significa que a cada iteração do caso transiente se recalcula um novo valor de h_{med} se utilizando a temperatura encontrada. Com o novo h_{med}, se atualiza a matriz M_{cc} e com isso se recalcula o valor de A, sendo o seu valor atualizado utilizado no solver da próxima iteração. Isso é feito para assegurar um valor de h_{med} mais próximo à realidade e garantir uma maior precisão na simulação.

Listing 3.3: Trecho do Módulo Central - Resultado

```
#Mostrar resultados
T_max_cord = np.argmax(T) #encontrar indice da temperatura mais elevada
T_max = T[T_max_cord] #Temperatura mais elevada
print ("Temperatura maxima no VRM: "+str(T[T_max_cord]))

if cycle==5:
    point_data = {"temp!":T}
    meshio.write_points_cells(f"Teste_conducao"+str(ver)+".vtk"
    ,msh.points,msh.cells,point_data=point_data,)
    return
        ("-#####################################################################")
cycle +=1
```

Com o fim de cada etapa do código, tanto as permanentes quanto a transiente, após o cálculo da temperatura em cada vértice do modelo é informado ao usuário a maior temperatura encontrada por meio do console do *Python*. Na etapa 5 (cycle 5), o resultado encontrado é salvo em um arquivo .VTK por meio da biblioteca *meshio*.

3.3.3 Módulo Malha

O Módulo Malha se utiliza do programa *Gmsh* através de uma biblioteca homônima para controlar a geometria do modelo e o refino da malha. Consequentemente, isso permite que o usuário modifique esses aspectos da simulação por meio das entradas por ele inseridas, apesar do uso de um programa externo na geração da malha. O módulo está dividido em 2 funções, o *mesh-generation* e o *get-mesh*.

A *mesh-generation* tem como objetivo a geração da geometria e malha do sistema *MOSFET* dissipador especificado pelo usuário. A geometria é construída, em essência, por meio da função gmsh.model.occ.addBox(x,y,z,dx,dy,dz,tag). Ela tem como resultado a geração de uma geometria de um paralelepípedo reto conforme os tamanhos e coordenadas especificados. Toda a geometria do modelo é construída por essa maneira, como os *MOSFET's*, a placa base do dissipador e as aletas. A posterior fusão de todos os paralelepípedos gerados em um único modelo é feita por meio da função gmsh.model.occ.fuse(). Após a sincronização do programa com o *Gmsh*, se realiza a geração da malha.

Por meio da função gmsh.model.mesh.generate(3) a malha tridimensional para a geometria é gerada, mas em um estado muito grosseiro. Para se realizar o refino, a função gmsh.model.mesh.refine() é utilizada, sendo ela chamada quantas vezes o usuário entender ser necessário para uma malha adequada. É importante ressaltar que a ação da função afeta a toda malha, não havendo a possibilidade de se refinar somente áreas de interesse.

Listing 3.4: Trecho da função mesh-generation

```
# Gerando a malha
gmsh.model.mesh.generate(3) #Malha 3D
if refino>=1:
    for i in range(1,refino+1):
        gmsh.model.mesh.refine()

# Salvando a malha
gmsh.write("dissipador.msh")
```

A malha resultante é salva no arquivo dissipador.msh e posteriormente a função get-mesh é utilizada para lê-lo para que os seus dados possam ser organizados e de fácil acesso.

Listing 3.5: Trecho da função get-mesh

```
msh = ms.read("dissipador.msh") #Lendo a malha gerada no mesh_generation
points = msh.points
data = msh.cells_dict
IENbound = data["triangle"]
IEN = data["tetra"]
X = [];Y = []; Z = []
#passando por todos os pontos e guardando as coordenadas X, Y e Z
    separadamente
for e in range(len(points)):
    X.append(points[e][0]);Y.append(points[e][1]);Z.append(points[e][2])
```

A função get-mesh organiza a IEN e IENbound por meio do tipo de elemento que compõe cada uma delas, além de salvar as posições de cada ponto separadamente para a direção X, Y e Z. Também é gerada uma IENBound mais restritiva, contendo somente os elementos que estarão sujeitos a condição de contorno de convecção. Esses dados são então repassados ao Módulo Central para que a simulação possa continuar.

3.3.4 Módulo Matrix

O módulo tem o objetivo de gerar as matrizes locais e globais tanto para os elementos tetraédricos da malha, quanto para os que compõe o contorno do modelo (triangulares). Esse último é necessário para se impor as condições de contorno, no caso deste projeto, somente convecção.

Listing 3.6: Trechos do Módulo Matrix - Malha local referente ao contorno de convecção

```
for e in range(0,len(IENbound_conv1)):
    [v1,v2,v3] = IENbound_conv1[e]
    cv_med = ((cv[IEN[e][0]]) + (cv[IEN[e][1]]) + (cv[IEN[e][2]]))/3.0
        #Definindo o cv medio como a media entre os cv dos 3 pontos
    rho_med = ((rho[IEN[e][0]]) + (rho[IEN[e][1]]) +
    (rho[IEN[e][2]]))/3.0 #Definindo o rho medio como a media entre os
        rho dos 3 pontos

    ...
    ...
    ...

    melem_tri = (A_tri/(12.0*cv_med*rho_med))*np.array([ [2,1,1],
        [1,2,1], [1,1,2] ])
```

Para a correta construção das matrizes elementares é necessário assegurar que as propriedades do material usadas na formulação da matriz sejam constantes dentro dela. Para isso, antes da montagem da matriz para cada elemento, se calcula uma média entre as propriedades dos pontos em cada elemento. Isso é necessário para que células que estejam posicionadas na fronteira entre materiais sejam corretamente calculadas. Esse procedimento é realizado tanto para a matriz elementar de massa triangular para contorno, como pode ser visto no trecho do código acima, quanto para a matriz de massa e rigidez dos elementos tetraédricos.

Para que o Módulo possa montar as matrizes globais, é necessário realizar o procedimento de *assembly*.

Listing 3.7: Trechos do Módulo Matrix - Assembling das matrizes globais tetraédricas

```
K = lil_matrix((npoints, npoints),dtype = "double")
M = lil_matrix((npoints, npoints),dtype = "double")
...
...
```

```
...
for e in range(0,ne):
...
...
...
    for ilocal in range(0,4):
        iglobal = IEN[e,ilocal]
        for jlocal in range(0,4):
            jglobal = IEN[e,jlocal]
            K[iglobal,jglobal] += kelem[ilocal,jlocal]
            M[iglobal,jglobal] += melem[ilocal,jlocal]
...
...
...
M = M.tocsr()
K = K.tocsr()
```

Para se realizar o *assembly* é feito um *loop* para cada elemento da malha, se calculando a sua matriz elementar e inserindo esses valores na matriz global (traduzindo as coordenadas locais para globais). O código para a montagem das matrizes globais para elementos tetraédricos K e M podem ser vistas a cima. O trecho responsável pelo *assembly* para as matrizes elementares foi suprimido pelo seu tamanho. Contudo, o código completo, com todos os seus módulos, está presente no anexo deste trabalho.

A montagem e posterior solução da equação matricial é um procedimento que exige grande capacidade de memória e processamento. No entanto, grande parte dos elementos das matrizes globais são compostos de valores 0. Tendo isso em conta, é vantajoso a utilização de matrizes esparsas, uma vez que elas são capazes de ocupar muito menos memória que as comuns. No código acima é possível ver a criação das matrizes globais por meio da biblioteca *scipy*. Dois tipos distintos de matrizes esparsas são utilizadas, a tipo *lil* e a tipo *csr*. Isso foi feito, pois, com a primeira, é mais rápido modificar seus elementos e, desse modo, é a mais adequada para a realização do *assembly*. Em contrapartida, a *csr* é menos custosa para se realizar

cálculos. Por isso, ocorre a conversão para esse tipo de matriz antes de elas serem enviadas ao Módulo Central para posterior aplicação do *solver*.

3.3.5 Módulo Auxiliar

O módulo tem 2 objetivos, que são atingidos por meio das duas principais funções que o compõe. A primeira tarefa do código corresponde ao cálculo da geração de calor por parte dos *MOSFET's*, feita pela função:

Listing 3.8: Trechos do Módulo Auxiliar - Função vrm_w

```
def vrm_w(vrm_n,cpu_v,cpu_pl1,cpu_pl2,cycle):
    cpu_w = cpu_pl1 #estado normal do cpu
    if cycle>=5:
        cpu_w= cpu_pl2 #Estado temporario em pl2
    vrm_amp=cpu_w/(vrm_n*cpu_v)
    dados =
        [0,0.915,0.938,0.945,0.945,0.940,0.936,0.928,0.920,0.911,0.901,0.890,0.876]
        #Eficiencia do VRM/MOSFET IR3575

    n = math.floor((vrm_amp)/5)
    val_1 = dados[n]
    val_2 = dados[n+1]
    aux1 = (vrm_amp - (5*n))/5
    vrm_efic = ((val_1*(1-aux1))+(val_2*(aux1)))
    vrm_w = (1-vrm_efic)*cpu_v*vrm_amp
    q = vrm_w/(0.6*0.6*0.09)
    print ("Valor de q: "+str(q))
    return (q)
```

A função acima se inicia com o cálculo da corrente que está sendo solicitado de cada *MOSFET*. Isso é feito se considerando qual etapa se está na simulação, permanente (*PL1*) ou transiente (*PL2*) e assim definindo o consumo energético do *CPU* de acordo. Posteriormente se é calculado a corrente solicitada em cada *MOSFET* por meio da equação 3.1. Com esse dado em mãos se encontra a eficiência

de operação do componente e consequentemente a geração de calor do mesmo.

A outra missão do código corresponde ao cálculo do coeficiente de convecção médio (h_{med}), sendo a função $h_{calc}()$ utilizada para esse fim. O cálculo do h_{med} é executado como foi descrito na seção 3.1.4, se calculando o Número de Rayleigh por meio da 3.4, o Número de Nusselt pela 3.2 e por fim o coeficiente médio pela 3.3. As características físicas do fluido no código são funções da temperatura de filme e esse aspecto é considerado no cálculo de h_{med}. A função h_{calc} também considera a limitação da equação utilizada para o cálculo de Nu:

Listing 3.9: Trechos do Módulo Auxiliar - Função vrm_w

```
Ra = Gr*Pr #Numero de Rayleigh
print ("Ra: "+str(Ra))
if Ra>=2.9*10**5 and Ra<=4.6*10**6:
    Ra_flag=0
if Ra_flag==1:
    print ("Erro - Modulo_Matrix - h_calc")
    return (1)
```

Caso o número de Rayleigh não estiver na faixa aceitável para o cálculo de Nu, a função irá avisar o usuário da ocorrência do problema e irá retornar ao Módulo Central, não o h_{med}, mas o valor 1 que será interpretado como um erro e resultará na terminação do programa.

Capítulo 4

Verificação

Para o correto funcionamento do código em calcular a temperatura dos Módulos Reguladores de tensão nas situações propostas, o programa foi testado em todos os aspectos necessários ao problema. São eles:

1. Operar com mais de 1 material. O dissipador e os *MOSFET's* são de materiais distintos.

2. Condição de contorno de convecção. Necessário para simulações com dissipadores de calor.

3. Calcular casos permanentes e transientes. Necessário em função do regime de trabalho dos processadores.

4. Geração de calor interna. Os *MOSFET's* são fontes de geração durante seu funcionamento.

5. Condução. Há condução dentro e entre componentes do sistema.

Para realizar as validações dos itens 1, 3, 4 e 5, foi realizada algumas comparações entre os resultados do código com um programa comercial, o *Ansys*, tanto para um caso permanente quanto transiente. Para o item 2, foi utilizado uma comparação com um problema com solução analítica.

4.1 Validação da Condução Para 2 Materiais Distintos

4.1.1 Parâmetros da Simulação

O caso utilizado para a validação é de um paralelepípedo reto de 400mm de largura e altura e 100mm de profundidade, sendo usado o programa comercial *Ansys* para comparação. A geometria é composta de 2 materiais distintos, *Cooper Alloy* na metade a esquerda e *Structural Steel* a direita, ambos presentes na biblioteca do *Ansys*. As temperaturas das duas superfícies laterais e a inferior foram definidas como 10°C e a superfície superior como 100°C. As demais superfícies, as faces da placa, foram definidas como adiabáticas. Foi realizado testes com esses parâmetros tanto para o caso permanente quanto para o caso transiente em 10, 20, 40, 80 e 160 segundos. O número de iterações em cada uma das simulações transientes foi igual para o programa desenvolvido e o *Ansys*.

4.1.2 Comparação de Resultados

A comparação entre as temperaturas encontradas no código desenvolvido e no *software* comercial *Ansys* foi realizada em 3 pontos em todos os casos propostos. As coordenadas desses pontos estão listadas na tabela 4.1 e os eixos estão definidos na figura 4.1.

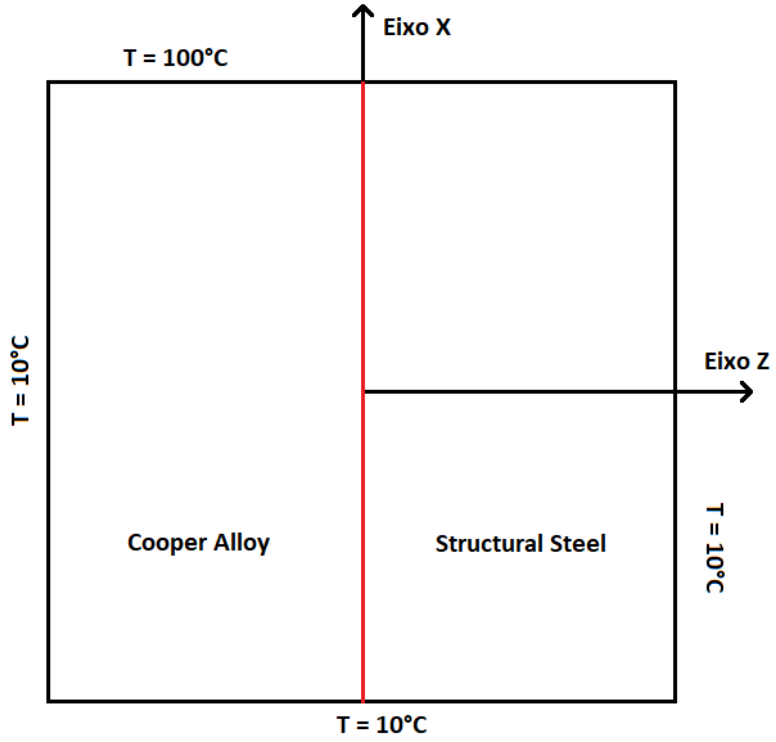


Figura 4.1: Condição de contorno para o caso da validação da condução com 2 materiais distintos. A linha vermelha corresponde a divisão dos 2 materias no sólido.

