Universidade Federal de Ouro Preto
Instituto de Ciências Exatas e Aplicadas
Departamento de Engenharia Elétrica

Trabalho de Conclusão de Curso

Estratégias para redução dos tempos de execução de algoritmos SAFT para formação de imagens acústicas usando processadores quad-core

Mateus Henrique Gonçalves

João Monlevade, MG
2024

Mateus Henrique Gonçalves

Estratégias para redução dos tempos de execução de algoritmos SAFT para formação de imagens acústicas usando processadores quad-core

Trabalho de Conclusão de Curso apresentado à Universidade Federal de Ouro Preto como parte dos requisitos para obtenção do Título de Bacharel em Engenharia Elétrica pelo Instituto de Ciências Exatas e Aplicadas da Universidade Federal de Ouro Preto.
Orientador: Dr. Marcelo Moreira Tiago

Universidade Federal de Ouro Preto
João Monlevade
2024

SISBIN - SISTEMA DE BIBLIOTECAS E INFORMAÇÃO

G635e Gonçalves, Mateus Henrique.
Estratégias para redução dos tempos de execução de algoritmos SAFT para formação de imagens acústicas usando processadores quad-core. [manuscrito] / Mateus Henrique Gonçalves. - 2024.
85 f.: il.: color., gráf..

Orientador: Prof. Dr. Marcelo Moreira Tiago.
Monografia (Bacharelado). Universidade Federal de Ouro Preto. Instituto de Ciências Exatas e Aplicadas. Graduação em Engenharia Elétrica .

1. Administração do tempo - processamento de imagens - acústicas. 2. Algoritmos paralelos. 3. Algoritmos sequenciais. 4. Imagens acústicas. 5. Processamento de imagens. I. Tiago, Marcelo Moreira. II. Universidade Federal de Ouro Preto. III. Título.

CDU 004.932

Bibliotecário(a) Responsável: Flavia Reis - CRB6-2431

MINISTÉRIO DA EDUCAÇÃO
UNIVERSIDADE FEDERAL DE OURO PRETO
REITORIA
INSTITUTO DE CIENCIAS EXATAS E APLICADAS
DEPARTAMENTO DE ENGENHARIA ELETRICA

FOLHA DE APROVAÇÃO

Mateus Henrique Goncalves

Estratégias para redução dos tempos de execução de algoritmos SAFT para formação de imagens acústicas usando processadores *quad-core*

Monografia apresentada ao Curso de Engenharia Elétrica da Universidade Federal de Ouro Preto como requisito parcial para obtenção do título de bacharel em Engenharia Elétrica.

Aprovada em 17 de setembro de 2024.

Membros da banca

Dr. Marcelo Moreira Tiago — Orientador — Universidade Federal de Ouro Preto
Dr. Glauco Ferreira Gazel Yared — Universidade Federal de Ouro Preto
Dr. Júlio César Eduardo de Souza — Universidade Estadual Paulista "Júlio de Mesquita Filho"

Marcelo Moreira Tiago, orientador do trabalho, aprovou a versão final e autorizou seu depósito na Biblioteca Digital de Trabalhos de Conclusão de Curso da UFOP em 19/09/2024.

Documento assinado eletronicamente por **Marcelo Moreira Tiago**, **PROFESSOR DE MAGISTERIO SUPERIOR**, em 19/09/2024, às 10:29, conforme horário oficial de Brasília, com fundamento no art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015.

A autenticidade deste documento pode ser conferida no site http://sei.ufop.br/sei/controlador_externo.php?acao=documento_conferir&id_orgao_acesso_externo=0 , informando o código verificador **0780382** e o código CRC **1C7E0838**.

R. Diogo de Vasconcelos, 122, - Bairro Pilar Ouro Preto/MG, CEP 35402-163
Telefone: (31)3808-0818 - www.ufop.br

Dedico este trabalho aos meus pais.

Agradecimentos

Primeiramente, agradeço a Deus pela vida, e por ser Ele a razão da minha fé, principalmente nos dias de turbulência, dificuldades, aflições, e dúvidas que naturalmente vieram durante esta jornada.

Não poderia deixar de fazer um agradecimento especial aos meus pais, Lucinéia e Ivaldete, que estiveram presentes, de forma assídua, durante todos os momentos da minha vida. Sou grato pelos conselhos, pelo apoio incondicional para tornar este feito realizável, e por não medirem esforços para me fazerem bem.

Faço menção de agradecer ao professor Marcelo, pelas reuniões que fizemos, orientações, e conselhos para a construção deste trabalho, além do conhecimento passado em outros momentos da graduação.

Gostaria de citar também os colegas de curso que estiveram juntos comigo durante este tempo na UFOP, dividindo conhecimento, experiências, e risadas.

Por último, como são muitas pessoas envolvidas, quero demonstrar gratidão também aos demais que foram importantes, direta ou indiretamente, para a realização deste feito.

"Confie no Senhor de todo o seu coração,
e não se apoie em seu próprio entendimento.
Reconheça-o em todos os seus caminhos,
e Ele endireitará suas veredas."
Provérbios 3:5-6

Resumo

A geração de imagens acústicas tem, como um dos propósitos, avaliar a integralidade de objetos e materiais. Para cumprir essa finalidade existem diferentes abordagens, sendo que algumas apresentam alto custo computacional. Desta forma, o objetivo deste trabalho é apresentar uma análise comparativa do desempenho entre algoritmos sequenciais e paralelos na geração de imagens acústicas, com diferentes técnicas de abertura sintética e implementações em Matlab e *C*, considerando funções MEX, que permitem a transferência de dados bidirecional entre Matlab e *C*. A plataforma utilizada para os testes foi um computador Lenovo modelo ideapad 330, com processador Intel Core i5-8250U e 8 GB de memória RAM. As métricas utilizadas para a avaliação são a resolução lateral da imagem, e principalmente o tempo de processamento. Dentre os casos comparados, considerando uma imagem de dimensão 20 mm x 20 mm, resolução igual a λ, e método de focalização total SAFT-TFM, o pior cenário apresentou um tempo de processamento no valor de 12,32 s, enquanto na melhor abordagem este resultado foi reduzido para 1,79 s, uma diferença significativa de 10,53 s. Os resultados demonstram que implementações concorrentes em linguagem de nível inferior apresentam substancial ganho no tempo de processamento, ao manter o mesmo *hardware*.

Palavras-chave: Imagens Acústicas; Abertura Sintética; Algoritmos Sequenciais; Algoritmos Paralelos; Tempo de execução.

Abstract

The generation of acoustic images has, as one of its purposes, the evaluation of the integrity of objects and materials. To achieve this goal, different approaches are used, some of which have high computational costs. Therefore, the objective of this work is to present a comparative performance analysis between sequential and parallel algorithms in the generation of acoustic images, utilizing different synthetic aperture techniques and implementations in Matlab and *C*, considering MEX functions, which allow bidirectional data transfer between Matlab and *C*. The platform used for testing was a Lenovo ideapad 330 computer, equipped with an Intel Core i5-8250U processor and 8 GB of RAM. The metrics used for evaluation are the lateral image resolution, and primarily the processing time. Among the cases compared, considering an image with dimensions of 20 mm x 20 mm, resolution equal to λ, and using the SAFT-TFM total focusing method, the worst-case scenario presented a processing time of 12.32 s, while in the best approach, this result was reduced to 1.79 s, a significant difference of 10.53 s. The results demonstrate that concurrent implementations in a lower-level language provide a substantial gain in processing time while maintaining the same *hardware*.

Keywords: Acoustic Images; Synthetic Aperture; Sequential Algorithms; Parallel Algorithms; Execution Time.

Lista de algoritmos

Lista de ilustrações

Lista de siglas e abreviaturas

2R-SAFT	técnica de focalização por abertura sintética com 2 receptores (2R-SAFT, do inglês *2 Receivers - Synthetic Aperture Focusing Technique*)
API	interface de programação de aplicações (API, do inglês *Application Programming Interface*)
CPU	unidade central de processamento (CPU, do inglês *Central Processing Unit*)
CRT	tempo de execução em *C* (CRT, do inglês *C Runtime*)
DAB-SAFT	técnica de focalização por abertura sintética - baseado em desenho de arco (DAB-SAFT, do inglês *Drawing Arc Based - Synthetic Aperture Focusing Technique*)
DDR	taxa de dados dupla (DDR, do inglês *Double Data Rate*)
DSP	processador digital de sinal (DSP, do inglês *Digital Signal Processing*)
FPGA	matriz de portas programáveis por campo (FPGA, do inglês *Field Programmable Gate Array*)
GPU	unidade de processamento gráfico (GPU, do inglês *Graphic Processing Unit*)
HRI	imagem de alta resolução (HRI, do inglês *High Resolution Image*)
LRI	imagem de baixa resolução (LRI, do inglês *Low Resolution Image*)
MEX	executável *Matlab* (MEX, do inglês *Matlab Executable*)
MIMD	múltiplos dados de múltiplas instruções (MIMD, do inglês *Multiple Instruction Multiple Data*)
MISD	dados únicos de múltiplas instruções (MISD, do inglês *Multiple Instruction Single Data*)
MSAF	focalização por abertura sintética com multielementos (MSAF, do inglês *Multi-element Synthetic Aperture Focusing*)
NDE	avaliação não destrutiva (NDE, do inglês *Non-Destructive Evaluation*)
PA	matriz faseada (PA, do inglês *Phased Array*)
PCI	imagem com fase coerente (PCI, do inglês *Phase-Coherent Imaging*)

PCI-E	interconexão de componentes periféricos expressa (PCI-E, do inglês *Peripheral Component Interconnect Express*)
RAM	memória de acesso aleatório (RAM, do inglês *Random Access Memory*)
SA	abertura sintética (SA, do inglês *Synthetic Aperture*)
SAFT	técnica de focalização por abertura sintética (SAFT, do inglês *Synthetic Aperture Focusing Technique*)
SAFT-TFM	técnica de focalização por abertura sintética - método de focalização total (SAFT-TFM, do inglês *Synthetic Aperture Focusing Technique - Total Focusing Method*)
SCI	imagem de composição espacial (SCI, do inglês *Spatial Compounding Imaging*)
SIMD	múltiplos dados de instrução única (SIMD, do inglês *Single Instruction Multiple Data*)
TOF	tempo de voo (TOF, do inglês *Time of Flight*)

Lista de símbolos

α	coeficiente de atenuação
λ	comprimento de onda
ρ	densidade do material
c	velocidade do som
D	diâmetro do elemento ou abertura
f	frequência
k	constante de proporção
L	comprimento do elemento ou da abertura
N	comprimento da região de campo próximo
T	período
Z	impedância acústica

Sumário

1 Introdução

1.1 Contextualização

As técnicas de processamento de imagem utilizando ondas ultrassônicas vêm ganhando cada vez mais espaço em diferentes aplicações, podendo ser empregadas, por exemplo, em ambientes médicos (diagnósticos por imagem, aplicações odontológicas e tratamentos fisioterápicos) e industriais (química, petroquímica, farmacêutica e metalúrgica). A diversidade no campo de aplicações pode ser justificada pela possibilidade de se realizar a avaliação não destrutiva (NDE, do inglês *Non-Destructive Evaluation*) de um objeto ou estrutura com grande poder de penetração, comparado a técnicas de imagem que utilizam ondas eletromagnéticas, como o radar, viabilizando o aproveitamento de técnicas de processamento de imagem.

Em seu início, as técnicas que utilizam *arrays* e focalização por abertura sintética tiveram origem a partir da abstração de métodos utilizados em radares, sendo que ambos possuem semelhanças quanto ao funcionamento (utilizam ondas para mapear e visualizar objetos ou estruturas), e diferenciam-se pela natureza da onda envolvida, frequência e aplicações específicas.

O funcionamento de um equipamento matriz faseada (PA, do inglês *Phased Array*) baseia-se em sistemas de excitação e aquisição compostos por transdutores piezoelétricos. Transdutores piezelétricos são instrumentos capazes de converter energia mecânica (pressão, por exemplo) em energia elétrica (diferença de potencial), podendo utilizar como material constituinte, cristais de quartzo, semicondutores, compósitos, polímeros e cerâmicas.

Pode-se dizer que o volume de dados adquiridos é influenciado diretamente pela quantidade, disposição, e formato do transdutor que compõe o equipamento usado. À medida que a demanda por imagens com melhores resoluções geradas em tempo real aumentam, a complexidade do sistema eletrônico responsável por adquirir e processar os sinais também é elevada, principalmente na etapa de aquisição. Dessa forma, o melhor aproveitamento da capacidade computacional hoje disponível, bem como técnicas de pós-processamento mais sofisticadas, tornaram-se indispensáveis.

Com o intuito de reduzir os tempos de processamento necessários, são propostas atualmente técnicas para executar os algoritmos de forma concorrente, por meio de paralelização das tarefas. Essas funções podem ser processadas em placas gráficas unidade de processamento gráfico (GPU, do inglês *Graphic Processing Unit*) ou por meio de matriz de portas programáveis por campo (FPGA, do inglês *Field Programmable Gate Array*). O objetivo dessas abordagens, é justamente dividir de forma equitativa o esforço computacional entre todos os núcleos disponíveis, ou utilizar as unidades gráficas presentes.