Fonte: Elaborado pelo Autor

	X(mm)	Y(mm)	Z(mm)
Ponto 1	148,507	100,000	-108,431
Ponto 2	149,090	100,000	107,461
Ponto 3	133,967	100,000	26,703

Tabela 4.1: Posição dos pontos de prova para as comparações.

Segue abaixo algumas imagens com a visualização das temperaturas superficiais do *Ansys* e do *software* desenvolvido para algumas das simulações. Também está presente as tabelas com os resultados encontrados para as temperaturas.

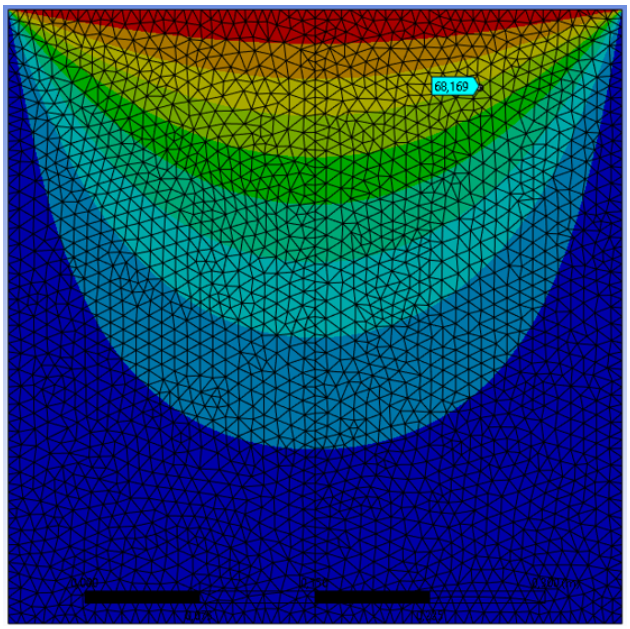


Figura 4.2: Visualização da simulação no *Ansys* para o caso permanente da condução com 2 materiais.

Fonte: Elaborado pelo Autor

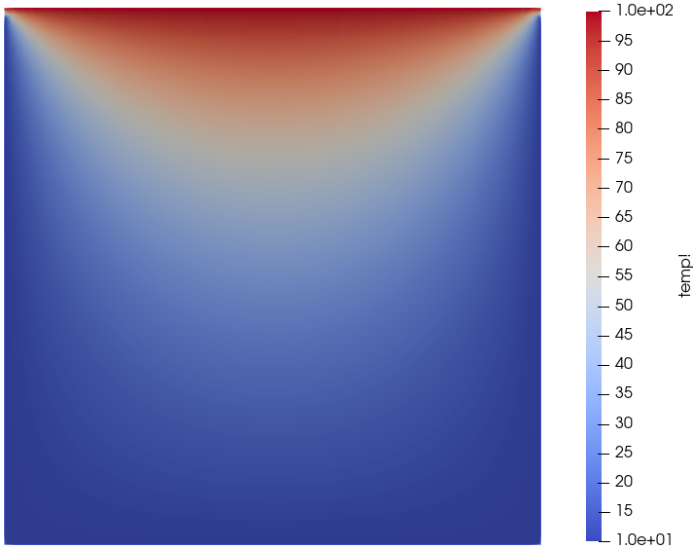


Figura 4.3: Visualização no *Paraview* do resultado do código para o caso permanente da condução com 2 materiais.

Fonte: Elaborado pelo Autor

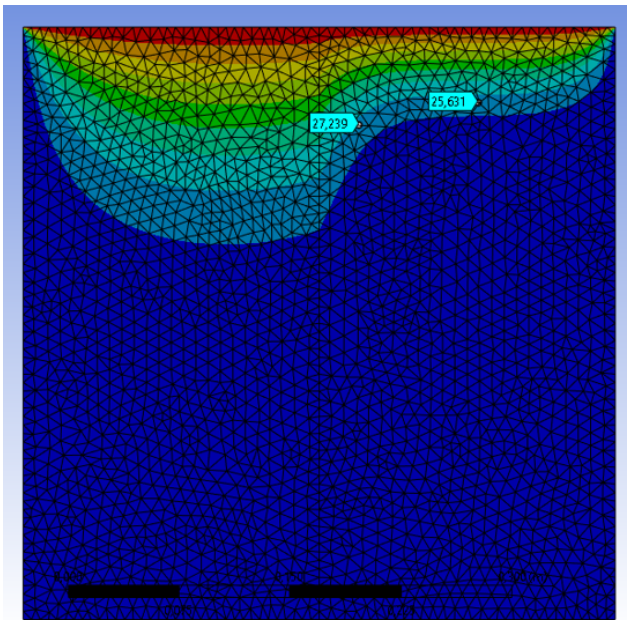


Figura 4.4: Visualização da simulação no *Ansys* para o caso transiente da condução com 2 materiais. Tempo em 40 segundos.

Fonte: Elaborado pelo Autor

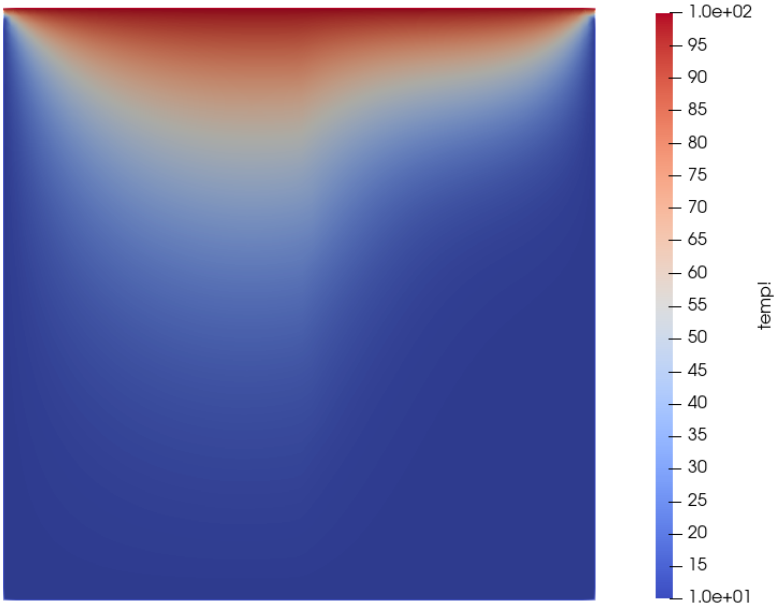


Figura 4.5: Visualização no *Paraview* do resultado do código para o caso transiente da condução com 2 materiais. Tempo em 40 segundos

Fonte: Elaborado pelo Autor

Tempo(s)	10			20			40		
Ponto	1	2	3	1	2	3	1	2	3
Programa	35,72°	10,57°	10,55°	48,20°	14,95°	15,22°	57,67°	25,52°	27,74°
Ansys	35,76°	10,87°	10,81°	48,22°	15,23°	15,37°	57,72°	25,63°	27,24°
Erro	0,04°	0,30°	0,26°	0,02°	0,28°	0,15°	0,05°	0,11°	0,50°
Iterações	10			10			20		

Tabela 4.2: Tabela de comparação de resultados em 10, 20 e 40 segundos para o caso teste da condução com 2 materiais.

Tempo(s)	80			160			Permanente		
Ponto	1	2	3	1	2	3	1	2	3
Programa	63,01°	39,86°	43,64°	65,82°	53,66°	58,08°	67,61°	68,13°	70,74°
Ansys	62,93°	39,88°	44,44°	65,78°	53,79°	58,66°	67,67°	68,17°	70,77°
Erro	0,08°	0,02°	0,80°	0,04°	0,13°	0,58°	0,06°	0,04°	0,03°
Iterações	20			32			N/A		

Tabela 4.3: Tabela de comparação de resultados em 80, 160 e no estado permanente para o caso teste da condução com 2 materiais.

Os resultados indicam que o código está obtendo temperaturas próximas quando comparado ao *Ansys*. No entanto, há uma grande discrepância nos erros do ponto 3 quando comparado aos outros 2. Isso se explica devido à proximidade desse ponto a interface dos materiais e a maneira com que ela é tratada no código. No programa desenvolvido, quando um ponto da malha se encontra inserido em um material, ele assume as propriedades do mesmo. Em contrapartida, quando o mesmo está na exata interface entre materiais ele assume uma média entre as propriedades. Na geração das matrizes elementares para o Método de Elementos Finitos, é utilizada uma média dos valores das propriedades dos pontos que o compõe. Como a maneira como a interface é lidada difere entre programas, é esperado que a maior divergência entre o código e o *Ansys* ocorra perto da transição entre materiais. Também é possível perceber que o erro é muito menor no caso permanente quando comparado ao transiente.

4.2 Validação da Geração de Calor

4.2.1 Parâmetros da Simulação

Para a validação do programa em geração de calor, foi utilizado uma geometria de uma placa com um pequeno paralelepípedo gerador de calor acoplado em uma das superfícies. Essa geometria foi escolhida para se poder aproveitar o gerador de malha criado para o projeto, gerando assim um dissipador de calor sem aletas e com um único *MOSFET* em uma das extremidades. As dimensões da placa são de 101 mm de comprimento, 26 mm de largura e 2 mm de profundidade. As dimensões da fonte geradora de calor são de 6 mm de comprimento e largura, além de 0,9 mm de profundidade. A taxa de geração de calor é de $10^7 W/m^3$, gerando no total $3{,}24W$. O material usado para todo o modelo é o *Structural Steel* presente no *Ansys*. As temperaturas das bordas da placa foram fixadas em 10 graus e as demais superfícies foram definidas como adiabáticas. O caso transiente foi calculado para 3 tempos distintos: 2,5 segundos; 5 segundos e 10 segundos, utilizando 10 iterações no código e no *Ansys*. Também foi calculado o caso permanente para comparação, seguindo os mesmos parâmetros do caso transiente.

4.2.2 Comparação de Resultados

A comparação entre as temperaturas encontradas na ferramenta proposta e no *software* comercial *Ansys* foram realizadas em 2 pontos, tanto no caso permanente, quanto no transiente. Um desses pontos é simplesmente a maior temperatura registrada nas simulações e logo não possui uma coordenada específica. Já o outro tem coordenada fixa e está localizado na superfície próximo ao canto inferior esquerdo do *MOSFET*, mas em lados opostos no modelo.

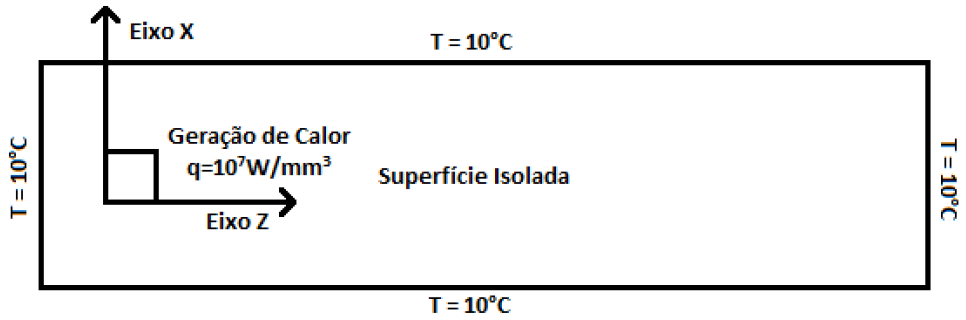


Figura 4.6: Condição de contorno para o caso de validação da geração de calor.

Fonte: Elaborado pelo Autor

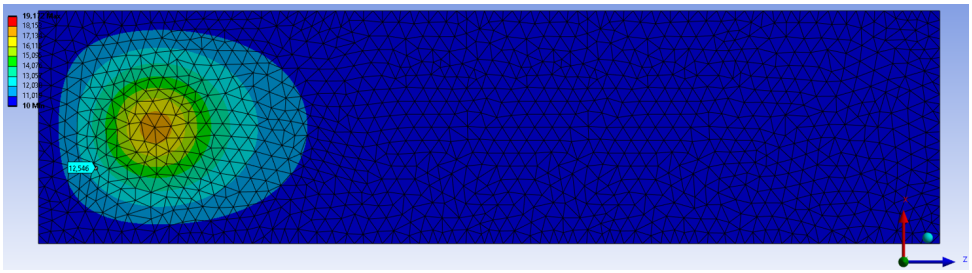


Figura 4.7: Visualização no *Ansys* para a simulação de validação no caso permanente. Posição do ponto cuja temperatura foi usada para comparação está visível próximo a fonte de calor. *MOSFET* está acoplado na superfície oposta na placa.

Fonte: Elaborado pelo Autor

Figura 4.8: Visualização no *Paraview* para a simulação de validação. Tempo em 10 segundos. *MOSFET* está acoplado na superfície oposta na placa.

Fonte: Elaborado pelo Autor

Tempo(s)	2,5		5	
Ponto	1	MAX	1	MAX
Programa	11,48°	16,95°	12,17°	18,20°
Ansys	11,41°	17,01°	12,08°	18,26°
Erro	0,07°	0,06°	0,09°	0,06°
Iterações	10		10	

Tabela 4.4: Tabela de comparação de resultados em 2.5 e 5 segundos para o caso teste da geração de calor.

Tempo(s)	10		Permanente	
Ponto	1	MAX	1	MAX
Programa	12,53°	18,90°	12,64°	19,11°
Ansys	12,44°	18,96°	12,55°	19,17°
Erro	0,09°	0,06°	0,09°	0,06°
Iterações	10		N/A	

Tabela 4.5: Tabela de comparação de resultados em 10 segundos e no estado permanente para o caso teste da geração de calor.

Percebe-se que as temperaturas encontradas estão muito próximas, tanto no ponto 1, quanto na temperatura máxima, ambas não chegando a uma diferença de 0,1°C. Os erros permanecem no mesmo patamar nas simulações transientes e permanentes.

4.3 Validação para Condição de Contorno de Convecção

4.3.1 Parâmetros da Simulação

Para a validação da implementação da convecção, foi utilizado os parâmetros do exemplo 3.3 do Ozisik[8]. O exemplo consiste em uma geometria de uma chapa de 0,01 m, condutividade térmica de $20W/m°C$, coeficiente de convecção

de $4000W/m^2C$, geração de calor de $8 \times 10^7 W/m^3$ e temperatura ambiente de $100°C$. O problema foi modelado como unidimensional, sendo uma das extremidades adiabática e a outra com convecção. No código, como ele foi desenvolvido para modelos tridimensionais, a geometria foi gerada como um cubo de 0,01 m de lado. Cinco das superfícies do cubo foram consideradas adiabáticas enquanto a superfície restante foi definida com uma condição de contorno de convecção, tornando o problema equivalente a um unidimensional. O exemplo usado abarca unicamente o caso permanente e o código também foi utilizado para calcular a temperatura no mesmo estado.

4.3.2 Comparação de Resultados

A distribuição de temperatura na chapa, segundo o exemplo 3.3 do Ozisik[8], segue a equação:

$$T(x) = \frac{gL^2}{2k} + \left[1 - \left(\frac{x}{L}\right)^2\right] + \frac{gL}{h} + T_\infty \tag{4.1}$$

Usando a equação acima, foi calculado 8 pontos ao longo do eixo x na chapa, sendo os valores encontrados inseridos na Tabela 4.6, junto com as suas posições e os valores encontrados pelo código para comparação.

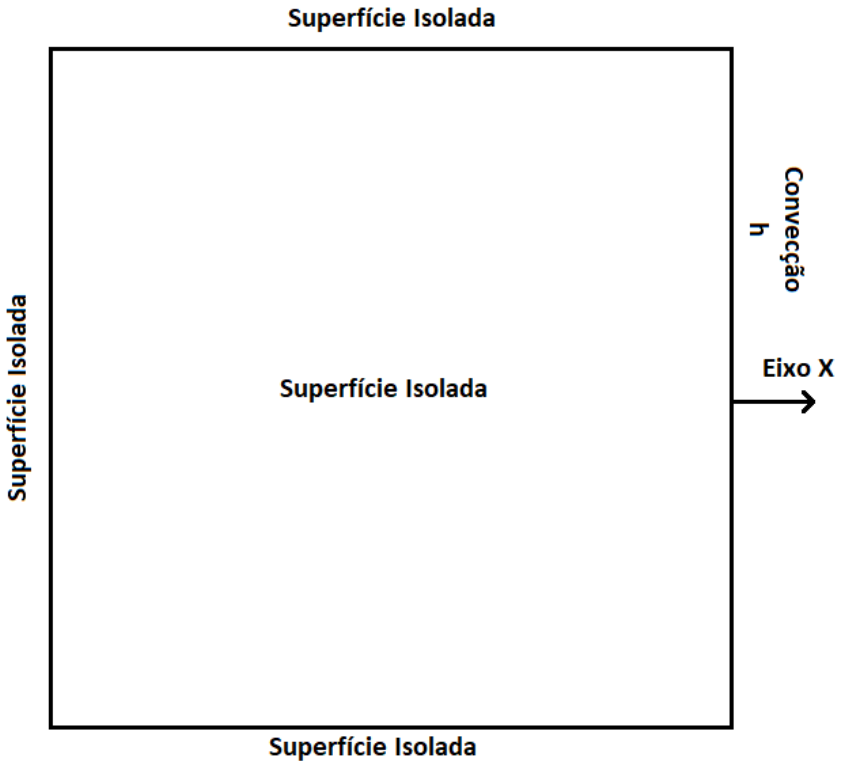


Figura 4.9: Condição de contorno para o caso da validação de convecção.

Fonte: Elaborado pelo Autor

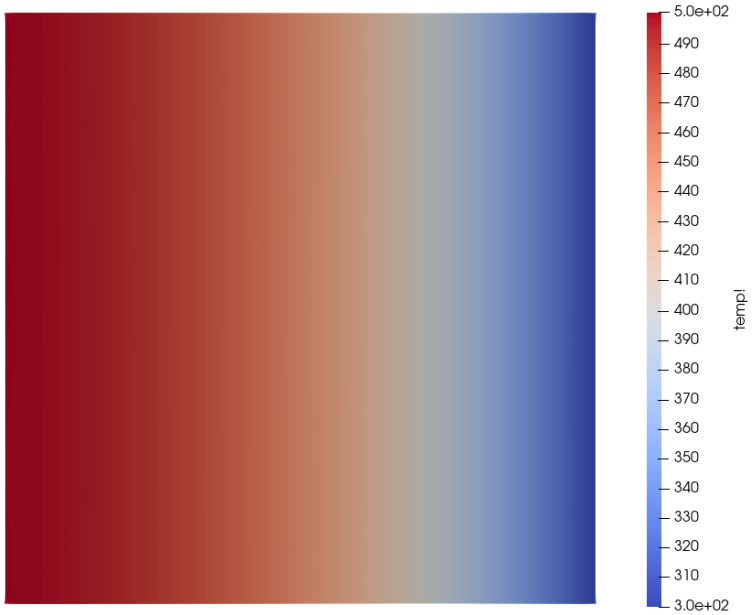


Figura 4.10: Visualização no *Paraview* do resultado do código para o caso teste da convecção. Estado permanente.

Fonte: Elaborado pelo Autor

Posição	Programa	Analítico	Erro
10,00mm	299,95°	300,00°	0,05°
9,64mm	314,08°	314,12°	0,04°
8,23mm	364,62°	364,70°	0,08°
6,62mm	412,54°	412,46°	0,08°
5,10mm	447,89°	447,89°	0,00°
3,59mm	474,18°	474,26°	0,08°
1,91mm	492,68°	492,74°	0,06°
0,00mm	499,98°	500,00°	0,02°

Tabela 4.6: Tabela de comparação de resultados para 8 pontos ao longo do caso teste unidimensional da convecção.

Como pode ser visto na Tabela 4.6, as temperaturas encontradas pelo código desenvolvido nesse projeto são coerentes com os resultados da solução analítica para o problema. Essa comparação demonstra que todos os aspectos do Método de Elementos Finitos envolvidos nesse teste estão corretamente implementados no programa, sendo um deles a condução.

Capítulo 5

Resultados

Neste capítulo serão apresentados os resultados obtidos com o código desenvolvido. O programa *Paraview* foi utilizado para visualizar as temperaturas superficiais encontradas. Serão apresentadas 5 simulações diferentes, 1 caso de referência e outros 4 que apresentam alguma mudança em um ou mais parâmetros em relação ao primeiro.

Os casos testados têm como objetivo reproduzir o procedimento de dimensionamento preliminar de um *VRM* e dissipador acoplado, o que corresponde a finalidade para o qual o código foi desenvolvido. Para tal é importante que, na pior das condições, os *MOSFET's* não excedam sua temperatura operacional em nenhum ponto. No caso do IR3575 usado no programa, essa temperatura é de $95°C$. Muitas das características geométricas, de materiais e elétricas são mantidas iguais entre os casos para facilidade de visualização do impacto que outras variáveis têm na temperatura máxima atingida. As variações nesses parâmetros em cada simulação são mais expressivas do que testes de dimensionamento normalmente seriam. Isso é feito também para deixar claro o impacto das mudanças na temperatura.

São mantidas inalteradas entre as simulações:

- Temperatura ambiente de $40°C$.
- Consumo energético do processador em estado constante ($PL1$) em $100W$.
- Consumo energético do processador em estado transiente ($PL2$) em $150W$.
- Tempo de sustentação de $PL2$ em 30 segundos.

- Material do dissipador em alumínio 1350.
- Material do *MOSFET* em silício, sendo que o usuário não pode modificar essa escolha de material pela interface gráfica.
- Espessura da placa e aletas em 1,5 mm.

Os demais parâmetros, como altura e quantidade de aletas, número de fases do *VRM* e tamanho do dissipador variam entre as simulações.

O computador utilizado para se realizar as simulações possui as seguintes configurações:

- Processador Intel 12700h
- 16GB de Ram DDR5 operando a 4800MHZ
- Sistema operacional *Windows 11 Home* versão 22H2

A versão dos programas utilizados pelo programa desenvolvido são:

- *Python* - Versão 3.11.0
- *Gmsh* - Versão 4.11.1
- *Paraview* - Versão 5.11.0

5.1 Caso 1 - Referência

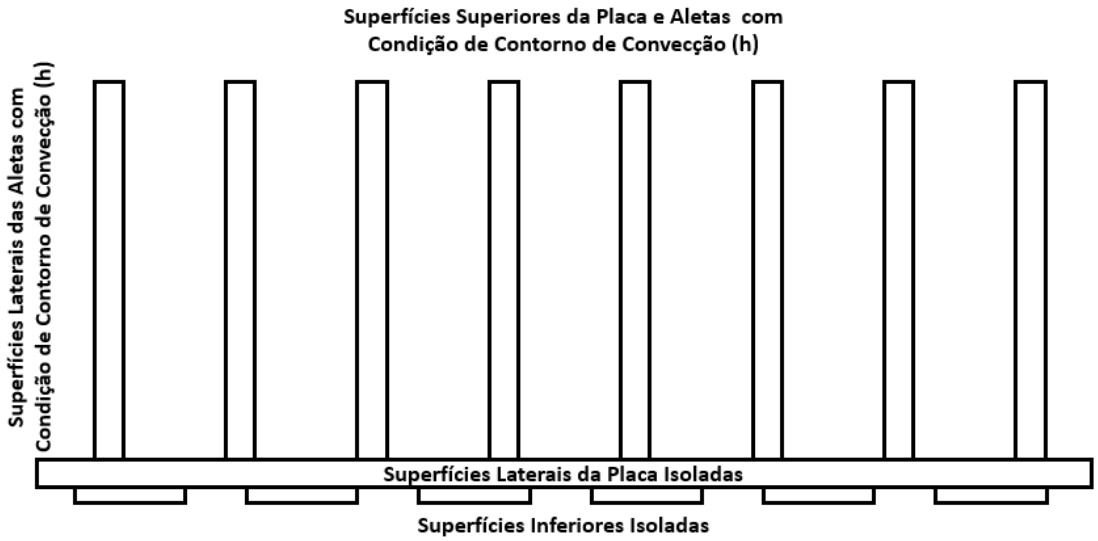


Figura 5.1: Condição de contorno para o caso 1. Desenho fora de escala.

Fonte: Elaborado pelo Autor

Os parâmetros para o caso de referência são:

- 6 Fases no *design* do *VRM*, com *MOSFET's* IR3575;
- Comprimento do dissipador de 60mm;
- Largura do dissipador de 18mm;
- Altura das aletas de 19mm;
- Número de aletas de 8.

O programa desenvolvido, como explicado em detalhes no capítulo 3, simula um estado permanente e utiliza a temperatura resultante como ponto de partida para o cálculo do estado transiente. As temperaturas máximas ao longo desse cálculo para o caso de referência estão expostas a seguir:

Caso 1 - Referência - Permanente	
Permanente Ciclo 1	$81,54°C$
Permanente Ciclo 2	$76,46°C$

Permanente Ciclo 3	$77,01°C$
Permanente Ciclo 4	$76,96°C$

Tabela 5.1: As temperaturas máximas calculadas pelo programa nos 4 ciclos do estado permanente.

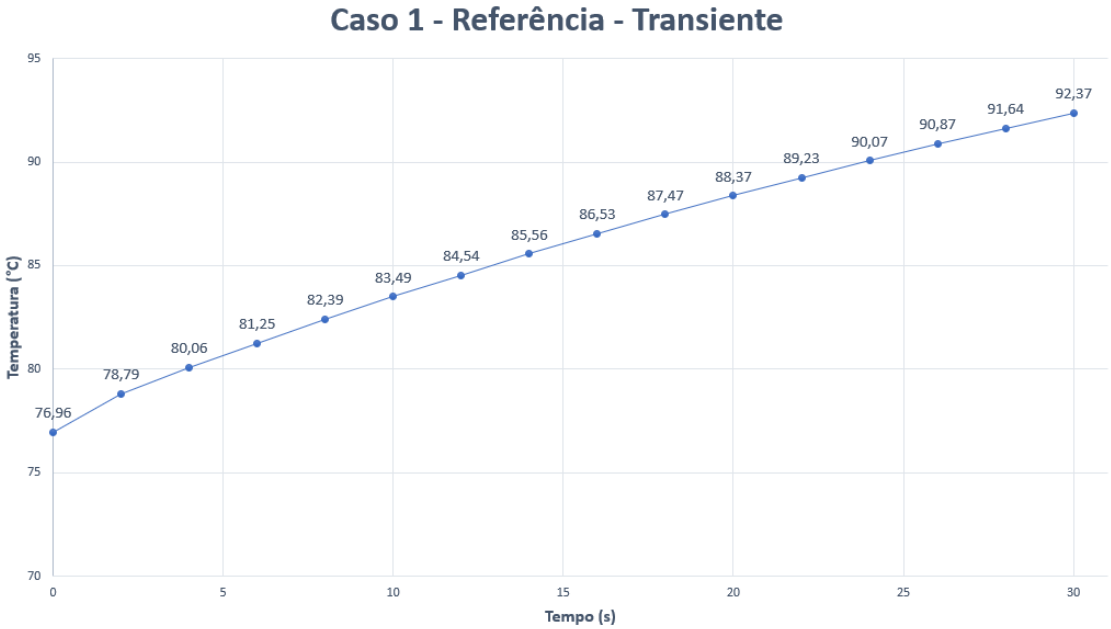


Figura 5.2: Visualização da temperatura máxima encontrada nos *MOSFET's* ao longo do tempo na simulação.