Basicamente, as técnicas propostas tentam alterar a forma que os algoritmos são implementados, substituindo o modo de execução sequencial por uma versão capaz de executar as tarefas de forma concorrente.

Seguindo este conceito de paralelização, a proposta deste trabalho é propriamente comparar o desempenho de três técnicas para formação de imagens acústicas, em implementações sequenciais e paralelas. Inicialmente, serão realizados testes utilizando algoritmos para implementar diferentes técnicas de focalização a partir do *software* Matlab. Posteriormente, serão realizadas implementações com algoritmos desenvolvidos em linguagem *C*. Serão feitas comparações entre os tempos de execução desses algoritmos no formato sequencial e concorrente, utilizando diferentes abordagens para divisão das tarefas de forma equilibrada entre os núcleos de processamento disponíveis no sistema.

1.2 Objetivos

O objetivo geral deste trabalho é comparar os tempos de execução das diferentes técnicas, usando variadas formas de implementação dos algoritmos. Para alcançar o objetivo geral, os objetivos específicos são:

- Desenvolver algoritmos para as técnicas de focalização SAFT, 2R-SAFT, e SAFT-TFM;
- Métodos para implementação concorrente em Matlab e em *C*;
- Medição do tempo de execução das implementações feitas;
- Análise, comparação e discussão dos tempos de execução dos algoritmos.

1.3 Revisão bibliográfica

Esta seção apresenta uma revisão contendo trabalhos que apresentam estudos utilizando *phased arrays*, técnicas de focalização para imagens ultrassônicas, e paralelização de algoritmos de processamento de imagens utilizando, majoritariamente, GPU. Os materiais aqui referenciados abordam, direta ou indiretamente, conteúdos que este trabalho pretende apresentar.

No trabalho apresentado por Drinkwater e Wilcox (2006), os autores apresentam uma revisão com os principais estudos, que até aquele momento, foram realizados sobre o tema, dando destaques a tópicos como transdutores, matrizes ultrassônicas, processamento de sinais, e outros pontos que envolvem a aplicação da avaliação não destrutiva. Os autores destacam que parâmetros como sensibilidade, tempo de execução e resolução ganharam maior relevância nos últimos anos, motivados pelos avanços tecnológicos e aplicações de tempo real.

Os autores destacam que, com a tentativa de aprimorar as imagens, surgiram diferentes configurações de transdutores, como o unidimensional ou linear (1-D), bidimensional (2-D) ou anular. Os *arrays* lineares (1-D) são os mais utilizados, consistindo em uma fileira de elementos em forma de tira. Os materiais utilizados na fabricação dos equipamentos, segundo apontam os autores, também foram modificados, sendo que no início a fabricação era feita com material piezo cerâmico (como chumbo). Posteriormente, passou-se a utilizar os materiais piezoelétricos, devido a sua baixa impedância acústica.

Destaca-se também as modalidades de inspeção que os autores citam, sendo as principais: *B-scan* e *focused B-scan*. Na varredura *B-scan*, um subconjunto dos elementos da matriz (chamada de abertura) é usado para inspecionar o componente. O transdutor é deslocado ao longo da peça analisada para formar uma imagem, sendo que os elementos dentro da abertura podem ser disparados em fase (para produzir um feixe plano) ou com atrasos (para produzir um feixe focalizado). O modo *focused B-scan* diferencia-se justamente por utilizar disparos em atraso. Ressalta-se que esse deslocamento pode ou não ser feito, levando em consideração as dimensões da peça analisada.

Para peças menores ou iguais ao *array*, o deslocamento não seria necessário, podendo ser utilizado outros subconjuntos de elementos do *array* para eliminar o deslocamento ao longo da peça. No trabalho realizado, o foco passou a estar direcionado para os desafios que as pesquisas sobre esse tema enfrentariam, que se referem principalmente a maiores exigências de instrumentos eletrônicos e capacidade computacional, para lidar com imagens em tempo real, imagens 3-D, maiores taxas de quadros e melhores resoluções.

Jensen et al. (2006) estudaram a aplicabilidade da abertura sintética utilizada em radares em sistemas para geração de imagens por ultrassom, bem como seu princípio de funcionamento, vantagens e desvantagens ao utilizá-la. O estudo ressalta que o método básico para focalização de imagens ultrassônicas por SAFT, é a emissão de onda de um único elemento por vez, e recepção por parte de todos os elementos presentes no transdutor (ou abertura). Essa dinâmica gera N imagens de baixa resolução, sendo N o número de elementos na abertura. Após essa aquisição, a combinação das imagens de baixa resolução pode resultar em uma imagem de alta resolução, uma vez que a focalização foi realizada para todos os pontos da imagem. As grandes desvantagens por parte dessa técnica estão relacionadas com a multiplexação dos elementos (pois um único elemento emite), fluxo e movimento. Tais problemas podem ser sanados combinando vários elementos para transmissão, e usando sinais com maior amplitude.

Holmes, Drinkwater e Wilcox (2005) abordam em seu estudo as vantagens significativas de desempenho que as técnicas oferecem à NDE, permitindo, por exemplo, que a matriz seja focada em todos os pontos do alvo. Os autores destacam a capacidade de efetivar múltiplas inspeções sem a necessidade de reconfiguração, representando melhor potencial para sensibilidade e cobertura.

O método de pós-processamento usado é o SAFT-TFM, onde o feixe é focalizado

em todos os pontos da região alvo. Essa técnica é comparada com as varreduras *B-scan* e *focused B-scan*, utilizando para isso, a geração de imagens a partir de cada método. A abordagem SAFT-TFM apresentou melhores resultados, que se justificam pelo fato de todos os elementos do *array* terem sido usados para a focalização, permitindo também, cobertura de áreas maiores.

Essas melhorias são significativas, tendo em vista que nos modelos convencionais, ao ser usado apenas um único elemento na transmissão, a potência acústica total de saída diminui, o que leva a uma baixa relação sinai-ruído. A principal desvantagem dos métodos de pós-processamento destacada pelos autores é a maior exigência computacional, e maiores intervalos de tempo de execução.

Zhang, Drinkwater e Wilcox (2013) citam alguns algoritmos utilizados para formação de imagens ultrassônicas, que foram amplamente utilizados e desenvolvidos em ensaios não destrutivos, NDE. Dentre os algoritmos citados, destacam-se o imagem com fase coerente (PCI, do inglês *Phase-Coherent Imaging*), imagem de composição espacial (SCI, do inglês *Spatial Compounding Imaging*) e, novamente, SAFT-TFM. Como poderá ser visto no Capítulo 3 deste trabalho, entre os 3 métodos citados, o SAFT-TFM será tratado com maior profundidade. Entretanto, é importante observar as comparações realizadas.

Especificamente, PCI e SCI não são métodos para formação de imagens, sendo usados principalmente para eliminação de ruído, que consequentemente melhora a qualidade da imagem. Os algoritmos foram avaliados pela resolução, robustez para diferentes tipos de defeitos em uma estrutura, e relação sinal-ruído. Definindo a resolução como a menor distância de separação entre dois pontos, o método PCI apresentou o melhor resultado, entre 0,52 mm e 0,51 mm, enquanto para SAFT-TFM o resultado foi de 0,6 mm, e 0,7 mm para SCI.

A robustez (resolução lateral) foi comparada por meio de diferentes tipos de defeitos, e os resultados do SAFT-TFM mostraram uma distribuição mais concentrada em todos os casos, o que indica um desempenho de imagem mais consistente. Com relação ao sinal-ruído, de modo geral a implementação com o PCI apresenta melhores resultados, mesmo com pequenas diferenças. A escolha pelo PCI deve ser criteriosa, pois podem surgir distorções nos resultados (em imagens de trincas, por exemplo).

No estudo desenvolvido por Karaman, Bilge e O'Donnell (1998), técnicas de abertura sintética com vários elementos em disparos sucessivos foram aplicadas para melhorar a qualidade das imagens. Entretanto, os autores destacam que esses métodos são suscetíveis a artefatos de movimento e saltos de fase. Em um sistema de imagem PA convencional, todos os elementos do transdutor podem ser utilizados para emitir e receber sinais, porém com alto custo eletrônico e financeiro.

Diferentemente da focalização no PA, que é eletrônica, em SAFT a composição da abertura é feita de forma sintética, a partir da multiplexação de uma pequena abertura ativa em uma matriz. As abordagens SAFT tornaram-se atraentes para sistemas

mais complexos, apresentando, porém, problemas com a fase e produção de artefatos de movimento.

No trabalho, os autores utilizaram a técnica MSAF, que utiliza múltiplos elementos para criar uma imagem de abertura sintética, combinando sinais adquiridos por diferentes elementos do transdutor. A abordagem explora a correlação entre os sinais, usando frequências espaciais comuns de sub aberturas de recepção sobrepostas para determinar o erro de fase entre etapas de disparo consecutivas devido ao movimento. Como comparação, para se corrigir anomalias de fase em sistemas PA, pode-se empregar vários disparos. Como contrapartida, a taxa de atualização das imagens (quadros/s) será reduzida. Dessa forma, a técnica MSAF é muito adequada para processamento em paralelo, que como poderá ser visto, está sendo amplamente usado atualmente.

Outro estudo que também explora a abordagem de abertura sintética com múltiplos elementos, foi o apresentado por Karaman, Li e O'Donnell (1995). Nesse tipo de sistema com abertura sintética (SA, do inglês *Synthetic Aperture*), apenas um elemento emite (acionado sequencialmente) e todos os outros recebem. Sendo assim, a potência acústica transmitida de um único elemento da matriz limita a relação sinal-ruído, representando baixa resolução de contraste. Os autores demonstram que a técnica com múltiplos elementos emitindo minimiza o problema da resolução lateral, aumenta a relação sinal-ruído e contraste-ruído, apresentando resultados que se aproximam da qualidade do PA.

Rasmussen e Jensen (2014) abordam e comparam estratégias para formação de imagens 3D em tempo real, como a formação de feixe paralelo (quando as ondas se propagam de forma paralela, sendo obtida ajustando-se a geometria e o formato do transdutor e, se necessário, utilizando elementos de lente acústica), e abertura sintética. O sistema desenvolvido foi aplicado para a geração de imagens cardíacas, que demandam reproduções com profundidade de até 15 cm, com uma taxa de quadros mínima de 20 Hz. Os autores propuseram um sistema composto por um *array* com 256 elementos ativos, comparando os resultados aos obtidos por sistemas comerciais, equipados com *arrays* com mais de 9000 elementos ativos e alto custo de produção

Pode-se perceber que a formação pelo método de feixe paralelo é capaz de aumentar a taxa de quadros da imagem de forma proporcional ao número de linhas formadas por emissão. Isso se dá pelo fato da técnica ampliar o feixe de transmissão, enquanto vários receptores são focalizados em paralelo. Em compensação, essa abordagem perde em resolução, o que impacta na profundidade da imagem reproduzida, bem como no nível de detalhes do resultado.

Em contrapartida, quando utilizada a abertura sintética, o desempenho por parte da resolução foi melhor em todas as posições da imagem, apresentando maiores contrastes e profundidade de penetração. Isso está relacionado com uma melhor relação sinal-ruído, quando comparada à formação de feixe paralelo.

Sloun et al. (2019) apresentam um estudo relacionado à paralelização de algoritmos utilizados para gerar imagens vasculares com alta resolução. Para tal cobertura, longos tempos de aquisição são necessários, o que torna o pós-processamento custoso computacionalmente. No trabalho os autores propõem a utilização de aprendizagem profunda junto à localização microscópica por ultrassom para atingir imagens com resolução de até 128x128 *pixels*.

A melhor resolução dificulta certas funcionalidades que, nesse cenário, referem-se à diminuição da profundidade de penetração. Isso se deve à necessidade de utilizar frequências mais elevadas, o que faz com que ondas acústicas sofram maior atenuação. Os autores demonstram que a técnica proposta torna essa implementação em tempo real mais viável, apesar de ainda se tratar de um grande desafio.

Ainda no contexto do uso do capacidade de processamento, Birk et al. (2014) propõem técnicas de processamento de imagens envolvendo tomografia computadorizada. Os autores destacam que imagens volumétricas de alta qualidade exigem alto custo computacional, e recomendam a utilização de GPUs, operando em conjunto com uma CPU, como plataforma de processamento para as imagens.

Os autores comparam as duas abordagens (CPUxGPU), quanto ao tempo de execução do algoritmo em determinadas resoluções e posições de abertura. Um processador Intel® Core™ i7-2700K e uma GPU Nvidia GeForce GTX 590 foram utilizados. Os resultados mostram que para resoluções laterais (nitidez na largura) e axiais (nitidez na profundidade) constantes, os tempos de execução aumentam linearmente em função do número de elementos da SA (n° de elementos que emitem/recebem). Os autores destacam que o desempenho das GPUs mostrou-se mais eficiente comparada à CPU, apresentando tempos de execução menores em todas as situações testadas. Esse resultado demonstra uma possível tendência envolvendo essa tema, que é a de se empregar cada vez mais placas de processamento gráfico.

O estudo desenvolvido por Yiu, Tsang e Yu (2011) corrobora essa tendência, que teve como objetivo o desenvolvimento de uma arquitetura com a capacidade de processamento paralelo em tempo real necessária para se utilizar técnicas de formação de imagens, como a abertura sintética, e a composição de ondas planas. A arquitetura desenvolvida destaca-se pela utilização de GPUs, sendo um grupo para processamento de imagens de alta resolução (HRIs), e os demais com a função de gerar as imagens de baixa resolução (LRIs).