A figura 5.14 mostra a evolução da temperatura ao longo do tempo, partindo do resultado permanente em 0 segundo e terminando em 30 segundos. Esse aumento ocorre em função do incremento na geração de calor por parte dos *MOSFET's* dada a entrada do processador em seu *PL2*. Nos próximos casos, as temperaturas intermediárias na simulação não serão mais expostas, uma vez que para o dimensionamento do *VRM* e dissipador só interessa a temperatura máxima atingida.

Referente as temperaturas nos ciclos permanentes na Tabela 5.1, é possível ver que há uma diferença considerável no resultado entre o ciclo 1 e 4. Isso deixa claro que há um impacto significativo na temperatura máxima atingida na etapa permanente da simulação ao se reintroduzir recursivamente a informação da temperatura superficial nos cálculos subsequentes. Há uma rápida convergência, havendo somente necessidade de 4 ciclos para um bom resultado. Abaixo, na Tabela 5.2, encontram-se

alguns dados relevantes da simulação:

Caso 1 - Referência	
Área de Convecção [cm^2]	70,08
Eficiência de *MOSFET* em *PL1*	94,3%
Geração de Calor Interna *PL1* [W]	5,70
Temperatura Máxima Permanente [°C]	76,96
Eficiência de *MOSFET* em *PL2*	94,4%
Geração de Calor Interna *PL2* [W]	8,40
Temperatura Máxima Transiente [°C]	92,37
Número de Elementos	668.904

Tabela 5.2: Dados referentes a simulação.

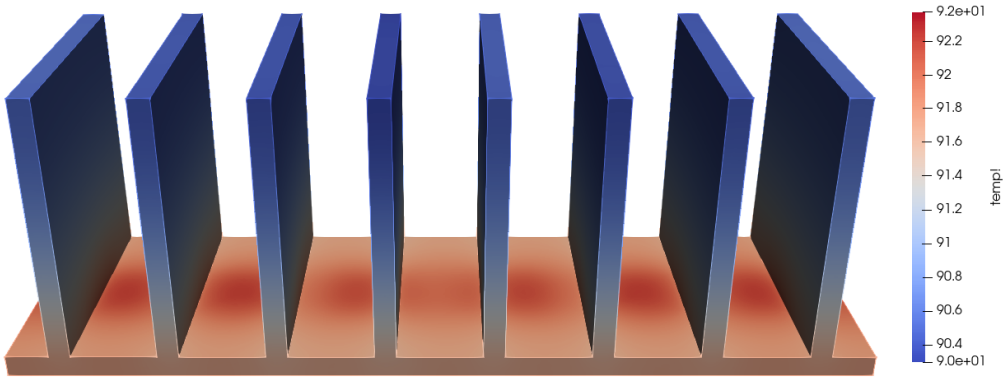


Figura 5.3: Visualização da simulação no caso de referência no *Paraview*, vista superior.

Fonte: Elaborado pelo Autor

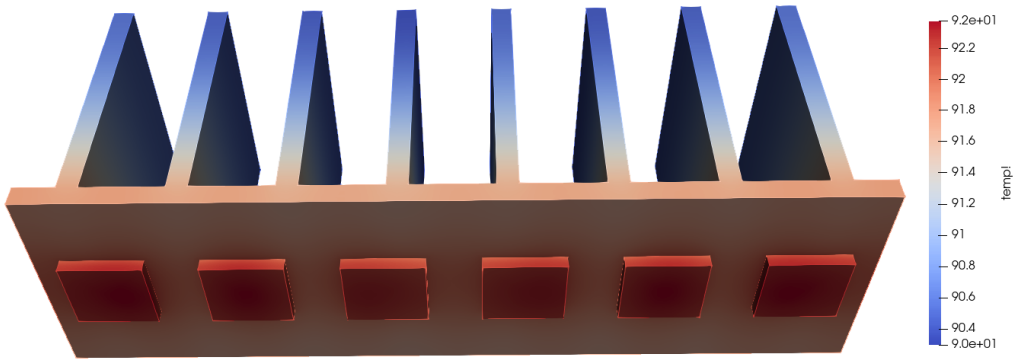


Figura 5.4: Visualização da simulação no caso de referência no *Paraview*, vista inferior.

Fonte: Elaborado pelo Autor

A maior temperatura calculada pelo programa nessa simulação é de $92,37°C$, localizado no *MOSFET* mais à direita na imagem 5.4. É possível visualizar que há uma diferença de temperatura entre esses componentes, sendo os mais frios presentes no centro do dissipador e os mais quentes nos cantos. Uma explicação possível para essa situação é o posicionamento das aletas em relação a cada *MOSFET*. No entanto, a diferença de temperatura entre os componentes eletrônicos são muito pequenas, todos estando na faixa de $92°C$. Como os componentes IR3575 operam muito próximos de sua temperatura máxima operacional de $95°C$, em um caso real, haveria problemas no funcionamento do computador em momentos de grande demanda de energia do processador. Dependendo do mecanismo de segurança da placa mãe, ela reduziria a frequência do processador, degradando o desempenho do mesmo, ou desligaria o computador.

5.2 Caso 2 - Redução para 5 Fases

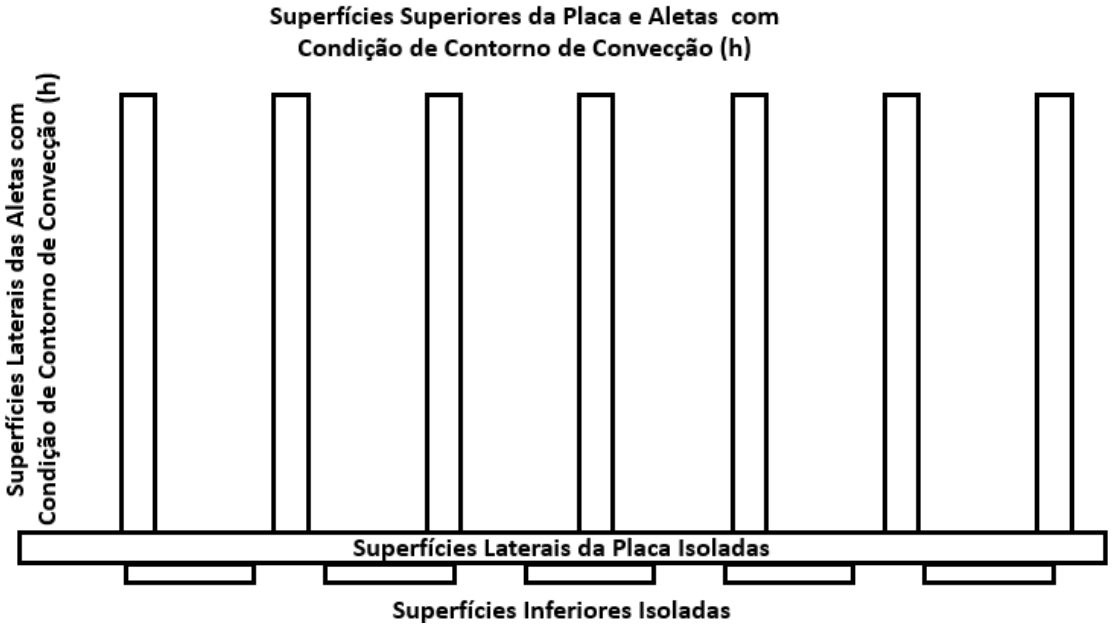


Figura 5.5: Condição de contorno para o caso 2. Desenho fora de escala.

Fonte: Elaborado pelo Autor

Parâmetros para a simulação do caso 2:

- 5 Fases no *design* do *VRM*, com *MOSFET's* IR3575;
- Comprimento do dissipador de 50 mm;
- Largura do dissipador de 18 mm;
- Altura das aletas de 19 mm;
- número de aletas de 7.

Abaixo, na Tabela 5.3, encontram-se alguns dados relevantes da simulação do caso 2:

Caso 2	
Área de Convecção [cm^2]	60,87
Eficiência de *MOSFET* em *PL1*	94,5%
Geração de Calor Interna *PL1* [W]	5,50

Temperatura Máxima Permanente [°C]	84,06
Eficiência de *MOSFET* em *PL2*	94,0%
Geração de Calor Interna *PL2* [W]	9,00
Temperatura Máxima Transiente [°C]	105,06
Número de Elementos	949.512

Tabela 5.3: Dados referentes a simulação.

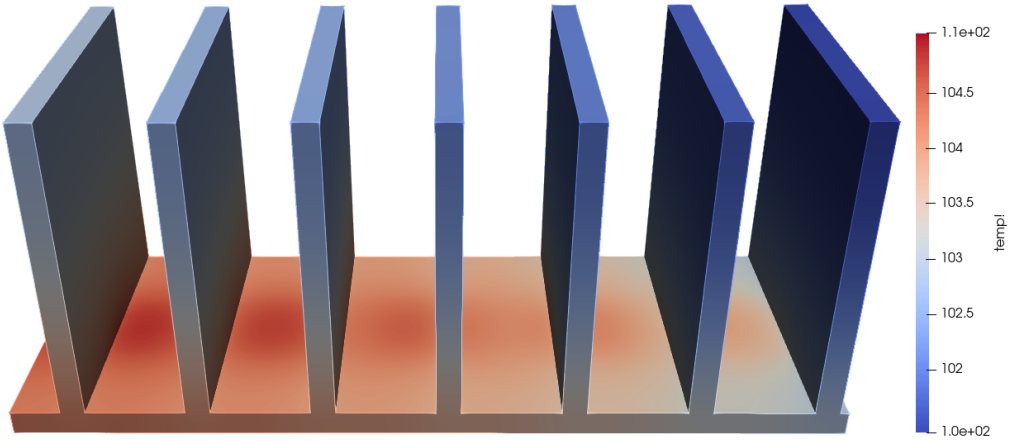


Figura 5.6: Visualização da simulação no caso 2 no *Paraview*, vista superior.

Fonte: Elaborado pelo Autor

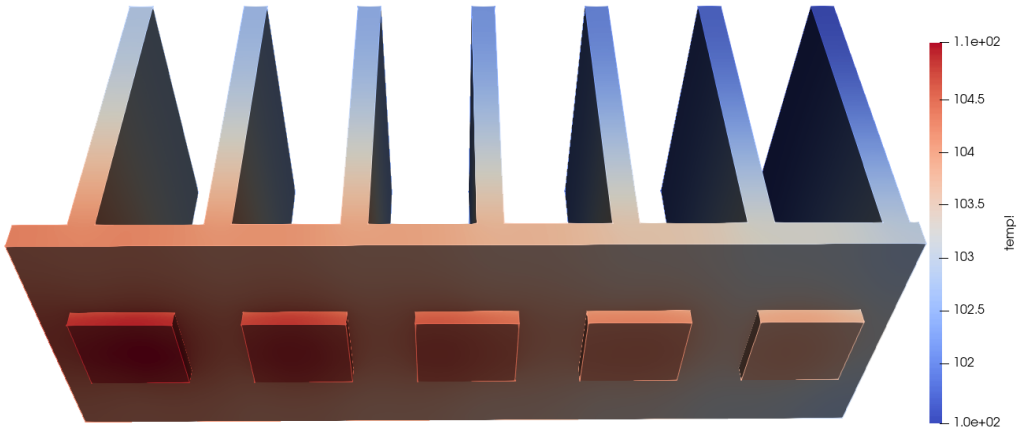


Figura 5.7: Visualização da simulação no caso 2 no *Paraview*, vista inferior.

Fonte: Elaborado pelo Autor

A maior temperatura obtida pelo programa nessa simulação com número de fases

e comprimento de dissipador reduzido é de $105,06°C$, localizado no *MOSFET* mais à esquerda na imagem 5.7. Desse modo, segundo a simulação, fica claro que uma redução no número de fases e área de convecção tem um impacto significativo na temperatura máxima, extrapolando o limite de $95°C$. Com uma frequência maior que na simulação 1, o mecanismo de segurança da placa mãe seria utilizado para que a temperatura limite do IR3575 não seja atingida. O aumento na geração de calor em cada *MOSFET* dado uma redução de 6 para 5 fases se deve por dois fatores. O primeiro é a menor eficiência com a qual os componentes estão operando, resultando em maior geração de calor total. O segundo fator é a maior concentração na geração de calor, sendo suportada a demanda do processador por corrente em somente 5 *MOSFET's*.

Como na simulação 1, é possível ver que o posicionamento das aletas novamente tem influência nas temperaturas locais. O componente a direita está cerca de 1 grau mais frio que o da esquerda, possivelmente em função do primeiro estar melhor posicionado, tendo 2 aletas próximas. Mas a pequena variação de temperatura mostra que o mais importante em um dissipador é a sua capacidade total de dissipar calor, sendo pouco relevante o posicionamento das aletas quanto aos *MOSFET's*.

5.3 Caso 3 - Aletas Curtas

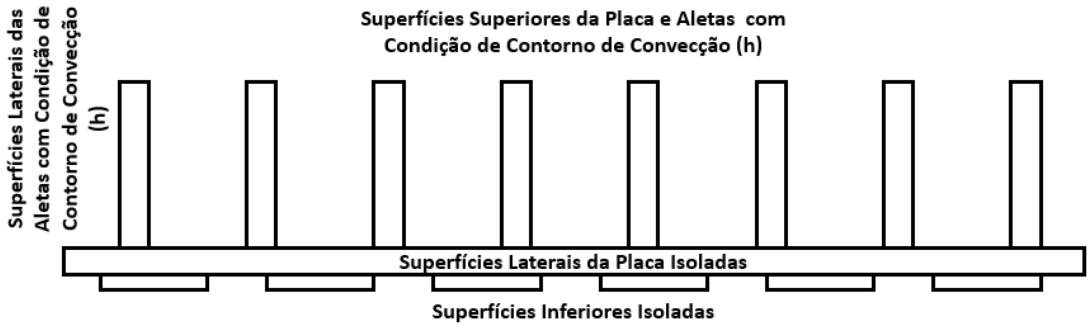


Figura 5.8: Condição de contorno para o caso 3. Desenho fora de escala.

Fonte: Elaborado pelo Autor

Parâmetros para a simulação do caso 3:

- 6 Fases no *design* do *VRM*, com *MOSFET's* IR3575;

- Comprimento do dissipador de 60 mm;
- Largura do dissipador de 18 mm;
- Altura das aletas de 9 mm;
- número de aletas de 8.

Abaixo, na Tabela 5.4, encontram-se alguns dados relevantes da simulação do caso 3:

Caso 3	
Área de Convecção [cm^2]	38,88
Eficiência de *MOSFET* em *PL1*	94,3%
Geração de Calor Interna *PL1* [W]	5,70
Temperatura Máxima Permanente [°C]	100,33
Eficiência de *MOSFET* em *PL2*	94,4%
Geração de Calor Interna *PL2* [W]	8,40
Temperatura Máxima Transiente [°C]	124,23
Número de Elementos	493.544

Tabela 5.4: Dados referentes a simulação.

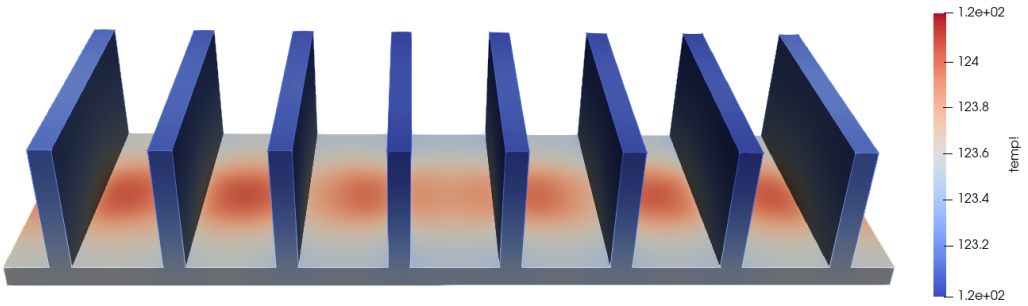


Figura 5.9: Visualização da simulação no caso 3 no *Paraview*, vista superior.

Fonte: Elaborado pelo Autor

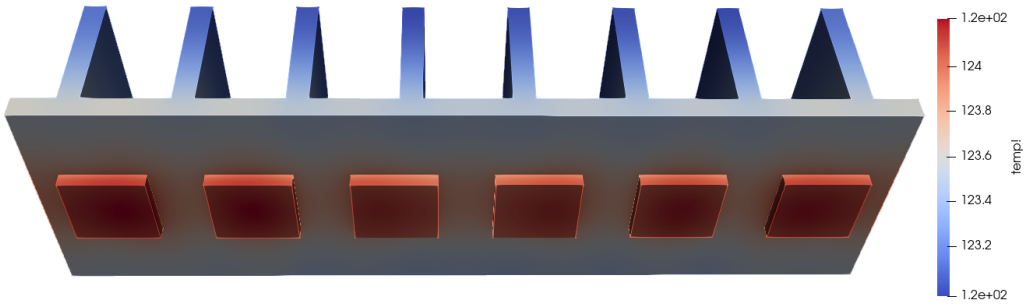


Figura 5.10: Visualização da simulação no caso 3 no *Paraview*, vista inferior.

Fonte: Elaborado pelo Autor

A simulação consistiu em uma redução no comprimento das aletas quando comparado ao caso de referência. O efeito obtido dessa mudança pelo código desenvolvido é um grande aumento na temperatura. As maiores temperaturas encontradas na simulação, $124,21°C$, estão localizadas nos 2 *MOSFET's* da esquerda na imagem 5.10. Isso excede com folga o limite operacional de $95°C$ especificado pelo fabricante. O resultado é esperado uma vez que há uma grande redução na área de convecção quando comparado ao caso de referência. Como a temperatura operacional foi consideravelmente excedida, isso sugere que frequentemente a placa mãe terá de acionar seus mecanismos de segurança, o que levará a frequentes desligamentos do computador ou a desempenho significativamente comprometido do mesmo.

5.4 Caso 4 - Redução do Número de Aletas

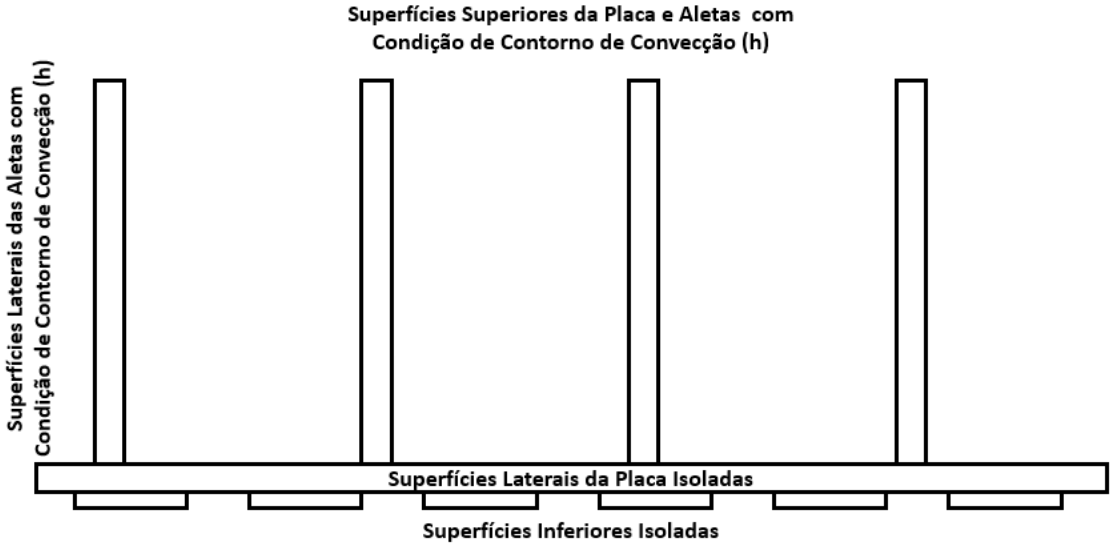


Figura 5.11: Condição de contorno para o caso 4. Desenho fora de escala.

Fonte: Elaborado pelo Autor

Parâmetros para a simulação do caso 4:

- 6 Fases no *design* do *VRM*, com *MOSFET's* IR3575;
- Comprimento do dissipador de 60 mm;
- Largura do dissipador de 18 mm;
- Altura das aletas de 19 mm;
- número de aletas de 4.

Abaixo, na Tabela 5.5, encontram-se alguns dados relevantes da simulação do caso 4:

Caso 4	
Área de Convecção [cm^2]	40,44
Eficiência de *MOSFET* em *PL1*	94,3%
Geração de Calor Interna *PL1* [W]	5,70
Temperatura Máxima Permanente [°C]	83,28

Eficiência de *MOSFET* em *PL2*	94,4%
Geração de Calor Interna *PL2* [W]	8,40
Temperatura Máxima Transiente [°C]	107,48
Número de Elementos	439.176

Tabela 5.5: Dados referentes a simulação.

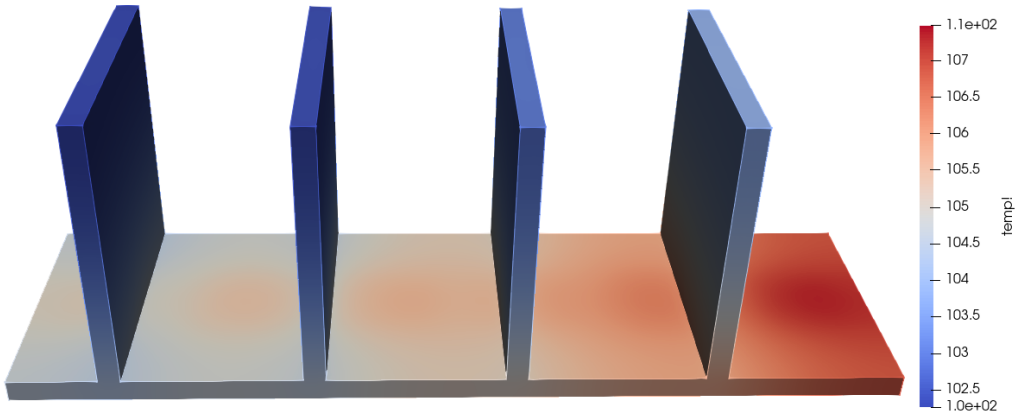


Figura 5.12: Visualização da simulação no caso 4 no *Paraview*, vista superior.

Fonte: Elaborado pelo Autor

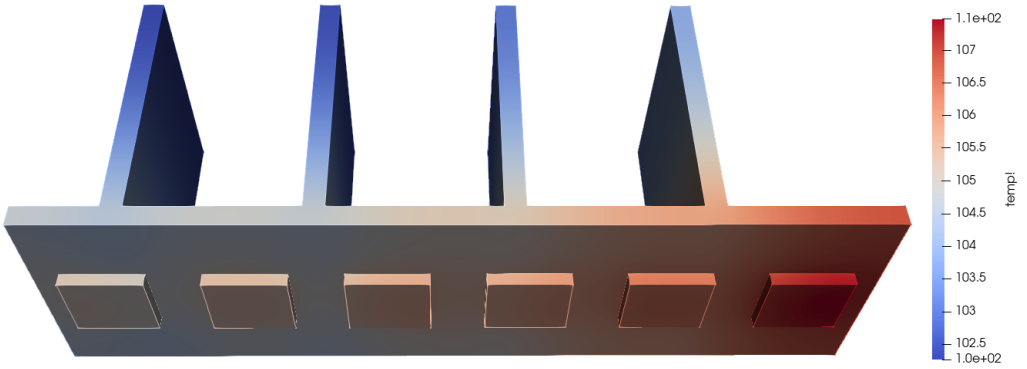


Figura 5.13: Visualização da simulação no caso 4 no *Paraview*, vista inferior.

Fonte: Elaborado pelo Autor

A simulação consistiu em uma diminuição pela metade no número de aletas quando comparado a referência, indo de 8 para 4. O efeito obtido dessa mudança pelo código desenvolvido é um aumento na temperatura, mas menos expressivo

que no caso anterior. As maiores temperaturas encontradas nessa simulação estão localizadas no *MOSFET* mais à direita na imagem 5.13, sendo o máximo encontrado de $107,47°C$. De maneira similar a simulação 3, o mecanismo de segurança da placa mãe seria utilizado para que a temperatura limite do IR3575 não seja atingida. Mais uma vez o posicionamento das aletas tem pouco impacto na temperatura máxima, havendo somente um delta de cerca de $2,5°C$ entre o *MOSFET* com a maior e menor temperatura. Isso ocorre apesar do posicionamento desvantajoso das aletas para o componente de maior temperatura.

Na simulação do caso 4, houve uma redução drástica de área de convecção, assim como no caso 3. No entanto, a diferença de temperatura entre às duas simulações foi substancial. Isso se deve a um aumento significativo do coeficiente de convecção em relação ao caso anterior que reduziu o impacto da perda de superfície exposta ao ar na temperatura máxima encontrada.

5.5 Caso 5 - Aumento das Dimensões do Dissipador

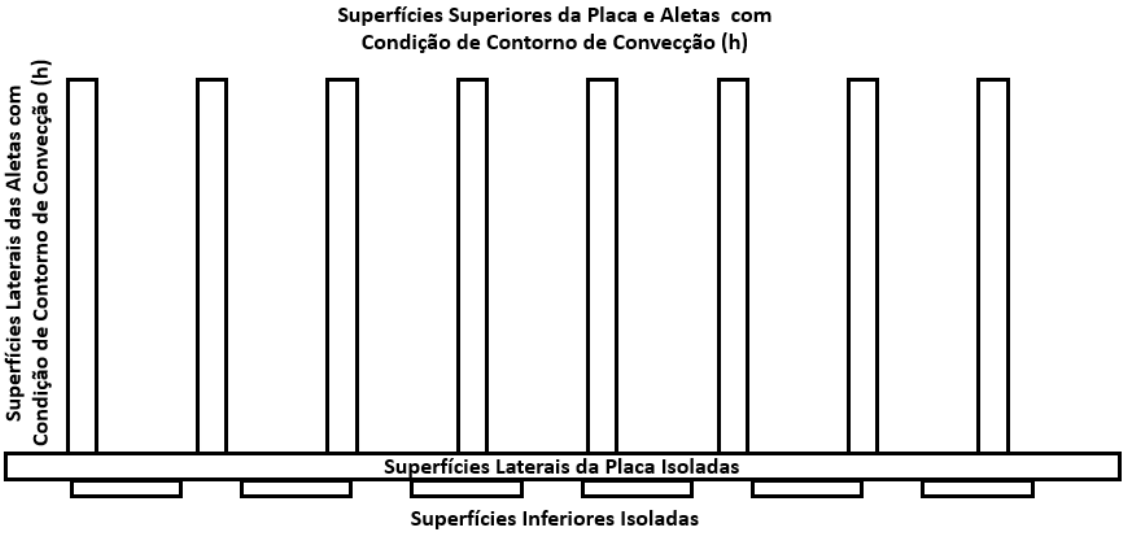


Figura 5.14: Condição de contorno para o caso 5. Desenho fora de escala.

Fonte: Elaborado pelo Autor

Parâmetros para a simulação do caso 5:

- 6 Fases no *design* do *VRM*, com *MOSFET's* IR3575;

- Comprimento do dissipador de 64 mm;
- Largura do dissipador de 24 mm;
- Altura das aletas de 19 mm;
- número de aletas de 8.