LRIs são formadas a partir de cada disparo recebido em todos os elementos do equipamento transdutor, enquanto as imagens ditas de alta resolução, HRIs, são obtidas por meio da soma recursiva de uma série de LRIs. Computacionalmente, cada *pixel* HRI é calculado através da soma de vários *pixels* LRIs na mesma posição. No sistema utilizado, as tarefas foram divididas e atribuídas aos *threads* que compõem a GPU. Pode-se ressaltar também outra vantagem que GPUs apresentam: sua programabilidade torna-a uma opção

mais acessível quando comparada a outros hardwares, como FPGAs e DSPs.

Præsius et al. (2022) apresentam outra possibilidade de aplicação de imagens ultrassônicas, que é a de análise de vasos sanguíneos com até 50 μm. Os autores descrevem com detalhes o *hardware* utilizado, bem como as etapas do método usado para gerar imagens com elevadas resoluções. Foi empregado uma GPU Nvidia GeForce RTX™ 3090 com um processador Intel® Core™ CPU i9-11900K, sendo utilizado o *software* Matlab R2021b.

A GPU usada serviu para realizar cálculos para correção de movimento, e o sistema de aquisição utilizado contou com um *array* de 256 elementos, frequência de amostragem de 62,50 MHz, possibilitando grandes volumes de dados durante o processo de aquisição. Esses detalhes foram importantes, pois possibilitaram a visualização de imagens com super resolução da microvasculatura.

Yang, Qin e Li (2014) destacam as limitações da técnica SAFT para a formação de imagens em objetos com múltiplas camadas. Em seu trabalho, os autores propõem uma abordagem diferente, denominada técnica de focalização por abertura sintética - baseado em desenho de arco (DAB-SAFT, do inglês *Drawing Arc Based - Synthetic Aperture Focusing Technique*), que se trata de uma técnica mais avançada e complexa (em se tratando de cálculos).

Objetos com múltiplas camadas são compostos de vários revestimentos com meios anisotrópicos (suas propriedades mecânicas variam em diferentes direções) e não homogêneos, seja nas laterais, na profundidade, ou em ambas as direções. O método de SAFT é ineficiente para esses objetos devido ao caminho percorrido pela onda sonora não ser uma linha reta, devido à refração do som nas diferentes interfaces.

O método DAB-SAFT foi proposto com base no princípio do algoritmo de conversão de varredura em círculos para economizar muitas operações computacionais em contraste com o SAFT tradicional. Os autores destacam que a complexidade computacional do método DAB-SAFT é menor do que a do SAFT, sendo uma opção mais viável para esses casos.

Kjeldsen et al. (2014) utilizam processadores paralelos, como CPUs e GPUs multicore. A metodologia abordada consistiu na comparação entre as implementações com CPU Intel Core i7, e GPUs AMD HD7850 e Nvidia GTX 680. A implementação usando CPU foi feita na linguagem *C* com a interface de programação *OpenMP*, enquanto para as GPUs foram usadas e comparadas as APIs *OpenCl* e *OpenGL*. Para processar imagens geradas a partir de sistemas com *arrays* de transdutores com mais elementos, os autores destacam um sistema típico, que utiliza 128 canais, conversores AD resolução de 12 bits e frequências de amostragem de 40 MHZ, operando com uma largura de banda de 9,54 GB/s.

O estudo apresenta comparações entre implementações para formação de imagem em diferentes linguagens e *softwares*, como Matlab, SIMD, *OpenCl* (arquitetura para programação de plataformas heterogêneas) e *OpenGl* (API utilizada para computação

gráfica). As principais diferenças observadas pelos autores envolvem os cálculos da raiz dos erros quadráticos médios, e na razão sinal-ruído (SNR). A justificativa para essas desigualdades pode ser baseada no fato de que o Matlab trabalha com precisão numérica de ponto flutuante em 64 bits, enquanto as demais usam representações de até 32 bits.

No trabalho, os tempos medidos são obtidos com uma CPU Intel® Core™ i7-2600, e GPUs AMD HD7850, e Nvidia GTX 680. Como comparação, o tempo para formação do feixe utilizando apenas 1 *thread* foi de 2,95 s, enquanto para 8 *threads*, levou-se 699 ms. Quanto a GPUs, a implementação feita na arquitetura *OpenCL* se mostrou ligeiramente mais eficiente comparada à API *OpenGL*, e a placa da AMD foi relativamente mais rápida do que a unidade da Nvidia, com uma ligeira diferença de 0,12 ms (0,44 ms para AMD, e 0,56 ms para Nvidia). De modo geral, a utilização de GPUs se mostrou consideravelmente mais eficiente do que a implementação em CPU.

Ainda sobre paralelização e comparação entre tempos de execução, Iriarte, Cosarinsky e Brizuela (2016) apresentam um estudo sobre os ensaios não destrutivos (NDE) e técnicas de pós-processamento que demandam elevado custo computacional como, por exemplo, SAFT-TFM. A técnica proposta baseia-se no método pulso-eco, e a imagem é gerada pela adição de sinais ultrassônicos adquiridos em diferentes posições. A intensidade de cada *pixel* é calculada conforme a onda refletida em cada local da peça.

Na implementação, utilizou-se um processador Intel® Core™ i7-3770, com algoritmos desenvolvidos em Matlab, linguagem *C*, e na API *OpenCL*, utilizando GPU. Para o caso do script em Matlab, o tempo requerido para gerar a imagem de saída foi de 41,18 s, para a implementação em *C*, levou-se 11,31 s, e para *OpenCL* o tempo demandado foi de 16 ms.

Esses resultados mostram que o uso de uma plataforma de processamento paralelo apresenta ótimos resultados, comparado aos tempos decorridos em Matlab e em *C* (implementação sequencial), que ficam limitados em casos onde precisa-se gerar imagens com maior resolução obtidas a partir de um número maior de sinais adquiridos. Com base nessas observações, neste trabalho serão realizadas comparações entre os tempos de execução dos algoritmos SAFT, 2R-SAFT, e SAFT-TFM. Espera-se que os tempos necessários para processar esses algoritmos utilizando programação sequencial sejam maiores do que os obtidos utilizando-se programação paralela. Além disso, espera-se que os algoritmos desenvolvidos em linguagem *C* sejam executados em um tempo menor do que os implementados em Matlab.

1.4 Estrutura do trabalho

A organização deste trabalho está estruturada no seguinte formato:

No Capítulo 2, apresenta-se uma revisão teórica que engloba conceitos sobre teoria de geração de imagens por ultrassom, abordando-se temas como: transdutores, ondas

acústicas longitudinais e transversais, velocidade, atenuação, campo acústico, equipamentos ultrassônicos, métodos de medição, e algoritmos para formação de imagens acústicas, desde os métodos de focalização, até as estratégias adotadas para redução do tempo de processamento.

No Capítulo 3 define-se a metodologia adotada e a plataforma de testes, incluindo a descrição do procedimento experimental empregado para capturar os sinais necessários para as etapas posteriores e a exibição de uma imagem da peça, com informações sobre as especificações do equipamento transdutor, propriedades do material, dimensões, frequência de amostragem do sistema de aquisição, e uma descrição da disposição da matriz de pontos que será utilizada.

Além disso, ainda é mostrada a estratégia por trás da implementação dos algoritmos sequencias SAFT, 2R-SAFT, e SAFT-TFM, dispondo de pseudocódigos para cada abordagem. O método utilizado para a paralelização, em Matlab e *C*, também é descrito.

No Capítulo 4, são apresentados os resultados, comparando cada caso por meio da apresentação das imagens e do tempo de execução decorrido. Finalmente, no Capítulo 5 são feitas as considerações finais, além da apresentação de propostas para futuros estudos sobre a temática deste trabalho.

2 Revisão teórica

Este Capítulo apresenta uma revisão teórica para os conteúdos que serão tratados posteriormente como, por exemplo, conceitos básicos envolvendo propagação de ondas pelo meio, tais como técnicas para cálculo de velocidade de propagação, coeficiente de atenuação, impedância acústica, diretividade, campo próximo e campo distante, e transdutores.

Além disso, este Capítulo também apresenta uma revisão sobre sistemas *phased array*, tais como métodos de controle de disparo dos elementos do transdutor, técnicas de conformação de feixe, e métodos de aquisição de sinais.

O Capítulo prossegue com a apresentação de técnicas por abertura sintética SA utilizadas no processo de formação de imagens acústicas, tais como o método de focalização SAFT, 2R-SAFT, e SAFT-TFM. Finalmente, são discutidos assuntos como a utilização da *Parallel Computing Toolbox*, funções *MEX* no Matlab, e a biblioteca de paralelização de tarefas para linguagem *C*, *windows.h*. No que diz respeito à relação do conteúdo deste Capítulo com os assuntos posteriores, podem ser destacadas as descrições feitas para os equipamentos ultrassônicos, diferença entre as terminologias envolvendo a paralelização de algoritmos, como *workers* e *threads*, e implementações a serem contempladas posteriormente.

2.1 Ondas acústicas e transdutores

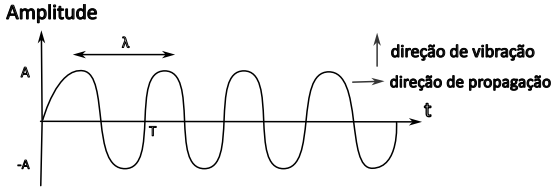
Em equipamentos de ultrassom, os transdutores são os instrumentos responsáveis por emitir e receber ondas acústicas. Esses dispositivos são capazes de transformar uma forma de energia (pressão, para o caso do ultrassom) em diferença de potencial, ou vice-versa. Para compreender a forma com que transdutores são usados para geração de imagens acústicas, antes é importante compreender conceitos e características acerca de ondas acústicas, tais como modos de propagação e propriedades. Esses pontos serão apresentados na sequência.

2.1.1 Ondas transversais e longitudinais

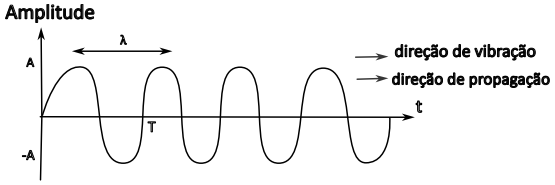
Uma onda transversal é aquela em que as partículas do meio oscilam em uma direção normal à direção de propagação de onda. As ondas longitudinais são aquelas em que a vibração das partículas se dá de forma paralela (na mesma direção) com relação à direção de propagação de onda. A Figura 1 apresenta duas imagens que representam esses dois tipos de onda. Na Figura, o comprimento de onda (λ) [m], e o período (T) [s] são também representados.

Figura 1 – Tipos de onda.

(a) Onda transversal.



(b) Onda longitudinal.



Fonte: Adaptado de Costa et al. (2004).

2.1.2 Velocidade, impedância acústica, e atenuação.

De forma análoga a ondas eletromagnéticas, ondas acústicas experimentam fenômenos como reflexão, refração e absorção ao se propagarem em um meio específico. A velocidade do som (c) [m/s] com que uma onda acústica se propaga em determinado meio, depende do seu comprimento de onda λ e da frequência (f) [Hz]. A equação 2.1 mostra a relação entre essas variáveis.

$$c = \lambda \cdot f \tag{2.1}$$

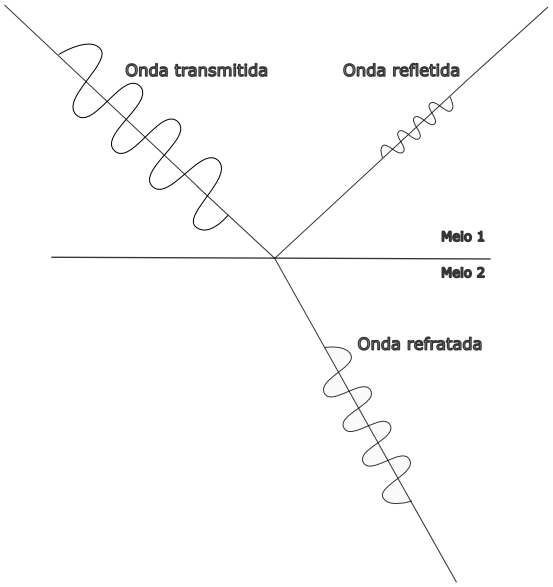
A propagação de ondas acústicas depende diretamente do meio pelo qual a onda se propaga. A impedância acústica (Z) [Rayls] é o parâmetro utilizado para relacionar a velocidade de propagação do som c com a densidade do material (ρ) [kg/m^3], e pode ser calculada a partir da equação 2.2.

$$Z = \rho \cdot c \tag{2.2}$$

Esses fatores, impedância acústica Z, densidade ρ, e velocidade c, influenciam diretamente no comportamento de uma onda sonora na interface entre dois meios compostos por materiais distintos, onde uma parcela da onda é refletida, e a outra é transmitida, ou refratada, conforme é mostrado na Figura 2. A reflexão de ondas sonoras em uma superfície ocorre quando há uma mudança na impedância acústica entre dois meios, podendo ocorrer reflexão total ou parcial.