Abaixo, na Tabela 5.6, encontram-se alguns dados relevantes da simulação do caso 5:

Caso 5	
Área de Convecção [cm^2]	92,88
Eficiência de *MOSFET* em *PL1*	94,3%
Geração de Calor Interna *PL1* [W]	5,70
Temperatura Máxima Permanente [°C]	71,01
Eficiência de *MOSFET* em *PL2*	94,4%
Geração de Calor Interna *PL2* [W]	8,40
Temperatura Máxima Transiente [°C]	83,3
Número de Elementos	810.360

Tabela 5.6: Dados referentes a simulação.

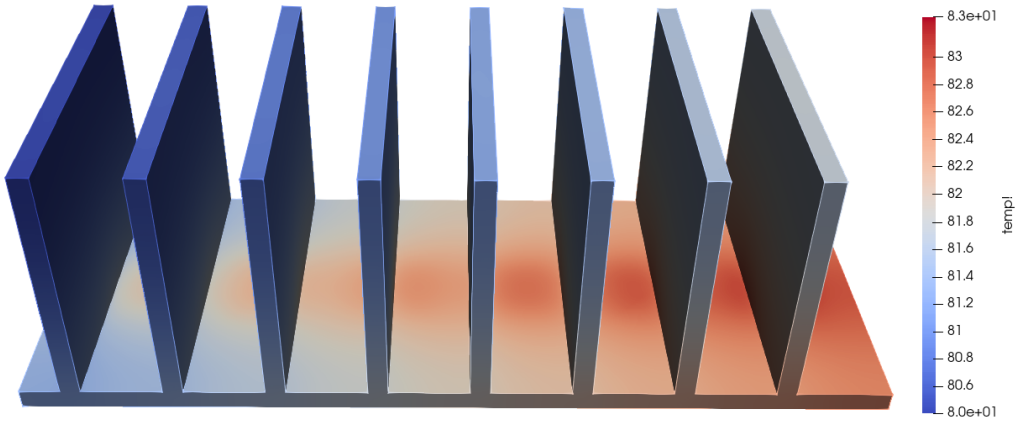


Figura 5.15: Visualização da simulação no caso 5 no *Paraview*, vista superior.

Fonte: Elaborado pelo Autor

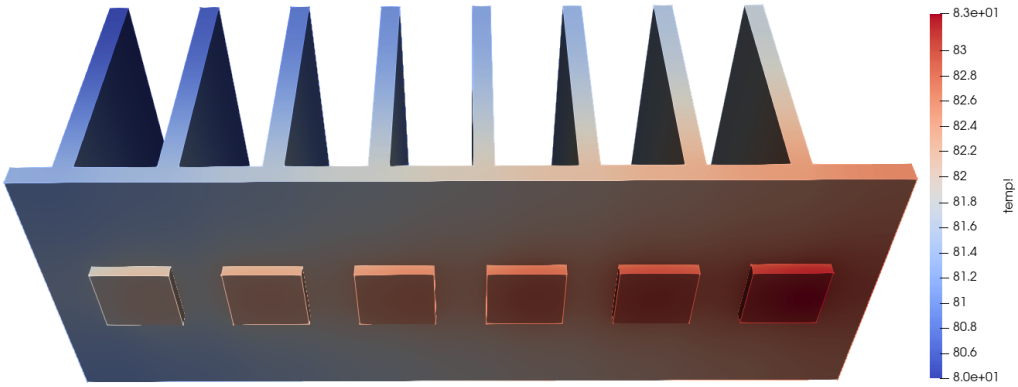


Figura 5.16: Visualização da simulação no caso 5 no *Paraview*, vista inferior.

Fonte: Elaborado pelo Autor

A simulação do caso 5 consistiu em um aumento das dimensões do dissipador quando comparado a referência. Isso leva a um aumento considerável na área de convecção, reduzindo a maior temperatura encontrada nos *MOSFET's* para $83,29°C$. Essa temperatura foi encontrada no *MOSFET* mais à direita na imagem 5.16, sendo bem inferior à temperatura limite operacional do componente de $95°C$. Diferentemente das demais configurações de Módulos Reguladores de Tensão testados, este não levaria a desligamentos repentinos do computador ou a desempenho reduzido do processador.

Capítulo 6

Conclusão

O código desenvolvido em *Python* utiliza o Método de Elementos Finitos para resolver a equação de calor tridimensional permanente e transiente para o caso específico de *MOSFET's* de um *VRM* sendo arrefecidos por um dissipador de calor em convecção natural. Essa é uma situação com aplicação real e alta complexidade, envolvendo condução em diferentes materiais, geração de calor e convecção. Isso mostra que os métodos numéricos, e em especial o MEF, são uma válida alternativa na solução dos mais variados problemas térmicos.

A ferramenta foi testada nos vários aspectos relevantes envolvidos nos problemas em questão através de comparações com o *software* comercial *Ansys* e também com uma solução analítica. O código proposto apresentou erros na casa dos centésimos de grau em todos os testes, com exceção no caso da condução de calor com 2 materiais, em virtude do tratamento diferenciado do código as regiões de interface dos materiais. Mas, mesmo nesse último, o erro apresentado foi tolerável, ficando abaixo de 1 grau Celsius no pior dos casos.

As 5 simulações realizadas deixam claro que somente a última seria capaz de atender adequadamente as demandas do processador. Isso significa que, para o caso de um módulo regulador de tensão de 6 fases, há a necessidade de um dissipador de calor de grandes proporções, similar ao utilizado na simulação 5. Outro ponto importante é de que não somente a área de convecção de um dissipador é relevante para o arrefecimento. Outro fator de grande impacto é a relação entre a geometria do dissipador e o coeficiente de convecção médio. Essa relação ficou clara ao se comparar a simulação 3 e 4, onde se pode concluir que a geometria da 4 é mais

eficiente, ao ter um h_{med} mais elevado e área de convecção similar.

O programa foi desenvolvido de maneira a requerer mínima interação com o usuário, sendo possível controlar muitos dos aspectos relevantes da simulação através de sua interface gráfica. Desse modo, o usuário pode modificar a geometria do dissipador, número de fases, temperatura ambiente, além do *PL1* e *PL2* do processador somente alterando as entradas do *software*. Isso proporciona maior velocidade ao usuário para o dimensionamento de um *VRM* e dissipador para suportar um determinado processador. Vale ressaltar que o código desenvolvido para este projeto, com todos os seus módulos, está inteiramente contido no Apêndice.

6.1 Sugestões para futuros desenvolvimentos

Para futuros trabalhos e desenvolvimentos os seguintes pontos podem ser implementados ou melhorados:

- Implementar a simulação de dissipadores de calor horizontalmente posicionados.
- Realizar uma análise de convergência para a variação dos elementos da malha.
- Incluir na simulação a pasta térmica entre os componentes e o dissipador.
- Modelar o *PCB* ao qual os *MOSFET's* estão localizados e inclui-los na simulação térmica.
- Tornar o código mais abrangente, permitindo também o dimensionamento de dissipadores passivos para processadores através de entradas do usuário na interface gráfica do programa.

Referências Bibliográficas

[1] NITHIARASU, K. N. S. . R. W. L. . P., *Fundamentals of the Finite Element Method for Heat and Mass Transfer*. John Wiley Sons Ltd: Chichester, West Sussex, PO19 8SQ, United Kingdom.

[2] CHEN, W.-K., *The Electrical Engineering Handbook*. Elsevier Academic Press: Burlington, MA, USA.

[3] *60A Exposed Top Integrated PowIRstage IR3575*, Tech. rep., International Rectifier, El Segundo, California, 2017.

[4] PARISI, C., *Multiphase Buck Design From Start to Finish (Part 1)*, Tech. Rep. SLVA882B, Texas Instruments, Dallas, Texas, 2017.

[5] SILVA, V., "Experimental Analysis of the Influence of Heat Sink Geometric Parameters on Natural Convection", *Engenharia Térmica*, v. 15, pp. 26–32, 2016.

[6] *12th Generation Intel® Core™ Processors*, Tech. Rep. No.: 655258, Rev.: 010, Intel Corporation, Santa Clara, California, 2023.

[7] INCROPERA, D. . B. . L. ., *Fundamentos de Transferência de Calor e de Massa*. LTC: Rio de Janeiro, RJ, 2008.

[8] OZISIK, M. N., *Heat Transfer, a Basic Approach.*. McGraw-Hill Book Company, 1985.

[9] LOGAN, D. L., *A First Course in the Finite Element Method*. Thomson, 2007.

[10] ANJOS, G., "Computação Científica para Engenheiros", 2022.

Apêndice A

Código Fonte

O código é dividido em 5 módulos, todos presentes na mesma pasta:

- Módulo Executar
- Módulo Central
- Módulo Malha
- Módulo Matrix
- Módulo Auxiliar

O código de cada módulo em conjunto com uma pequena descrição de sua função estão presentes a seguir.

A.1 Módulo Executar

O código do Módulo Executar tem como função gerar uma interface gráfica para facilitar a inserção das entradas necessárias para o funcionamento do programa. Após a captação dos dados e o usuário pressionar o botão "rodar", é chamado a função principal do Módulo Central passando os *Inputs* do usuário. A interface gráfica é gerada por meio da biblioteca *tkinter*.

```
import Modulo_Central as principal
from tkinter import *
class MyWindow:
    def __init__(self, win):
        #Legenda para Inputs da Primeira Coluna
        self.lbl1=Label(win, text='Material do Dissipador')
        self.lbl2=Label(win, text='Temperatura Ambiente [C]')
        self.lbl3=Label(win, text='Numero de MOSFETs')
        self.lbl4=Label(win, text='Versao do VRM')
        #Legenda para Inputs da Segunda Coluna
        self.lbl20=Label(win, text='Grossura da Placa [CM]')
        self.lbl21=Label(win, text='Altura das Aletas [CM]')
        self.lbl22=Label(win, text='Distancia Entre Aletas [CM]')
        self.lbl23=Label(win, text='Grossura das Aletas [CM]')
        #Legenda para Inputs da Terceira Coluna
        self.lbl6=Label(win, text='PL1 do CPU [W]')
        self.lbl7=Label(win, text='PL2 do CPU [W]')
        self.lbl8=Label(win, text='Tempo em PL2 [s]')
        self.lbl9=Label(win, text='Refino da Malha')
        self.lbl10=Label(win, text='Comp Dissipador [CM]')
        self.lbl11=Label(win, text='Largura Dissipador [CM]')
        self.lbl99=Label(win, text='Status')
        #Inputs
        self.t1=Entry(bd=1)
        self.t2=Entry()
        self.t3=Entry()
        self.t4=Entry()
```

```
self.t6=Entry()
self.t7=Entry()
self.t8=Entry()
self.t9=Entry()
self.t10=Entry()
self.t11=Entry()
self.t20=Entry()
self.t21=Entry()
self.t22=Entry()
self.t23=Entry()
self.t99=Entry()
#Botao
self.btn1 = Button(win, text='Add')
#Posicionando Primeira Coluna
self.lbl1.place(x=100, y=50)
self.t1.place(x=260, y=50)
self.lbl2.place(x=100, y=100)
self.t2.place(x=260, y=100)
self.lbl3.place(x=100, y=150)
self.t3.place(x=260, y=150)
self.lbl4.place(x=100, y=200)
self.t4.place(x=260, y=200)
#Posicionando Segunda Coluna
self.lbl20.place(x=450, y=50)
self.t20.place(x=610, y=50)
self.lbl21.place(x=450, y=100)
self.t21.place(x=610, y=100)
self.lbl22.place(x=450, y=150)
self.t22.place(x=610, y=150)
self.lbl23.place(x=450, y=200)
self.t23.place(x=610, y=200)
#Posicionando Terceira Coluna
self.lbl6.place(x=800, y=50)
self.t6.place(x=960, y=50)
```

```
        self.lbl7.place(x=800, y=100)
        self.t7.place(x=960, y=100)
        self.lbl8.place(x=800, y=150)
        self.t8.place(x=960, y=150)
        self.lbl9.place(x=800, y=200)
        self.t9.place(x=960, y=200)
        self.lbl10.place(x=800, y=250)
        self.t10.place(x=960, y=250)
        self.lbl11.place(x=800, y=300)
        self.t11.place(x=960, y=300)
        ########################################################
        self.b1=Button(win, text=' Rodar ', command=self.add)
        self.b1.place(x=600, y=350)
        self.lbl99.place(x=600, y=400)
        self.t99.place(x=760, y=400)
    def add(self):
        self.t99.delete(0, 'end')
        #Inserindo as variaveis dos Inputs do usuario como Inputs da
            Funcao do Modulo Central
        mat=float(self.t1.get())
        if mat == 0: #material do dissipador sendo aluminio
            k=2.092
            rho=2.7
            cv=0.92048
        if mat == 1: #material do dissipador sendo cobre
            k=4.010
            rho=8.933
            cv=0.385
        T_inf=float(self.t2.get())
        vrm_n=int(self.t3.get())
        ver=float(self.t4.get())
        pl1=float(self.t6.get())
        pl2=float(self.t7.get())
        tpl2=float(self.t8.get())
```

```
        refino=int(self.t9.get())
        plc_x=float(self.t10.get())
        plc_y=float(self.t11.get())
        plc_z=float(self.t20.get())
        fin_z=float(self.t21.get())
        fin_d=float(self.t22.get())
        fin_x=float(self.t23.get())
        result=principal.main_func(k,rho,cv,0.09,1,
        refino,10,plc_z,fin_z,fin_d,ver,T_inf,fin_x,1.0,
        plc_x,plc_y,vrm_n,pl1,pl2,tpl2)
        self.t99.insert(END, str(result))



window=Tk()
mywin=MyWindow(window)
window.title('Calculo de Temperatura do MOSFET VRM')
window.geometry("1200x500+10+10")
window.mainloop()
```