A reflexão total é quando quase toda a energia das ondas acústicas é refletida de volta para o primeiro meio, e apenas uma pequena porção é transmitida para o segundo meio. Ela ocorre nos casos em que a impedância acústica Z do segundo meio é significativamente maior do que a do primeiro meio. Por outro lado, a reflexão parcial ocorre quando apenas uma fração da energia das ondas acústicas é refletida, e outra parcela é transmitida para o segundo meio. Esse caso ocorre quando a diferença entre as impedâncias acústicas dos dois meios não é significativa. Quando a onda experimenta uma transição para um meio com impedância acústica muito menor, há pouca parcela de reflexão da onda.

Figura 2 – Comportamento de uma onda sonora na transição entre dois meios diferentes.



Fonte: Adaptado de Costa et al. (2004).

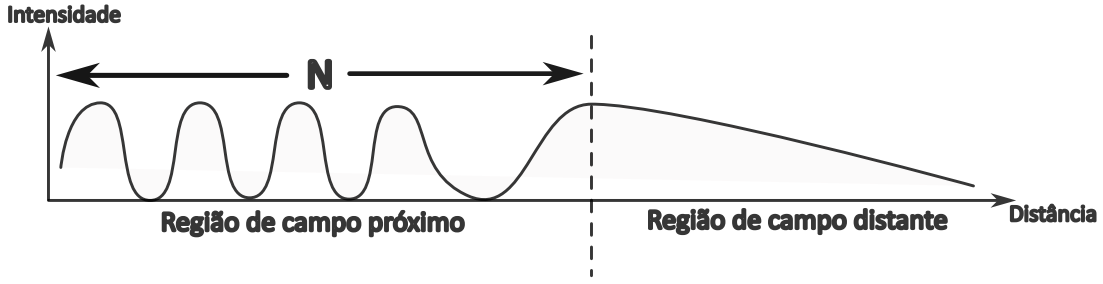
Os pontos citados, como velocidade, impedância acústica, reflexão, e refração, estão associados a outra característica importante no que diz respeito à propagação de ondas, a atenuação. O coeficiente de atenuação (α) [Np/m] é utilizado para quantificar o decaimento da amplitude de uma onda ao se propagar por um determinado material. Alguns fatores contribuem para a atenuação de ondas acústicas, entre eles:

- Absorção: parte da energia presente nas ondas é absorvida pelo meio a qual se propagam, podendo ser maior ou menor, a depender de parâmetros como a viscosidade e a compressibilidade adiabática do meio;
- Difração: quando a onda contorna algum obstáculo, por meio da abertura do feixe acústico, inevitavelmente ela sofre um pouco de atenuação;
- Divergência do feixe com relação ao eixo central: implica em maiores distâncias de propagação, o que provoca diminuição da amplitude;

2.1.3 Campo acústico e diretividade

A descrição do campo acústico gerado por um transdutor pode ser dividida em duas regiões ao longo do eixo de propagação da onda. Uma dessas regiões se encontra próxima do transdutor e é denominada região de campo próximo, ou região de difração de *Fresnel*. Por sua vez, a outra região é a de campo distante, também chamada de região de *Fraunhofer*. Essas regiões são ilustradas na Figura 3.

Figura 3 – Variação da intensidade da onda em função da distância.



Fonte: Adaptado de Costa et al. (2004).

Nessas regiões, o comportamento ondulatório é diferente, sendo que na região de campo próximo, ocorre sobreposição entre as ondas de borda (geradas na periferia do transdutor) e as ondas diretas (geradas em toda a face do transdutor). No campo próximo, podem ocorrer interferências construtivas e destrutivas, o que resulta em máximos e mínimos na intensidade do campo acústico, e caracteriza essa região com a presença de feixes com pouca divergência.

Na região de campo distante, a diferença de fase entre as ondas de borda e as ondas diretas não são tão evidentes, resultando em interferência construtiva que forma uma frente de onda quase plana. Por essa razão, o campo distante é caracterizado por apresentar uma natureza ondulatória que tende a ser divergente. No entanto, à medida que essa onda se afasta da fonte, ela se atenua no meio de propagação, conforme visto na Figura 3. O comprimento da região de campo próximo (N) [m] varia conforme a equação 2.3, dependendo diretamente do comprimento do elemento ou da abertura (L) [m], da constante de proporção (k), da frequência f e velocidade c.

$$N = \frac{kL^2 f}{4c} = \frac{kL^2}{4\lambda} \tag{2.3}$$

Em elementos circulares, a equação deve substituir a constante k e o comprimento L pelo diâmetro do elemento ou abertura (D) [m], da forma conforme representada na equação 2.4.

$$N = \frac{D^2 f}{4c} = \frac{D^2}{4\lambda} \tag{2.4}$$

Um último importante conceito relacionado com as propriedades físicas das ondas ultrassônicas, é o de diretividade. Pode-se dizer que a diretividade refere-se à tendência que uma fonte tem de emitir o som em uma direção específica com maior intensidade do que em outras direções. Sendo assim, está mais associada às características da fonte sonora, não dependendo diretamente da distância (diferentemente da intensidade) entre o transdutor e o objeto alvo, ou ponto de observação.

Dentre os fatores que podem influenciar a diretividade, destacam-se:

- Obstáculos: reflexões e refrações de ondas acústicas em superfícies podem afetar a diretividade. Dependendo de certas condições, superfícies refletoras podem aumentar a diretividade, ao direcionar o som para áreas específicas. Da mesma forma, certas superfícies podem diminuir a diretividade, ao orientar a onda para regiões de pouco interesse;

- Frequência: a diretividade pode variar com a frequência do som emitido. Em geral, ondas mais direcionais são emitidas por fontes sonoras em frequências mais altas. Enquanto em frequências mais baixas, as ondas tendem a se espalhar de forma mais ampla;

- Formato e *design* da fonte: tanto a forma física da fonte, quanto o seu *design*, impactam na diretividade. Como exemplo, uma abertura mais estreita pode resultar em uma emissão mais direcional, comparado a uma superfície plana.

2.1.4 *Arrays*

Os transdutores ultrassônicos convencionais usados em aplicações acústicas geralmente consistem em um único elemento ativo que gera e recebe ondas mecânicas de alta frequência, ou dois elementos emparelhados, um para transmissão, e um para recepção. *Arrays* referem-se ao agrupamento de diversos desses elementos piezoelétricos em formato vetorial ou matricial. O desenvolvimento da eletrônica possibilitou um controle individual do tempo de disparo desses elementos transdutores. Esse controle, somado a sistemas de aquisição mais rápidos e com maior resolução, possibilitou o desenvolvimento dos *phased arrays*, PAs.

Em um *array*, os elementos são dispostos em uma grade fixa, e a varredura do feixe é geralmente realizada por meio do movimento físico do equipamento transdutor. Assim também é feito nos casos em que se muda a direção do feixe ultrassônico, ou seja, é obtida a partir da orientação física desse *array*. Além disso, a construção de determinados formatos desses transdutores se torna um desafio, por apresentar focalização de feixe fixa, limitando a flexibilidade desse tipo de configuração.

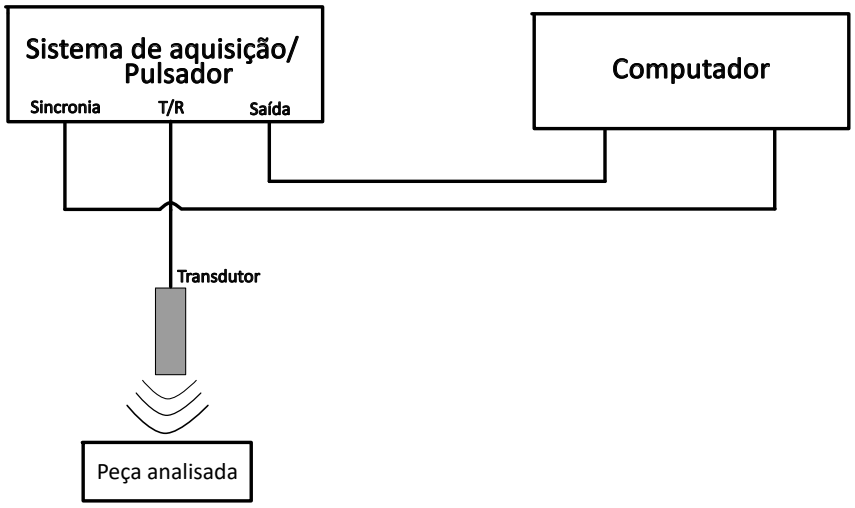
Tendo em vista o citado anteriormente, um *array* de transdutores, atualmente, só é aplicado em situações muito específicas, que exigem baixo custo. De forma geral, o PA apresenta vantagens em praticamente todos os pontos com relação ao *array* de transdutores. Essas vantagens serão descritas a seguir.

2.2 Equipamentos ultrassônicos

Nesta seção serão abordadas as principais características que compõem um equipamento ultrassônico, como seu sistema de excitação e aquisição, métodos de medição,

modo de funcionamento, e PAs. Um equipamento básico de ultrassom é formado pelas unidades ilustradas na Figura 4. Os elementos que o compõem são basicamente um pulsador, sistemas de controles e processamento, e sinais de entrada e saída.

Figura 4 – Equipamentos básicos que compõem um equipamento ultrassônico.



Fonte: Adaptado de Costa et al. (2004).

Os itens representados na Figura 4 são:

- T/R: presente no pulsador, ele é usado para geração e transmissão dos pulsos elétricos para excitação dos transdutores;
- Sincronia: responsável pelo controle e processamento, e configurações dos parâmetros das unidades de transmissão e recepção;
- Saída: envio dos sinais captados para o computador, sendo este último utilizado também como unidade para visualização dos resultados do processamento.

2.2.1 Sistemas de excitação e aquisição de sinais

A geração de ondas ultrassônicas, ocorre mediante a aplicação de um sinal elétrico (como, por exemplo, um pulso de curta duração, uma série de senoides, ou ondas retangulares) a um dispositivo transdutor. Normalmente, o transdutor entra em contato com o objeto em análise, utilizando um meio de acoplamento para maximizar a parcela de energia transmitida. Em geral, água, alguns tipos de óleo mineral e géis costumam ser utilizados para esta finalidade.

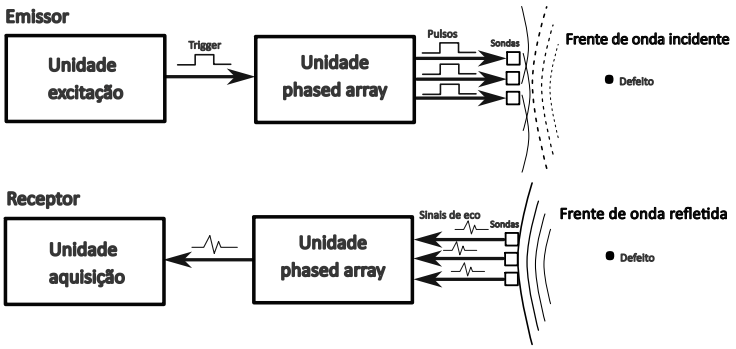
As ondas geradas pelo transdutor se propagam no interior do objeto, interagindo com suas distintas estruturas e gerando ondas refletidas (ecos), que se dispersam em múltiplas direções, inclusive na direção do transdutor emissor. O próprio transdutor, desempenhando o papel de receptor, detecta esses ecos. Conhecendo a velocidade de propagação do som no meio em questão, é possível calcular as distâncias pelas quais o sinal acústico

se propagou, de modo a identificar reflexões geradas por interfaces que apresentam valores de impedância acústica diferentes.

Para esse procedimento de varredura a partir de feixes controlados por computador, existem basicamente três abordagens, sendo elas: digitalização eletrônica, focalização de profundidade dinâmica, e digitalização setorial. A mais aplicada dentre as três, é a digitalização eletrônica, que aborda o procedimento de multiplexação da mesma lei focal e do mesmo atraso em um grupo de elementos ativos ao longo do equipamento PA. Na focalização de profundidade dinâmica, a digitalização é realizada utilizando diferentes profundidades focais, enquanto na digitalização setorial, o feixe é deslocado mediante uma faixa de varredura para uma profundidade focal específica, usando os mesmos elementos.

A Figura 5 ilustra o processo de excitação e aquisição de sinais, sendo que no processo de transmissão o instrumento envia um sinal de disparo (*trigger*) para o PA, que por sua vez, converterá o sinal em um pulso com largura e atraso de tempo predefinidos. Como cada elemento recebe um pulso, um feixe com ângulo e focalização específica será formado. No processo de recepção, por sua vez, os sinais são recebidos e deslocados no tempo, sendo então reunidos (ou somados) para formar um único pulso de ultrassom, transmitido à unidade de aquisição, para posteriormente ser usado para visualização da imagem.

Figura 5 – Sistemas de excitação e aquisição de sinais.



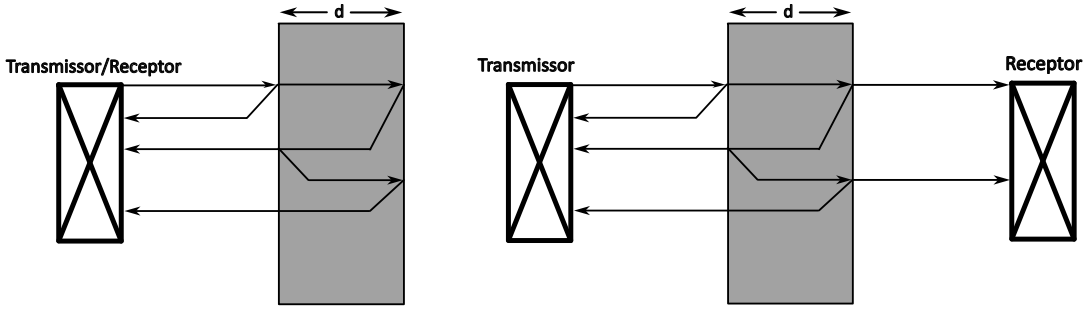
Fonte: Adaptado de Olympus (2004).

2.2.2 Método de medição por onda pulsada

O método de medição por onda pulsada pode ser utilizado com um ou dois transdutores. A Figura 6 ilustra o arranjo feito durante a medição. No primeiro cenário, o transdutor opera no modo pulso-eco, ou seja, o mesmo elemento é responsável por emitir e receber os sinais acústicos. Nessa configuração, existem desafios que incluem problemas decorrentes do pulso de excitação, que pode resultar em um fenômeno conhecido como tempo morto e o aumento da atenuação do sinal, uma vez que a onda deverá percorrer o dobro da distância de propagação.

Figura 6 – Medição em pulso-eco e transmissão-recepção.

(a) Medição em pulso-eco. (b) Medição em transmissão/recepção.



Fonte: Adaptado de Adamowski et al. (2004).

No método transmissão, utiliza-se um transdutor como transmissor e um segundo transdutor como receptor. Como desvantagem, pode-se citar o custo elevado dos transdutores, as dificuldades de alinhamento entre transdutor e receptor, particularmente em situações que envolvem sistemas com altas frequências, além de possíveis diferenças construtivas associadas aos transdutores. Além disso, pode ser visto experimentalmente que, certos materiais operam de maneira mais eficiente como transmissores, enquanto outros são mais eficazes como receptores.

2.2.3 Modo de funcionamento

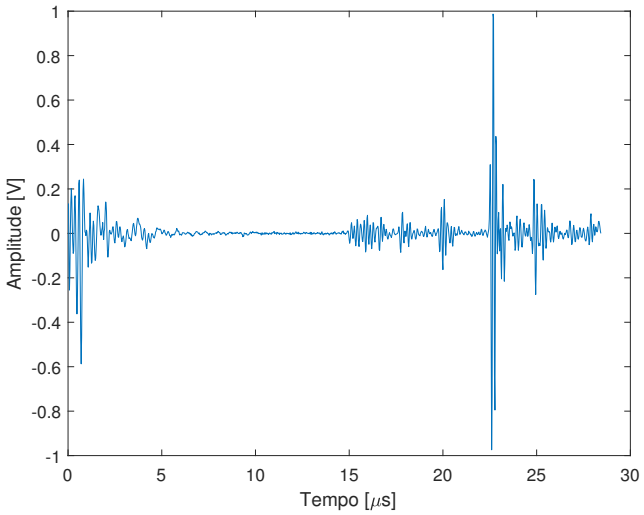
Em aplicações ultrassônicas, existem alguns modos de funcionamento, ou modos de representação gráfica, dentre as quais destacam-se: *A-scan*, *B-scan*, *C-scan*, *D-scan*, e *S-scan*. Os modos *A-scan* e *B-scan*, serão descritos com mais detalhes nas próximas seções. A depender do modo utilizado e da aplicação, uma abordagem pode se tornar mais útil para avaliar a amplitude dos ecos e identificar interfaces entre diferentes materiais, podendo ser utilizada para medir distâncias e detectar variações na amplitude que indicam irregularidades. Da mesma forma, certas abordagens podem oferecer uma visão mais completa e detalhada do objeto a ser analisado, auxiliando na identificação de características específicas.

2.2.3.1 Modo *A-scan*

No modo *A-scan*, a exibição é feita com a amplitude do sinal ultrassônico (eixo vertical) em função do tempo (eixo horizontal). Em sua representação, picos de amplitude podem indicar a presença de interfaces entre diferentes materiais, como uma fronteira entre camadas em um material sólido. O tempo decorrido entre a transmissão do pulso e a recepção do eco é proporcional à profundidade de penetração, o que possibilita o cálculo da distância à interface. A Figura 7 ilustra um exemplo de representação no modo

A-scan, onde no eixo horizontal é representado o tempo (obtido a partir da frequência de amostragem e do número de pontos), e no eixo vertical a amplitude do sinal. Essa Figura foi obtida a partir da matriz de dados gerada no ensaio utilizado para a metodologia deste trabalho.

Figura 7 – Representação *A-scan*.



Fonte: Do autor.

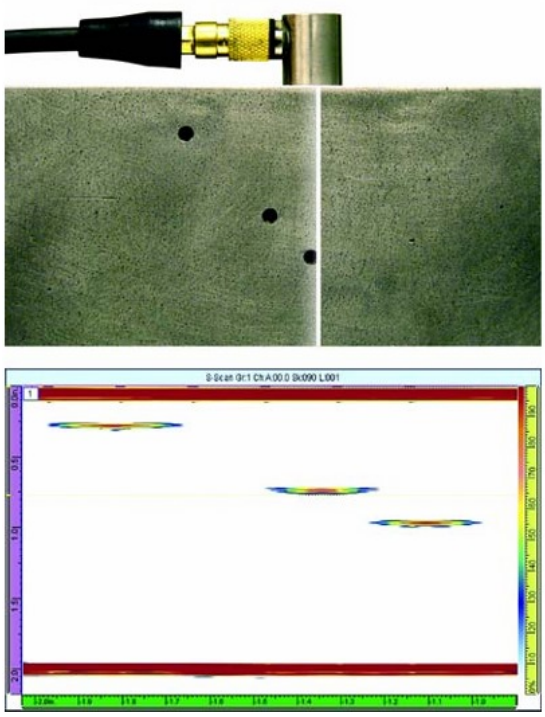
2.2.3.2 Modo *B-scan*

O modo *B-scan*, por apresentar visualizações de estruturas internas com maiores detalhes do que o modo *A-scan*, pode ser utilizado em ultrassonografia médica e em aplicações de NDE. Seu funcionamento é baseado na geração e transmissão de uma série de pulsos ultrassônicos em diferentes direções, com os ecos resultantes sendo registrados. A posição e intensidade desses ecos são mapeadas para criar uma imagem bidimensional, que representa a anatomia ou a estrutura interna do objeto.

Como apresentado na Figura 8, no eixo vertical da imagem é representada a profundidade da amostra, com cada ponto ao longo desse eixo correspondendo a uma profundidade específica (simbolizada pela mudança do brilho), permitindo visualizar diferentes camadas ou estruturas internas. No eixo horizontal, por sua vez, é representada a posição lateral da amostra. Ao percorrer o eixo horizontal, é possível visualizar diferentes regiões do objeto em uma seção transversal.

O *B-scan* também pode ser usado em conjunto com outros modos de exibição, como o *A-scan*, possibilitando obter informações mais abrangentes do objeto. Quando se utiliza *B-scan*, é importante e necessária a utilização de métodos de varredura que permitam obter imagens em tempo real, como a varredura eletrônica (amplamente usada).

Figura 8 – Representação *B-scan*.



Fonte: Retirado de Olympus (2010).

2.2.4 *Phased Arrays*

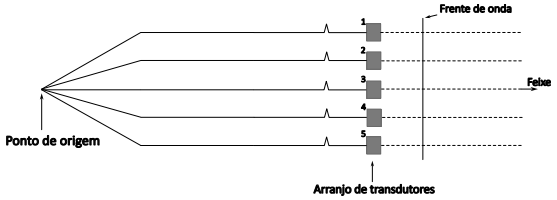
Phased array é um equipamento transdutor composto por elementos piezoelétricos controláveis eletronicamente. Em geral, esses dispositivos apresentam de 16 a 256 elementos individuais, que podem ser pulsados separadamente. Amplamente utilizado em NDE, especialmente em inspeções volumétricas e detecção de descontinuidades, o PA oferece flexibilidade e controle eletrônico significativos na direção e foco do feixe.

A Figura 9 apresenta o mecanismo de funcionamento de um arranjo de 5 elementos transdutores em um PA linear para realizar a varredura eletrônica, sendo que o direcionamento do feixe é obtido por meio de atrasos na excitação dos transdutores.

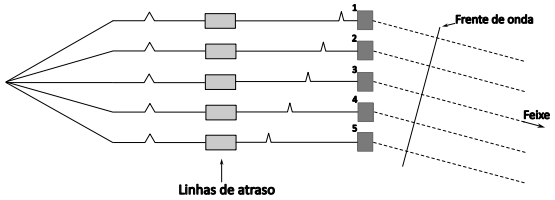
Em um PA, os elementos são acionados paralelamente, e cada um deles pode ser configurado para transmitir o sinal em momentos ligeiramente diferentes, resultando em adição ou cancelamento da energia dos sinais. Essa ação possibilita orientar e direcionar o feixe acústico da forma que se deseja, conforme mostrado na Figura 9. O direcionamento de forma dinâmica mediante vários ângulos e pontos focais torna viável a avaliação do material de teste em uma variedade de perspectivas diferentes.

Figura 9 – Varredura e direcionamento de feixe.

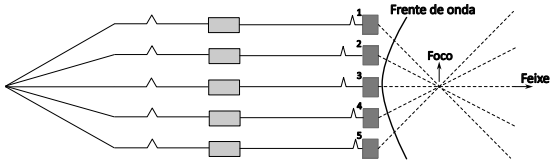
(a) Direcionamento do feixe paralelo.



(b) Direcionamento do feixe inclinado.



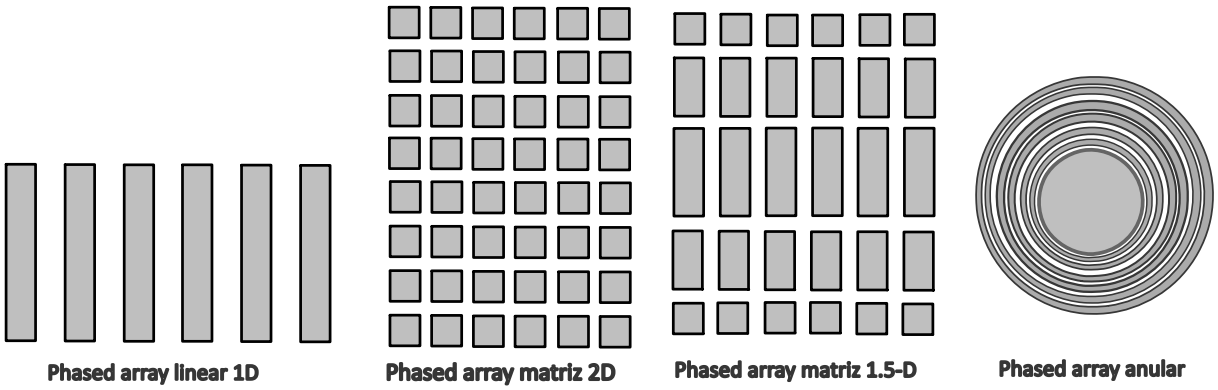
(c) Direcionamento do feixe focalizado.



Fonte: Adaptado de Costa et al. (2004).

Diferentes formatos de PA alteram a direção do feixe e a complexidade da implementação, impactando diretamente na avaliação acústica. A Figura 10 ilustra os formatos mais utilizados para aplicações industriais.

Figura 10 – Diferentes tipos de *phased array* para aplicações industriais.



Fonte: Adaptado de Olympus (2004).

Nos PAs, os transdutores podem ser organizados em formato linear, em matriz (2D, anular, circular) ou até mesmo em uma forma mais complexa, como mostrado na Figura 10. Assim como acontece com os transdutores convencionais, o PA pode ser projetado para uso com contato direto, operando em frequências na faixa de 2 MHz a 10 MHz (atualmente existem equipamentos que operam entre 25 MHz e 100 MHz). A grande vantagem do sistema PA consiste na possibilidade de varredura ou focalização por meio de uma faixa de ângulos diferentes, ou por meio de focalização dinâmica em diversas profundidades, aumentando assim a flexibilidade e capacidade em configurações de inspeção, além de possibilitar maior alcance na propagação da onda, tendo em vista que mais elementos emitem.