A.2 Módulo Central

O código do Módulo Central interage com todos os outros Módulos e é nele que o sistema é montado e resolvido. O código é composto de uma única função, cujas entradas correspondem a todos os parâmetros do programa que podem ser modificados. Nem todas as variáveis dessa função estão expostas ao usuário no Módulo Executar.

```
## ===================================================================== ##
#  this is file Modulo_Central, created at 26-jul-2023                  #
#  maintained by Gabriel Affonso Costa Waehneldt                         #
#  e-mail: gabrielaffonsocosta@poli.ufrj.br                              #
## ===================================================================== ##

#########################################################################
#importacao de modulos e bibliotecas

import numpy as np
import sys
import matplotlib.pyplot as plt
import matplotlib
import Modulo_Malha as mesh
import Modulo_Matrix as matrix
import Modulo_Auxiliar as misc
import meshio
import scipy as sp
from scipy.sparse import csr_matrix
from scipy.sparse import lil_matrix
from scipy.sparse.linalg import spsolve
np.set_printoptions(threshold=sys.maxsize)

#########################################################################
#Funcao Principal do Codigo
def main_func(k,rho,cv,vrm_z,dt,refino,it,plc_z,fin_z,fin_d,ver,T_inf,
fin_x,vrm_d,plc_x,plc_y,vrm_n,pl1,pl2,tpl2):
```

```
#########################################################################
#Comando de iteracao - Ciclo 1 a 4, rodando o caso permanente 4 vezes
    para maior precisao no h_med. Ciclo 5, caso transiente.
cycle = 1
for i in range(0,5):
    if cycle==1: #No primeiro ciclo considera como Ts=75 graus celcius
        Ts=60
        teta = 1
    if cycle>=2: #Nos outros ciclos considera a temperatura real de um
        ponto na condicao de contorno de conveccao
        Ts=T[IENbound[1][0]]
    if cycle==1: #Calcula 4 vezes o caso permanente para maior
        precisao no Ts e consequentemente h_med
        mode = 0
    if cycle==5: #Ciclo 5 e o transiente, com maior carga no
        processador
        mode = 1
        it = 15
        dt = tpl2/it
        T = T_per



    q = misc.vrm_w(vrm_n,1.2,pl1,pl2,cycle)
    #########################################################################
    # Construcao dos Vetores X, Y, Z e IEN
    if cycle == 1: #so precisa montar as matrizes no primeiro ciclo
        ms = mesh.mesh_generation(vrm_z,vrm_d,plc_z,fin_z,
        fin_d,refino,fin_x,plc_x,plc_y,vrm_n)
        msh = meshio.read("dissipador.msh")
        IEN = ms[0]
        IENbound = ms[1]
        IENboundA1 = ms[2]
        IENboundA2 = ms[3]
```

```
    X = ms[4];Y = ms[5];Z = ms[6]
    VRM_pos = ms[8] #lista das coordenadas dos MOSFET'S do VRM
    IENbound_conv1 = ms[9] #IENBOUND dos elementos de contorno que
        sofrem conveccao
    npoints = len(X) #Numero de pontos
    ne = len(IEN) #Numero de elementos

T_inf = (np.ones((npoints),dtype="double")*T_inf) #transformando
    T_inf em vetor

########################################################################
#Transformando as propriedades fisicas em vetores
if cycle == 1:
    cv_ini = cv
    rho_ini = rho
    k_ini = k
    cv = (np.ones((npoints),dtype="double")*cv)
    rho = (np.ones((npoints),dtype="double")*rho)
    k = (np.ones((npoints),dtype="double")*k)

########################################################################
#Chamando funcao para calculo de coeficiente de conveccao
H = fin_z/100 # Altura das aletas em metros
n = ms[12] # Numero de aletas
L = ms[11]/100 # Largura da placa em metros
W = ms[10]/100 # Comprimento da placa em metros
S = fin_d/100 # Distancia entre aletas em metros
t = fin_x/100 # Grossura da aleta
h_med = misc.h_calc(H,n,W,L,S,Ts,T_inf,t) #calculando h medio
if h_med==1: #Gerar erro caso o Numero de Rayleigh no esteja na
    faixa aceitavel para o calculo de h_med
    raise Exception("Numero de Rayleigh fora da faixa valida.
        Tente modificar a geometria do dissipador de calor.")
```

```
#########################################################################
# inicializacao das matrizes e vetores

m_data =
    matrix.ger_matrix(X,Y,Z,IEN,ne,k,npoints,IENbound_conv1,h_med,cv,rho)
    #uso do modulo para a geracao das matrizes globais
K = m_data[0] ; M = m_data[1] ; M_cc = m_data[2]

#########################################################################
#Montando o Sistema Linear

if mode == 0: #caso transiente
    it=1
    dt=1
b = np.zeros( (npoints),dtype='double' )
if cycle==1 or cycle==2 or cycle==3 or cycle==4:
    T = (np.ones( (npoints),dtype='double' ))*T_inf


A = (((M)/dt)*mode) + (teta*(K+M_cc))


#########################################################################
#Montando o Vetor de geracao de calor

if cycle == 1:
    Q = np.zeros( (npoints),dtype='double' )
    #Definindo geracao de calor para os pontos pertinentes
    for n in range(len(VRM_pos)):
        for e in IEN:
            for i in e:
                if Z[i]<=VRM_pos[n][5] and Z[i]>=VRM_pos[n][4] and
```

```
                    X[i]>=VRM_pos[n][0] and X[i]<=VRM_pos[n][1] and
                    Y[i]>=VRM_pos[n][2] and Y[i]<=VRM_pos[n][3]:
                    cv[i] = 0.7 #calor especifico do vrm
                    rho[i] = 2.329 #massa especifica do vrm
                    Q[i] = (q/(cv[i]*rho[i])) #geracao de calor no
                        vrm
                    k[i] = 1.19
                    if Z[i] == 0.09:#fazendo a media na interface
                        dos materiais
                        cv[i] = (0.7 + cv_ini)/2 #media entre
                            materiais
                        rho[i] = (2.329 + rho_ini)/2 #media entre
                            materiais
                        Q[i] = (q/(cv[i]*rho[i]*2)) #media entre
                            materiais
                        k[i] = (1.19+k_ini)/2 #media entre materiais


    MQ = M@Q
if cycle == 5:
    Q = np.zeros( (npoints),dtype='double' )
    #Definindo geracao de calor para os pontos pertinentes
    for n in range(len(VRM_pos)):
        for e in IEN:
            for i in e:
                if Z[i]<=VRM_pos[n][5] and Z[i]>=VRM_pos[n][4] and
                    X[i]>=VRM_pos[n][0] and X[i]<=VRM_pos[n][1] and
                    Y[i]>=VRM_pos[n][2] and Y[i]<=VRM_pos[n][3]:
                    cv[i] = 0.7 #calor especifico do vrm
                    rho[i] = 2.329 #massa especifica do vrm
                    Q[i] = (q/(cv[i]*rho[i])) #geracao de calor no
                        vrm
                    k[i] = 1.19
                    if Z[i] == 0.09: #fazendo a media na interface
```

```
                    dos materiais
                    cv[i] = (0.7 + cv_ini)/2 #media entre
                        materiais
                    rho[i] = (2.329 + rho_ini)/2 #media entre
                        materiais
                    Q[i] = (q/(cv[i]*rho[i]*2))#media entre
                        materiais
                    k[i] = (1.19+k_ini)/2 #media entre materiais

    MQ = M@Q
#####################################################################
#Imposicao das c.c.s de Dirichlet
#A=A.tolil()

#for e in IENboundA2:
#   for i in e:
#      A[i,:] = 0.0
#      A[i,i] = 1.0
#      b[i] = 10

#for e in IENboundA1:
#   for i in e:
#      A[i,:] = 0.0
#      A[i,i] = 1.0
#      b[i] = 100
#A=A.tocsr()
#A_sparse = csr_matrix(A)

#####################################################################
#Solucionando o Sistema Linear

for i in range(it):
    print ("Ciclo de solucao de Matrix:"+str(i+1))
```

```
    b = ((M)@(T/dt))*mode + MQ - ((K+M_cc)@((1-teta)*T)) +
        (M_cc@T_inf)
    #imposicao das c.c.s de Dirichlet

    #for e in IENboundA2:
    #    for i in e:
    #        b[i] = 10
    #for e in IENboundA1:
    #    for i in e:
    #        b[i] = 100

    T = spsolve(A,b) #resolvendo matriz esparsa
    if cycle==5:
        Ts=T[IENbound[1][0]]
        h_med_antigo=h_med #salvando o valor antigo de h_med
        h_med = misc.h_calc(H,n,W,L,S,Ts,T_inf,t) #recalculando
            h_med para cada nova temperatura no estado transiente
        M_cc=M_cc*(h_med/h_med_antigo) #modificando M_cc com o
            novo h_med
        A = (((M)/dt)*mode) + (teta*(K+M_cc)) #Recriando A para o
            proximo ciclo com o M_cc atualizado
    if cycle==4:
        T_per = T #Definido a temperatura do estado permanente
            como ponto de partida do estado transiente
############################################################
#Mostrar resultados
T_max_cord = np.argmax(T) #encontrar indice da temperatura mais
    elevada
T_max = T[T_max_cord] #Temperatura mais elevada
print ("Temperatura maxima no VRM: "+str(T[T_max_cord]))

point_data = {"temp!":T}
meshio.write_points_cells(f"Teste_conducao-"+str(ver)+".vtk",
msh.points,msh.cells,point_data=point_data,)
```

```
    if cycle==4:
        return ("-################################################
        ################################-")
    cycle +=1
```

A.3 Módulo Malha

O módulo tem como função receber do Módulo Central os dados geométricos e de número de *MOSFET's* e com isso gerar a malha do problema. Ele também lê os dados relevantes como posição dos pontos, IEN e IENbound e retorna-os para o Módulo Central. A geração de malha ocorre por meio do Gmsh, mas essa interação é feita inteiramente pelo código do módulo, sem necessidade de iteração do usuário.

```
# Import modules:
import gmsh
import sys
import meshio as ms

#############################################################################
#Funcao que Recebe Dados Geometricos e Retorna os Dados Relevantes da
    Malha

def
    get_mesh(vrm_z,vrm_d,plc_z,fin_z,fin_d,VRM_pos,Placa_C,Placa_L,n_fins):
    msh = ms.read("dissipador.msh") #Lendo a malha gerada no
        mesh_generation
    points = msh.points
    data = msh.cells_dict
    IENbound = data["triangle"]
    IEN = data["tetra"]
    X = [];Y = []; Z = []
    #passando por todos os pontos e guardando as coordenadas X, Y e Z
        separadamente
    for e in range(len(points)):
        X.append(points[e][0]);Y.append(points[e][1]);Z.append(points[e][2])

    #############################################################################
    #Gerando as IENbound especificas
    IENboundA1 = []; IENboundA2 = [];IENbound_conv1 = []
```

```
    for e in IENbound:
        aux1 = []
        aux2 = []
        aux3 = []
        #Definindo quais as condicoes para cada IENbound
        for vertex in e:
            if Z[vertex]<=(0.00):
                aux1.append(vertex)
            if Z[vertex]>=(vrm_z+plc_z+fin_z*0.9999999999):
                aux2.append(vertex)
            if Z[vertex]>=(vrm_z+plc_z): #Unico contorno utilizado no
               programa final, conveccao
                aux3.append(vertex)
        #Adicionando os elementos de contorno nas IENbound corretas
        if len(aux1)==3:
            IENboundA1.append(e)
        if len(aux2)==3:
            IENboundA2.append(e)
        if len(aux3)==3:
            IENbound_conv1.append(e)

    return (IEN,IENbound,IENboundA1,IENboundA2,X,Y,Z,msh,
    VRM_pos,IENbound_conv1,Placa_C,Placa_L,n_fins)

################################################################################
#Funcao que recebe dados geometricos e gera uma malha

def
    mesh_generation(vrm_z,vrm_d,plc_z,fin_z,fin_d,refino,fin_x,plc_x,plc_y,vrm_n):
    vrm_x = 0.6
    vrm_y = 0.6# Tamanhos dos MOSFET's do VRM

    # Initialize gmsh:
    gmsh.initialize()
```

```
#gerando a geometria de cada MOSFET do VRM
VRM=[]
VRM_pos=[] #posicao dos MOSFET's
for i in range(0,vrm_n):
    VRM.append(gmsh.model.occ.addBox( vrm_d*i , 0 , 0 ,
        vrm_x,vrm_y,vrm_z,i+1))
    VRM_pos.append([vrm_d*i,vrm_d*i+vrm_x,0,vrm_y,0,vrm_z])
Placa_C = ((vrm_d)*(vrm_n-1))+(plc_x*2)+vrm_x #Comprimento da placa
Placa_L = vrm_y+(plc_y*2) #Largura da placa
Plate = gmsh.model.occ.addBox( -plc_x , -plc_y , vrm_z ,
    Placa_C,Placa_L,plc_z,vrm_n+1) #addBox(x,y,z,dx,dy,dz,tag)
Fins=[]
n_fins = round((Placa_C+fin_d*0.5)/(fin_d+fin_x)-0.5)
for i in range(0,n_fins):
 Fins.append(gmsh.model.occ.addBox((fin_d+fin_x)*i-plc_x+(fin_d/2),-plc_y,
 vrm_z+plc_z, fin_x,Placa_L,fin_z,i+vrm_n+2))
     #addBox(x,y,z,dx,dy,dz,tag)
Fuse_model = gmsh.model.occ.fuse([(3, Plate)], [(3, VRM[0])])
model_num = Fuse_model[0][0][1] #pegando o indice do modelo da fusao
for i in range(1,vrm_n): #fusao entre modelos de placa e MOSFET's
 gmsh.model.occ.fuse([(3,model_num )], [(3, VRM[i])])
for i in range(0,len(Fins)):
 gmsh.model.occ.fuse([(3,model_num )], [(3, Fins[i])])



gmsh.model.occ.synchronize()
volumes = gmsh.model.getEntities(dim=3)

# Gerando a malha
gmsh.model.mesh.generate(3) #Malha 3D

# Refinando Malha
```

```
if refino>=1:
    for i in range(1,refino+1):
        gmsh.model.mesh.refine()

# Escrevendo os dados da malha
gmsh.write("dissipador.msh")

# Colocando o gmsh para abrir a malha gerada
if 'close' not in sys.argv:
    gmsh.fltk.run()

# Finalizando o gmsh

gmsh.finalize()

# Pegando os dados da malha

mesh =
    get_mesh(vrm_z,vrm_d,plc_z,fin_z,fin_d,VRM_pos,Placa_C,Placa_L,n_fins)
    #Recebendo os dados da funcao get_mesh
return (mesh)
```