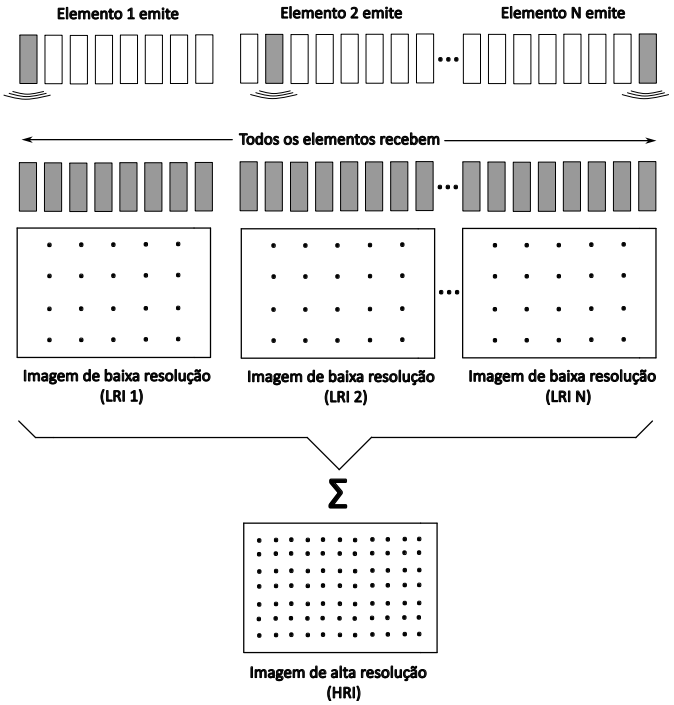
Dos modelos representados na Figura 10, o mais simples é o PA linear 1D, devido

a suas vantagens como fácil *design*, fácil produção, programação e simulação mais simples, baixo custo, e versatilidade. A disposição dos elementos do transdutor altera o formato do feixe (tendo em vista que a interação influencia diretamente a forma, a direção e a focalização do feixe acústico), de modo que um modelo pode ser mais ou menos eficiente do que outro, dependendo do objeto a ser analisado. Os modelos de PA linear 1D e matriz 2D apresentam formato de feixe elíptico. No modelo anular, por sua vez, o formato do feixe é esférico, enquanto o formato de PA matriz 1.5-D, existe a possibilidade de trabalhar tanto com feixes no formato elíptico, quanto no formato esférico.

2.2.5 *Synthetic aperture*

O princípio básico de funcionamento das técnicas para geração de imagens ultrassônicas por abertura sintética (SA) é mostrado na Figura 11. Um único elemento do transdutor é usado para transmitir uma onda acústica, cobrindo toda a região da imagem. Os sinais recebidos por todos os elementos na abertura são amostrados para cada transmissão. Esses dados podem ser usados para gerar uma imagem de baixa resolução. A focalização na transmissão é, portanto, sintetizada pela combinação das imagens de baixa resolução (LRIs) obtidas a partir da emissão feita por um elemento (de forma sequencial), e a recepção feita por todos. O resultado é uma imagem com melhor resolução lateral (HRI). A transmissão/recepção por mais elementos do transdutor permite um aumento na resolução lateral, pois a focalização da peça é completa.

Figura 11 – Princípio básico da imagem por abertura sintética (SA).



Fonte: Adaptado de Jensen et al. (2006).

2.3 Algoritmos para formação de imagens acústicas

Nesta seção serão abordados modelos de algoritmos para visualização de imagens do objeto a ser analisado. As técnicas aqui apresentadas dependem diretamente do conteúdo apresentado anteriormente, como *arrays*, sistemas de excitação e aquisição, e entre outros. Durante o processo de formação de imagens, a aquisição dos sinais representa um ponto crítico do processo, tendo em vista que existirá um grande volume de informações, além de utilização de técnicas de filtragem. É importante destacar que a aquisição não afeta o tempo de processamento da imagem, pois, nessa fase, a aquisição se encontra concluída.

2.3.1 SAFT

Técnicas de focalização por abertura sintética SAFT foram desenvolvidas para reduzir a complexidade eletrônica em comparação ao PA. Este objetivo é alcançado ao sintetizar uma abertura (elementos do transdutor emitindo ou recebendo) maior a partir da transmissão por mais elementos do *array*, combinando informações de várias posições da peça analisada.

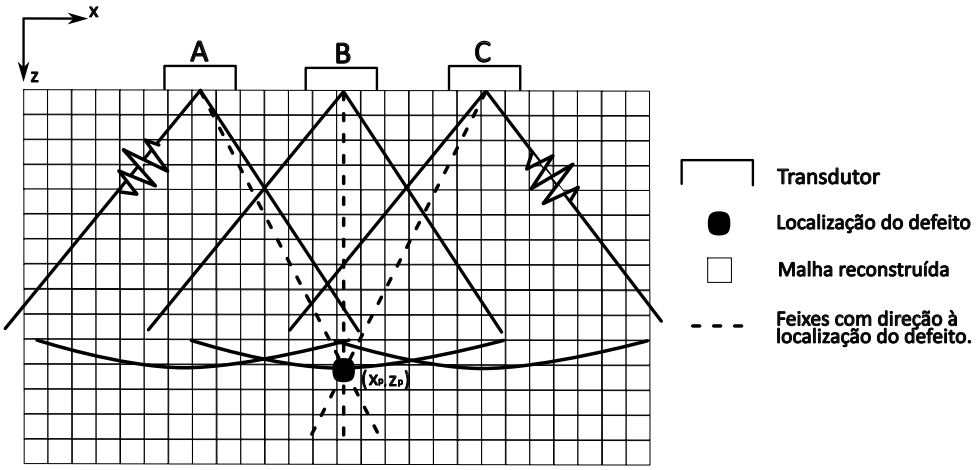
No domínio do tempo, a técnica é feita por meio de um processo de reconstrução para sobrepor (ou somar) coerentemente vários dados (ondas acústicas refletidas) no formato *A-scan* em todos os locais da região de interesse, envolvendo amostragem, mapeamento e soma. Mapeamento refere-se à colocação de cada um dos pontos adquiridos em uma grade tridimensional que representa o modelo geométrico do objeto a ser inspecionado, enquanto o processo de soma é feito a partir da combinação coerente de vários dados adquiridos.

O princípio de funcionamento do modelo SAFT pode ser representado de forma gráfica pela Figura 12, onde os elementos do *array* são mantidos fixos ao longo da superfície de varredura. Em cada uma das posições de aquisições de dados, os elementos do transdutor são configurados para emitir e receber sinais do tipo *A-scan*.

Deve-se ressaltar que os sinais adquiridos por um elemento do *array* em uma posição podem se sobrepor aos sinais adquiridos por outros elementos em outras posições, sendo um desafio a ser tratado. Na Figura 12, isso pode ser visto pela sobreposição dos sinais de eco (linhas tracejadas) do transdutor na posição A com os sinais produzidos pelos elementos nas posições B e C.

Cada um desses locais de sobreposição pode ser visto como um elemento de abertura. A ideia básica da reconstrução SAFT é processar os elementos de abertura como uma unidade, somando todos os elementos individuais.

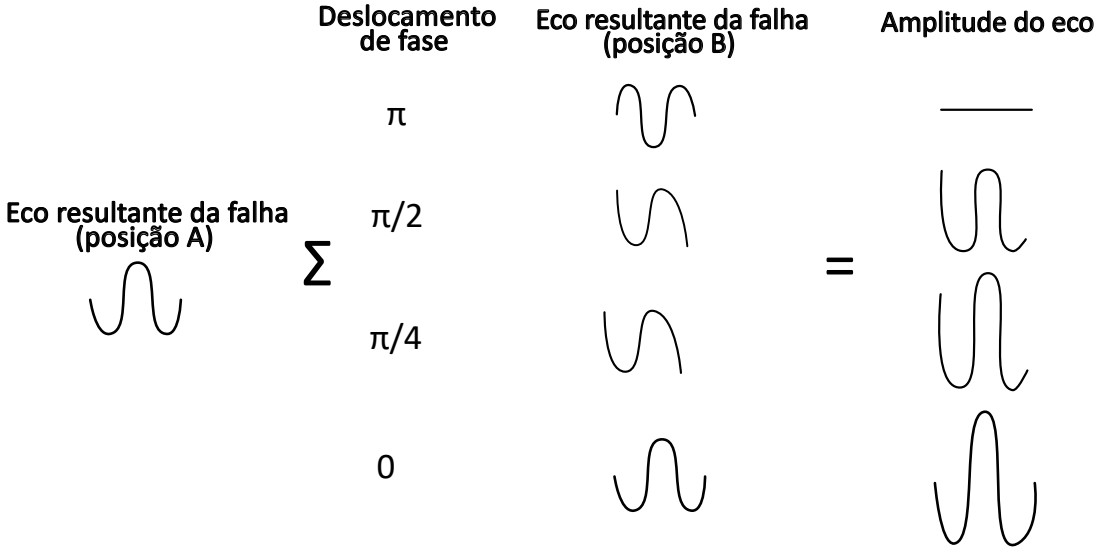
Figura 12 – Princípio básico do método SAFT.



Fonte: Adaptado de Guan, He e Rasselkorde (2015).

Outro detalhe que pode ser destacado ao utilizar a abordagem SAFT, é que se a abertura estiver centralizada no alvo (ou defeito), a soma de cada elemento sobreposto resultará em uma amplitude de eco maior, enquanto para o caso de uma localização fora do alvo a soma produzirá uma amplitude menor, como apresentado na Figura 13.

Figura 13 – Amplitude do eco resultante do método SAFT.



Fonte: Adaptado de Guan, He e Rasselkorde (2015).

Cada ponto (ou *pixel*) da imagem gerado por SAFT tem uma intensidade (representada por meio de cores) calculada por meio do tempo de voo (TOF, do inglês *Time of Flight*), que representa o tempo necessário para a onda se propagar pelo meio, entre o transdutor e o ponto focalizado, considerando a ida e a volta. Assumindo que o transdutor apresenta um comportamento omnidirecional, pode-se considerar que o pulso emitido por um transdutor em determinada posição (x_i,z_i) irá se propagar duas vezes pelo meio (ida e volta) até atingir o ponto (x,z). O tempo necessário para que a onda se propague de

um ponto a outro pode ser calculado por

$$t_{ip} = \frac{2}{c}\sqrt{(x_i - x_p)^2 + (z_i - z_p)^2}, \tag{2.5}$$

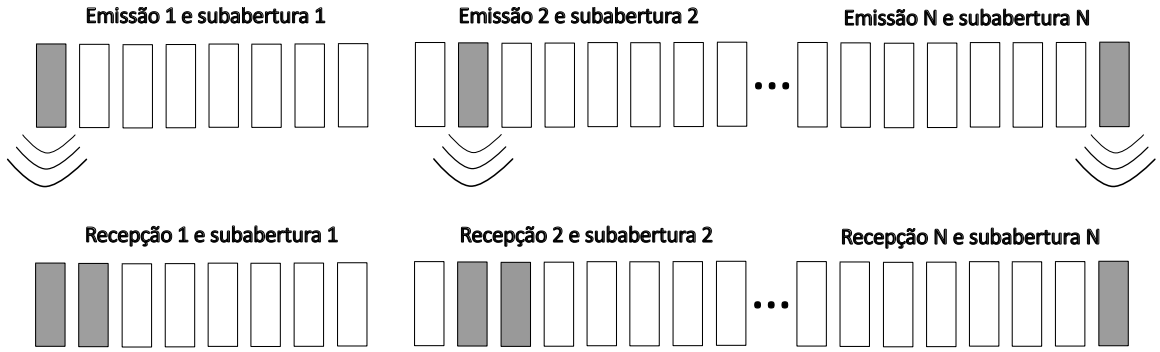
sendo t_{ip} o tempo $[s]$, c a velocidade $[m/s]$, e x_i, x_p, z_i, e z_p a posição $[m]$.

2.3.2 2R-SAFT

O método 2R-SAFT tem o mesmo princípio básico de funcionamento visto em SAFT, com a síntese da abertura a partir da soma coerente dos dados da transmissão/recepção de cada elemento, diminuindo a complexidade eletrônica. A principal característica que distingue o método 2R-SAFT é que há um elemento como transmissor, e 2 elementos como receptores.

A Figura 14 demonstra o processo citado. Percebe-se que, dada a configuração, são recebidos $2N - 1$ sinais, o que aumenta a relação sinal-ruído comparada à abordagem convencional SAFT. A principal desvantagem do 2R-SAFT é devido ao número total de sinais $(2N - 1)$, que é o dobro do número total de sinais armazenados no SAFT convencional (N). Sendo assim, em comparação com SAFT, o 2R-SAFT requer o dobro da memória para armazenar os sinais. Além disso, o tempo de processamento durante a emissão e recepção são maiores.

Figura 14 – Método de focalização com 2 receptores, 2R-SAFT.



Fonte: Adaptado de Romero et al. (2009).

2.3.3 SAFT-TFM

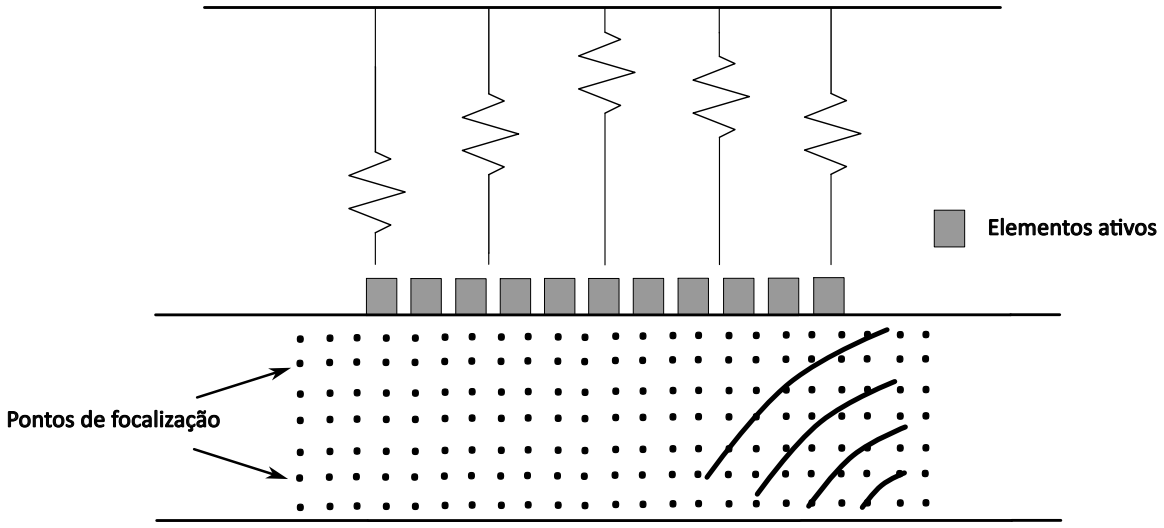
O método SAFT-TFM também representa uma abordagem derivada da técnica SAFT. O que diferencia o SAFT-TFM é a utilização de todos os elementos do *array* tanto para transmitir, quanto para receber. A abordagem leva em consideração a propagação do feixe ultrassônico em todas as direções, permitindo uma focalização mais eficaz.

O processo simultâneo apresenta vantagens durante avaliações de estruturas complexas e espessas, ao permitir melhor focalização comparada aos modelos SAFT e 2R-SAFT.

Para alcançar estes resultados, todos os elementos do transdutor emitem e todos recebem, enquanto em SAFT, há apenas um emissor e receptor (geralmente o mesmo elemento).

A Figura 15 ilustra o processo realizado em SAFT-TFM, onde todos os elementos do transdutor estão ativos.

Figura 15 – Método de focalização total, SAFT-TFM.



Fonte: Adaptado de Holmes, Drinkwater e Wilcox (2005).

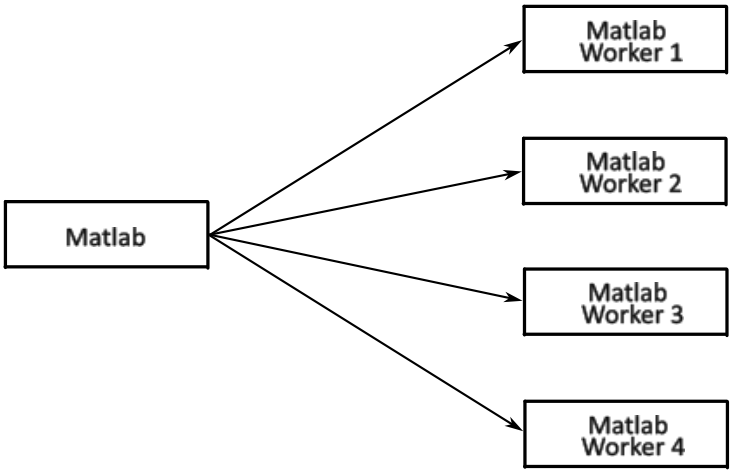
2.4 Estratégias para redução do tempo de execução

Esta seção tratará alguns tópicos importantes das abordagens utilizadas para tornar a geração de imagens acústicas mais eficiente, reduzindo o tempo de execução da implementação.

2.4.1 Paralelização no Matlab com a *Parallel Computing Toolbox*

Para a proposta deste trabalho, a implementação que se mostrou mais eficiente em Matlab foi com a utilização da *Parallel Computing Toolbox*. Essa ferramenta permite que o próprio Matlab divida tarefas entre os *cores* disponíveis no processador, visando diminuir o tempo de execução, como mostrado na Figura 16. É importante destacar que a terminologia utilizada pelo Matlab define *worker* como sendo uma instância independente que roda em segundo plano, separada da sessão cliente do Matlab (sessão que o usuário interage diretamente). Segundo a MathWorks (2024e), *workers* executam tarefas de forma simultânea, permitindo o processamento paralelo.

Figura 16 – Divisão de tarefas no Matlab.



Fonte: Adaptado de MathWorks (2024b).

Neste Capítulo, a proposta é mencionar cenários, conforme é apresentado pela MathWorks (2024c), em que a instrução de paralelização de laço *parfor* não apresenta resultados melhores, ou que até mesmo não pode ser utilizado. O conhecimento destes detalhes faz toda diferença na hora de resolver um problema usando esta abordagem, que voltará a ser tratada no Capítulo 3, com a implementação realizada.

2.4.1.1 Paralelização de laços aninhados

O Algoritmo 1 apresenta quatro laços *for* aninhados, sendo uma estrutura semelhante aos algoritmos que serão tratados posteriormente.

Algoritmo 1 – Exemplo com 4 laços *for* aninhados

```
for i = 1 to M do
    for j = 1 to N do
        for k = 1 to P do
            for l = 1 to Q do
                // Executa um processamento
            end for
        end for
    end for
end for
```

A princípio, poderia substituir qualquer laço *for* por um *parfor*, sem a apresentação de erros durante a execução. Entretanto, a boa prática é paralelizar apenas o laço mais externo, pois se trata de uma estratégia que evita sobrecarga de trabalho, que aconteceria caso fossem paralelizados laços internos.

Um laço interno concorrente certamente elevaria o tempo de execução do algoritmo, tendo em vista que os *workers* seriam inicializados várias vezes, e possivelmente haveria um mau gerenciamento da memória, pois cada variável do conjunto de instruções

paralelizadas seria copiada para uma instância *worker*. Portanto, o Algoritmo 2 apresenta a melhor escolha de paralelização para laços aninhados.

Algoritmo 2 – Boa prática: laço *parfor* externo a laços *for*

```
parfor i = 1 to M do
    for j = 1 to N do
      for k = 1 to P do
        for l = 1 to Q do
          //Executa um processamento
        end for
      end for
    end for
end parfor
```

2.4.1.2 Limites de um laço *parfor*

Na implementação concorrente do Matlab, a instrução *parfor* não permite a definição do limite superior de laços internos a partir de uma função, pois a princípio esse valor poderia ser modificado, o que impactaria na indexação de matrizes dentro do laço paralelizado. O Algoritmo 3 apresenta este problema, com a função *size* definindo o limite superior.

Algoritmo 3 – Inválido: limite definido por uma chamada de função

```
A = zeros(100, 200);
parfor i = 1 : size(A,1) do
    for j = 1 : size(A,2) do
      A(i, j) = i + j;
    end for
end parfor
```

O Algoritmo 4, por sua vez, apresenta uma possível solução com a utilização de uma variável escalar n para o limite superior do laço interno.

Algoritmo 4 – Válido: limite definido por uma variável constante

```
A = zeros(100, 200);
n = size(A, 2);
parfor i = 1 : size(A,1) do
    for j = 1 : n do
      A(i, j) = i + j;
    end for
end parfor
```

Para as implementações aqui apresentadas, este caso é importante nas estratégias com a redução de cálculos redundantes, pois o laço dos pontos em x é diminuído pela metade.

2.4.1.3 Operação com o índice de um laço interno ao *parfor*

Este cenário se aplica nos casos em que a variável do laço é alterada em qualquer outra parte da implementação além da instrução do *loop for*. Destaca-se que não é garantido que cada *worker* tenha disponível a região indexada. O Algoritmo 5 é inválido, pois tenta modificar o valor da variável de índice do *loop for* dentro do próprio laço.

Algoritmo 5 – Inválido: operação com o índice de um *loop* interno

```
A = zeros(10);
parfor i = 1 : 10 do
    for j = 1 : 10 do
        A(i, j) = 1;
        j = j + 1;
    end for
end parfor
```

Por outro lado, o Algoritmo 6 apresenta uma possível solução, com a utilização de uma variável temporária t. Essa prática, quando exigida, garante que cada instância *worker* tenha acesso à matriz indexada (cópia para cada instância), pois seu índice não está sendo alterado com uma operação.

Algoritmo 6 – Válido: utilização de uma variável temporária

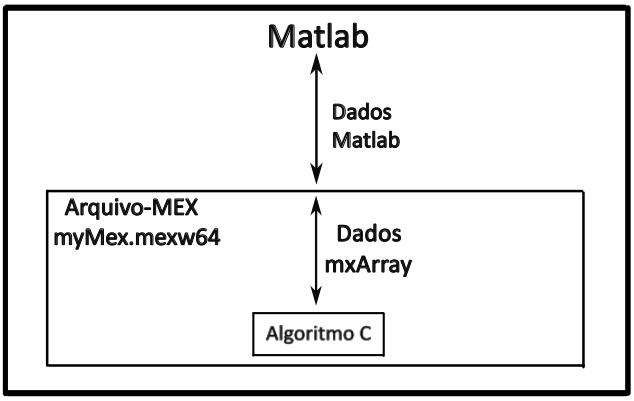
```
A = zeros(10);
parfor i = 1 : 10 do
    for j = 1 : 10 do
        A(i, j) = 1;
        t = j;
        t = t + 1;
    end for
end parfor
```

2.4.2 Matlab *Executable* (MEX)

De acordo com a MathWorks (2024a), é possível implementar algoritmos em $C/C++$ diretamente no Matlab, aproveitando as ferramentas gráficas que o *software* oferece. Na Figura 17 é ilustrado o princípio básico de seu funcionamento, onde o arquivo MEX utiliza os dados do Matlab como entrada para o algoritmo em C, utilizando para este objetivo uma abordagem com ponteiros. Este método será utilizado neste trabalho como estratégia de programação dos algoritmos em linguagem C. A justificativa desta escolha se baseia no formato da matriz de dados disponível, assunto que será melhor contemplado no Capítulo 3.

Para executar este tipo de arquivo é necessário primeiramente verificar se o compilador desejado está instalado. Para o caso aqui abordado, utilizou-se o compilador *MinGW-w64* para sistema operacional *Windows*.

Figura 17 – Funções MEX.



Fonte: Adaptado de Olah (2019).

Estando instalado, para configurar o compilador foi executado o comando '*mex -setup*' na janela de comandos do Matlab e selecionado o *MinGW-w64*. O algoritmo implementado é salvo na extensão desejada (*.c* por exemplo) e compilado na janela de comandos com a instrução '*mex mymex.c*'. Sem a presença de erros, esta sequência de passos gera um arquivo do tipo *mexw64*.

Deve-se destacar que em funções MEX existem algumas características intrínsecas, como o formato da função principal, com seus argumentos de entrada e saída seguindo o seguinte apresentado no Algoritmo 7.

Algoritmo 7 – Função MEX

```
void mexFunction(int nlhs, mxArray *plhs[ ], int nrhs, const mxArray *prhs[ ]) {
    // O código é feito aqui
}
```

A diferenciação dos argumentos de entrada e saída é feita pela segunda letra da expressão que define os ponteiros: l para saída e r para entrada. Sua utilização pode ser feita com gerenciamento de memória, e manipulação de dados em matrizes ou *arrays*, utilizando os argumentos plhs[ ] e prhs[ ]. Em contrapartida, a utilização de funções que retornam o valor diretamente se dá em grandezas escalares que indicam tamanho ou número de elementos (nlhs e nrhs), por exemplo.

Nessa configuração, conforme é descrito pela MathWorks (2024a), é possível chamar funções escritas em outras linguagens a partir de um código em Matlab. Essa vantagem, como será melhor evidenciada nos Capítulo 3 e 4, se dá principalmente pelo acesso à memória RAM, tendo em vista que em MEX a manipulação dos ponteiros de memória é direta, permitindo operações (tanto de leitura quanto de escrita) mais rápidas e efici-

entes sobre os dados. Neste trabalho, funções MEX foram utilizadas para implementar algoritmos de focalização por SA de forma sequencial e concorrente em linguagem *C*.

2.4.3 Processadores *multi-core*

Esta seção visa apresentar assuntos relacionados a processadores, acesso à memória, programação sequencial e paralela, bem como técnicas de implementação concorrente. Atualmente, este é um assunto bem consolidado no ambiente acadêmico, o que é justificado com o avanço substancial dos componentes eletrônicos, e consolidação dos computadores em diferentes aplicações. Como apresentado por George e Hawkes (1992), a maior acessibilidade tecnológica possibilitou gradativa incorporação de mais *cores* e memória nos *hardwares*, viabilizando a substituição da computação sequencial (ou centralizada) pela computação paralela (ou distribuída). A principal vantagem dessa substituição é a redução nos tempos de execução, dado o melhor aproveitamento computacional. O desafio muitas vezes fica por conta da maior dificuldade em programar de forma concorrente, comparada à forma centralizada.

2.4.3.1 Núcleos de processamento

A terminologia utilizada para caracterizar as funções de um processador define *cores* como unidades de processamento independente em uma CPU. Processadores modernos podem ter múltiplos núcleos, permitindo a execução de várias tarefas simultaneamente. Cada núcleo é capaz de executar instruções de forma independente, melhorando o desempenho e a capacidade de lidar com multitarefas. Em computação paralela, os modelos de arquitetura mais comuns são: múltiplos dados de múltiplas instruções (MIMD, do inglês *Multiple Instruction Multiple Data*), múltiplos dados de instrução única (SIMD, do inglês *Single Instruction Multiple Data*), e dados únicos de múltiplas instruções (MISD, do inglês *Multiple Instruction Single Data*).