A.4 Módulo Matrix

O módulo recebe os dados da malha e dados físicos em cada ponto e retorna as matrizes globais ao Módulo Central. Uma vez que a maior parte de seus elementos das matrizes globais são zero, as mesmas foram criadas como matrizes esparsas.

```
import numpy as np
import scipy as sp
from scipy.sparse import csr_matrix
from scipy.sparse import csc_matrix
from scipy.sparse import lil_matrix
import sys

def ger_matrix(X,Y,Z,IEN,ne,k,npoints,IENbound_conv1,h_med,cv,rho):

    h_med = h_med/(10000) #transformando de metro quadrado em centimetro
        quadrado
    ##############################################################################
    # Inicializacao das matrizes e vetores tetraedricos
    K = lil_matrix((npoints, npoints),dtype = "double")
    M = lil_matrix((npoints, npoints),dtype = "double")

    ##############################################################################
    # Definicao das matrizes elementares tetraedricas
    for e in range(0,ne):

     [v1,v2,v3,v4] = IEN[e] # Pegando o numero de cada ponto do vertice
         do tetraedro

     det_vol= np.linalg.det([[1,X[v1],Y[v1],Z[v1]], # Calculo
         determinante para volume do tetraedro
                             [1,X[v2],Y[v2],Z[v2]],
                             [1,X[v3],Y[v3],Z[v3]],
                             [1,X[v4],Y[v4],Z[v4]]])
```

```
V = abs((1.0/6.0)*det_vol) #Calculo do volume de cada tetraedro


# Calculando valores medios das propriedades fisicas para cada
   elemento da malha
cv_med = ((cv[IEN[e][0]]) + (cv[IEN[e][1]]) + (cv[IEN[e][2]]) +
    (cv[IEN[e][3]]))/4.0 #definindo o cv medio como a media entre os
    cv's dos 4 pontos
rho_med = ((rho[IEN[e][0]]) + (rho[IEN[e][1]]) + (rho[IEN[e][2]]) +
    (rho[IEN[e][3]]))/4.0 #definindo o rho medio como a media entre
    os rho's dos 4 pontos
melem = (V/(20*cv_med*rho_med))*np.array([ [2,1,1,1], #Matriz de
    massa tetraedrica
                                        [1,2,1,1],
                                        [1,1,2,1],
                                        [1,1,1,2] ])


bi = ((Y[v2]-Y[v4])*(Z[v3]-Z[v4]) - (Y[v3]-Y[v4])*(Z[v2]-Z[v4]))
bj = ((Y[v3]-Y[v4])*(Z[v1]-Z[v4]) - (Y[v1]-Y[v4])*(Z[v3]-Z[v4]))
bk = ((Y[v1]-Y[v4])*(Z[v2]-Z[v4]) - (Y[v2]-Y[v4])*(Z[v1]-Z[v4]))
bl = -(bi+bj+bk)

ci = ((X[v3]-X[v4])*(Z[v2]-Z[v4])-(X[v2]-X[v4])*(Z[v3]-Z[v4]))
cj = ((X[v1]-X[v4])*(Z[v3]-Z[v4])-(X[v3]-X[v4])*(Z[v1]-Z[v4]))
ck = ((X[v2]-X[v4])*(Z[v1]-Z[v4])-(X[v1]-X[v4])*(Z[v2]-Z[v4]))
cl = -(ci+cj+ck)

di = ((X[v2]-X[v4])*(Y[v3]-Y[v4]) - (X[v3]-X[v4])*(Y[v2]-Y[v4]))
dj = ((X[v3]-X[v4])*(Y[v1]-Y[v4]) - (X[v1]-X[v4])*(Y[v3]-Y[v4]))
dk = ((X[v1]-X[v4])*(Y[v2]-Y[v4]) - (X[v2]-X[v4])*(Y[v1]-Y[v4]))
dl = -(di+dj+dk)

B = (1/(6*V))*np.array([[bi,bj,bk,bl],
```

```
                        [ci,cj,ck,cl],
                        [di,dj,dk,dl]])

 B_T = np.transpose(B)

 k_med = ((k[IEN[e][0]]) + (k[IEN[e][1]]) + (k[IEN[e][2]]) +
     (k[IEN[e][3]]))/4.0 #definindo a condutividade media como a media
     entre as condutividades dos 4 pontos
 kelem = V*(k_med/(cv_med*rho_med))*B_T@B #Matriz de rigidez elementar

 ###############################################################################
#Montando as Matrizes Globais

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

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

###############################################################################
#Definicao da matriz de massa dos elementos triangulares de contorno
M_cc = lil_matrix((npoints, npoints),dtype = np.float64)
A_cc_conv=0
for e in range(0,len(IENbound_conv1)):
    [v1,v2,v3] = IENbound_conv1[e]
    cv_med = ((cv[IEN[e][0]]) + (cv[IEN[e][1]]) + (cv[IEN[e][2]]))/3.0
        #Definindo o cv medio como a media entre os cv's dos 3 pontos
    rho_med = ((rho[IEN[e][0]]) + (rho[IEN[e][1]]) +
        (rho[IEN[e][2]]))/3.0 #Definindo o rho medio como a media
        entre os rho's dos 3 pontos
    A_tri=abs((0.5)*np.linalg.det([[1,X[v1],Y[v1]],
                            [1,X[v2],Y[v2]],
```

```
                        [1,X[v3],Y[v3]]]))
    if A_tri==0:
        A_tri=abs((0.5)*np.linalg.det([[1,Z[v1],Y[v1]],
                        [1,Z[v2],Y[v2]],
                        [1,Z[v3],Y[v3]]]))
    if A_tri==0:
        A_tri=abs((0.5)*np.linalg.det([[1,Z[v1],X[v1]],
                        [1,Z[v2],X[v2]],
                        [1,Z[v3],X[v3]]]))

    A_cc_conv+=A_tri #rea da superficie de conveccao
    melem_tri = (A_tri/(12.0*cv_med*rho_med))*np.array([ [2,1,1],
                                                [1,2,1],
                                                [1,1,2] ])


##############################################################################
#Montando as matrizes dos elementos triangulares de contorno

    for ilocal in range(0,3):
      iglobal = IENbound_conv1[e][ilocal]
      for jlocal in range(0,3):
       jglobal = IENbound_conv1[e][jlocal]


       M_cc[iglobal,jglobal] += melem_tri[ilocal,jlocal]
M_cc = M_cc*h_med

M = M.tocsr()
K = K.tocsr()
M_cc = M_cc.tocsr()
print ("rea de conveccao: "+str((A_cc_conv)))#Printando area de
    conveccao
return [K,M,M_cc]
```

A.5 Módulo Auxiliar

O módulo abriga 2 funções que interagem com o Módulo Central em momentos diferentes. Há uma função para se calcular a geração de calor interno dos *MOSFET's* e outra para o valor do coeficiente de convecção médio. A função "interp"não interage com o Módulo principal e é usada somente para interpolação pelas funções internas ao Módulo.

```
import math
#############################################################################
#Funcao para imterpolar valores para as propriedades fisicas do ar
def interp(dados,valor):
    n = math.floor((valor-35)/5)
    if n>10: #Prevenindo que o programa crashe por falta de dados em
        temperaturas extremamente altas para o ar
        n=10
    val_1 = dados[n]
    val_2 = dados[n+1]
    aux1 = (valor - (5*n)-35)/5
    return ((val_1*(1-aux1))+(val_2*(aux1)))

#############################################################################
#Calculando o h_med em metros. Necessario converter para cm2
    posteriormente
def h_calc(H,n,W,L,S,Ts,T_inf,t):
    Ra_flag=1
    T_film = (Ts+T_inf[0])/2 #Temperatura de Filme, temperatura ambiente
        e um array
    #########################################################################
    #Propriedades Fisicas em 1 ATM em funcao da temperatura / Cengel e
        Gajar - Transferencia de Calor e Massa - Quarta Edicao - Tabela
        A-15 (de 35 a 90 graus)
    Pr_list = [0.7268,0.7255,0.7241,0.7228,0.7215,0.7202,
    0.7190,0.7177,0.7166,0.7154,0.7143,0.7132]
```

```
p_list = [1.145,1.127,1.109,1.092,1.076,1.028,1.014,0.9994,
0.9856,0.9718,0.9588,0.9458]
k_ar_list = [0.02625,0.02662,0.02699,0.02735,0.02772,0.02808,
0.02845,0.02881,0.02917,0.02953,0.02986,0.03024]
v_din_list = [1.895*(10**-5),1.918*(10**-5),1.941*(10**-5),
1.963*(10**-5),1.986*(10**-5),2.008*(10**-5),2.030*(10**-5),
2.052*(10**-5),2.074*(10**-5),2.096*(10**-5),2.119*(10**-5),2.139*(10**-5)]



#Caracteristicas Fisicas do ar a temperatura de filme

B = 2/(Ts+T_inf[0])
g = 9.81
Pr = interp(Pr_list,T_film) #Numero de Prandtl
v_din = interp(v_din_list,T_film) #viscosidade dinamica
p = interp(p_list,T_film) #massa especifica
v_cin = (v_din/p) #viscosidade cinematica
k_ar = interp(k_ar_list,T_film) #condutividade termica



Gr = (g*B*(Ts-T_inf[0])*(W**3))/(v_cin**2) #Numero de Grashof


Ra = Gr*Pr #Numero de Rayleigh
print ("Ra: "+str(Ra))
if Ra>=2.9*10**5 and Ra<=4.6*10**6:
    Ra_flag=0
if Ra_flag==1:
    print ("Erro - Modulo_Matrix - h_calc")
    return (1)

#Nu_med = 3.350*((Ra)**0.153)*((L/H)**0.121)*((S/H)**0.605) #Harahap
```

```
        and Lesmana.
    (2006) - Nusselt medio - aletas verticais, conveccao natural

    Nu_med = 0.042*((Ra)**0.229)*((S/L)**0.455)*((H/L)**-0.0112)
    *((t/L)**-1.082)*((n)**-0.119)

    h_med = (Nu_med*k_ar)/L #Coeficiente de transferecia de calor por
        conveccao medio
    print ("Valor de h_med: "+str(h_med))
    return (h_med)
#################################################################################
#Calculando q em W/cm3 dos MOSFET's do VRM em funcao da carga da CPU e
    numero de fases do VRM
def vrm_w(vrm_n,cpu_v,cpu_pl1,cpu_pl2,cycle):
    cpu_w = cpu_pl1 #estado normal do cpu
    if cycle>=5:
        cpu_w= cpu_pl2 #Estado temporario em pl2
    vrm_amp=cpu_w/(vrm_n*cpu_v)
    dados = [0,0.915,0.938,0.945,0.945,0.940,0.936,0.928,
    0.920,0.911,0.901,0.890,0.876]
    #Eficiencia do VRM/MOSFET IR3575
    n = math.floor((vrm_amp)/5)
    val_1 = dados[n]
    val_2 = dados[n+1]
    aux1 = (vrm_amp - (5*n))/5
    vrm_efic = ((val_1*(1-aux1))+(val_2*(aux1)))
    vrm_w = (1-vrm_efic)*cpu_v*vrm_amp
    q = vrm_w/(0.6*0.6*0.09)
    print ("Valor de q: "+str(q))
    return (q)
```

Apêndice B

Material para Cálculo das Matrizes Elementares Tetraédricas

Como foi apresentado no capítulo 2, a matriz elementar de rigidez é calculada da seguinte forma:

$$k^e = VkB^TB$$

Para esse cálculo, é necessário a definição da Matriz B que, segundo Nithiarasu [10], pode ser calculada como:

$$B = \frac{1}{6V}\begin{bmatrix} b_1 & b_2 & b_3 & b_4 \\ c_1 & c_2 & c_3 & c_4 \\ d_1 & d_2 & d_3 & d_4 \end{bmatrix}$$

Sendo V o volume do elemento tetraédrico. Os valores de b_i, c_i e d_i são calculados usando a posição dos nós dos elementos:

$$b_1 = -(y_2 - y_4)(z_3 - z_4) + (y_3 - y_4)(z_2 - z_4)$$

$$b_2 = -(y_3 - y_4)(z_1 - z_4) + (y_1 - y_4)(z_3 - z_4)$$

$$b_3 = -(y_1 - y_4)(z_2 - z_4) + (y_2 - y_4)(z_1 - z_4)$$

$$b_4 = -(b_1 + b_2 + b_3)$$

$$c_1 = -(x_3 - x_4)(z_2 - z_4) + (x_2 - x_4)(z_3 - z_4)$$

$$c_2 = -(x_1 - x_4)(z_3 - z_4) + (x_3 - x_4)(z_1 - z_4)$$

$$c_3 = -(x_2 - x_4)(z_1 - z_4) + (x_1 - x_4)(z_2 - z_4)$$

$$c_4 = -(c_1 + c_2 + c_3)$$

$$d_1 = -(x_2 - x_4)(y_3 - y_4) + (x_3 - x_4)(y_2 - y_4)$$

$$d_2 = -(x_3 - x_4)(y_1 - y_4) + (x_1 - x_4)(y_3 - y_4)$$

$$d_3 = -(x_1 - x_4)(y_2 - y_4) + (x_2 - x_4)(y_1 - y_4)$$

$$d_4 = -(d_1 + d_2 + d_3)$$

Por sua vez, o volume de um elemento tetraédrico pode ser encontrado por:

$$V = \frac{1}{6} \det \begin{bmatrix} 1 & x_1 & y_1 & z_1 \\ 1 & x_2 & y_2 & z_2 \\ 1 & x_3 & y_3 & z_3 \\ 1 & x_4 & y_4 & z_4 \end{bmatrix}$$