Conforme explicado por George e Hawkes (1992), no modelo SIMD uma única instrução é executada simultaneamente por múltiplos processadores ou elementos de processamento, sendo que cada um executa a mesma tarefa, mas opera em diferentes conjuntos de dados. É eficaz para os casos em que a mesma operação é aplicada a conjuntos grandes de dados simultaneamente, como em operações vetoriais. GPUs são muitas vezes projetadas a partir do modelo SIMD. Na abordagem MIMD, o princípio é o oposto, pois múltiplas unidades de processamento executam instruções independentes de forma simultânea. Cada elemento pode executar um programa diferente, operando em diferentes conjuntos de dados. Essa abordagem é mais flexível e versátil, sendo adequada para uma ampla variedade de tarefas paralelas. É muito aplicado em sistemas de computadores distribuídos, como *clusters*.

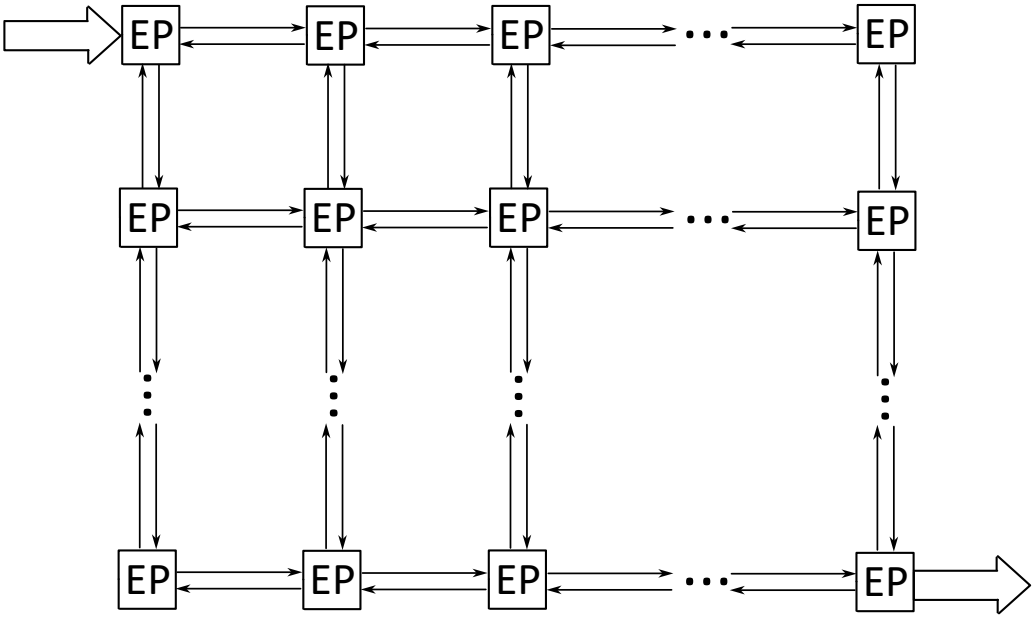
Na abordagem MIMD, por exemplo, a configuração linear permite que os dados sejam transmitidos de uma célula para sua vizinha, sendo que a comunicação é bidirecional, conforme mostrado na Figura 18, onde cada elemento de processamento é representado pelo bloco EP.

Figura 18 – Arranjo linear de elementos de processadores.

Fonte: Adaptado de George e Hawkes (1992).

Por último, o modelo MISD envolve múltiplas unidades de processamento que operam em um único conjunto de dados, mas cada uma executa uma instrução diferente. Pode ser empregado em casos onde é necessário verificar diferentes parâmetros de um mesmo conjunto de dados. Entretanto, sua utilização é menor, comparada aos outros modelos. Válido mencionar também que, existem diferentes arranjos de elementos de processamento, sendo o arranjo linear e matricial os mais comuns. A Figura 19 apresenta a configuração matricial.

Figura 19 – Arranjo matricial de elementos de processadores.



Fonte: Adaptado de George e Hawkes (1992).

O arranjo apresentado acima possui mais elementos de processamento comparado ao método linear, e existem duas possibilidades de implementação para o método matricial. Uma é com a utilização de uma referência de *clock*, o que é um grande problema para matrizes de ordem elevada, devido a diferenças de tempo. Por outro lado, a outra possibilidade opera de forma assíncrona com o protocolo *handshake*, usado em muitos processos de comunicação. Como desvantagens, pode-se destacar o tempo de atraso gerado pelo protocolo.

2.4.3.2 *Threads*

Define-se *thread* como uma unidade em termos de *software*, referindo-se a uma sequência de instruções que podem ser executadas de forma independente. Uma *thread*, portanto, simboliza uma unidade básica de execução de um programa. Assim como os *cores*, *threads* possibilitam que um programa seja executado em multitarefa, ou *multithreading*, permitindo que várias operações ocorram simultaneamente.

Com relação à implementação, *threads* podem ser configuradas no nível de núcleo (também chamado de nível *Kernel*) ou do usuário. No nível de núcleo, os *threads* são gerenciados diretamente pelo sistema operacional. Por outro aspecto, nos *threads* de usuário, como o próprio nome indica, o gerenciamento é feito por uma biblioteca presente na camada de usuário. Outra característica importante desses dois modos, é que no modo *Kernel* é possível ter acesso a todas as partes do *hardware*, enquanto no modo usuário as ações são limitadas.

2.4.4 Paralelização em *C* com a biblioteca *windows.h*

Desenvolvida pela *Microsoft*, a biblioteca *windows.h* foi feita para algoritmos *C/C++*, principalmente para as rotinas em que há maior interação com o sistema operacional *Windows*. Neste trabalho, essa biblioteca se aplica no algoritmo com tarefas paralelizadas, pois ela facilita o trabalho com *cores* e *threads*. Por facilitar a implementação, o algoritmo fica mais limitado ao que a biblioteca disponibiliza. Sendo assim, as *threads* serão configuradas no nível de usuário, e não de núcleo.

Dessa forma, a biblioteca possibilita a utilização de algumas funções para implementação de algoritmos concorrentes, bem como identificadores para as *threads* (*handle*) que possibilitam sua inicialização, suspensão, ou obtenção de *status*. Dentre as funções que existem, pode-se destacar (utilizadas no algoritmo implementado):

- ***beginthreadex***: parte da biblioteca padrão em *C* CRT, essa função permite criar *threads* em algoritmos que também fazem uso desses recursos;
- ***WaitForMultipleObjects***: função usada para aguardar até que um ou mais objetos (*threads* nesse caso) estejam em estado sinalizado (indicação de operação ou condição pelo qual o objeto está sendo aguardado foi satisfeita);
- ***CloseHandle***: fecha um *handle* aberto para liberar recursos;
- ***endthreadex***: usada para finalizar uma *thread*.

As funções citadas acima facilitam a implementação do algoritmo no método concorrente, porém apresenta limitações. A respeito dessas restrições impostas pela biblioteca, destaca-se a impossibilidade de utilizar o *multithreading*, e gerenciar o acesso dos *cores* à

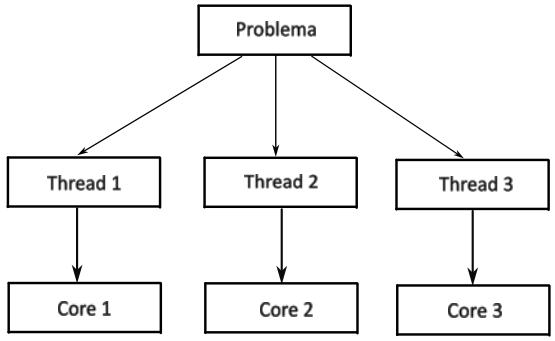
memória *cache* e RAM. Tais características serão exploradas no próximo Capítulo, onde os detalhes que envolvem a implementação concorrente dos algoritmos de focalização serão apresentados.

2.4.5 Terminologia utilizada para *cores*, *threads*, e *workers*

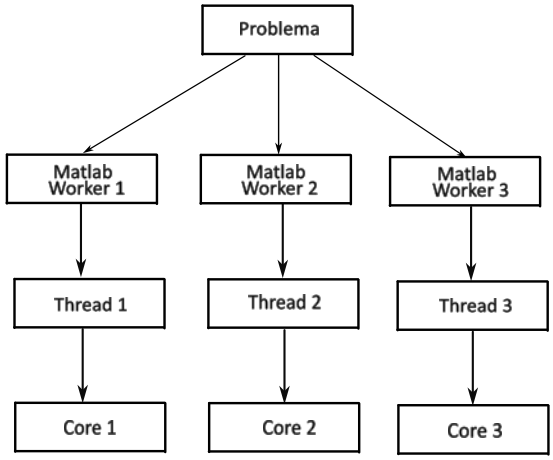
A Figura 20 apresenta e compara o modo como esses conceitos são aplicados em *C* e Matlab, onde há a associação de apenas um *thread* e um *worker* por núcleo físico.

Figura 20 – Diferença entre *core*, *worker*, e *thread* na paralelização de tarefas.

(a) Paralelização em *C*.



(b) Paralelização no Matlab.



Fonte: Adaptado de Bonannella (2022).

Como pôde ser visto ao longo deste Capítulo, foram abordados determinados conceitos relacionados à paralelização, como *cores*, *threads*, e *workers*. Ressalta-se que há uma diferença entre o significado de cada, e a distinção de cada terminologia é importante para o Capítulo 3, onde serão apresentadas as implementações realizadas.

A terminologia utilizada para as implementações no Matlab define *workers* como sessões independentes do *software* para processamento paralelo, que rodam em segundo plano. Aplicados exclusivamente ao Matlab, os *workers* também se associam com as *threads*, de modo que elas são utilizadas dentro de cada *worker* como um processo para execução paralela. Esses processos são executados de forma concorrente pelos núcleos de processamento da CPU.

Nas implementações em *C*/MEX, por sua vez, a definição de *threads* se refere como a menor unidade de processamento, ou um conjunto de instruções a serem executadas pelo processador. Elas podem ser gerenciadas por apenas um único núcleo, ou distribuídas em múltiplos núcleos para execução paralela.

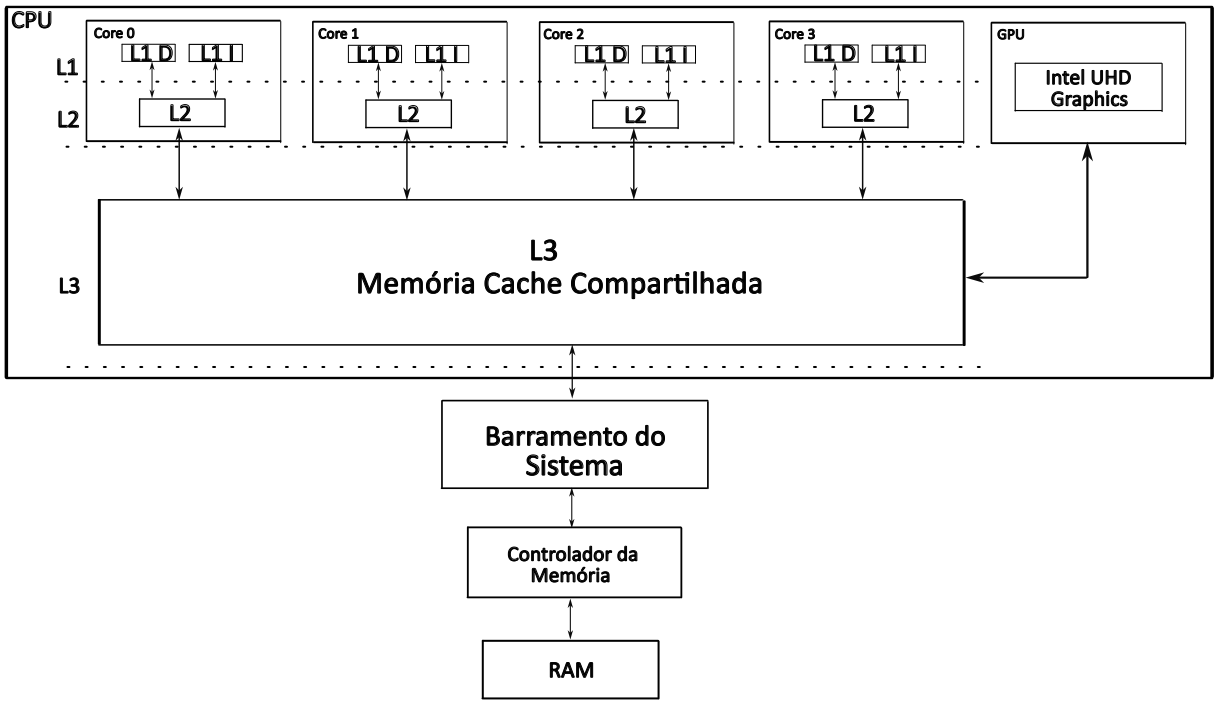
Em ambos os casos, para Matlab e *C*, *cores* são núcleos físicos do processador que permitem a execução de tarefas simultâneas. O número de *cores* disponíveis impacta

diretamente no número de *workers* e *threads*, tendo em vista que estas duas últimas se associam mais ao *software* do que ao *hardware* (associação dos *cores*).

2.5 Diagrama do barramento

O diagrama do barramento da CPU utilizada para os testes (Intel® Core™ i5-8250U) está representado pela Figura 21, em que se destaca o arranjo dos núcleos do processador com a memória *cache*, bem como o acesso à memória RAM, a partir do barramento do sistema (que engloba o barramento de dados, endereços, e controle) e o controlador da memória, componente responsável por gerenciar o fluxo de dados entre a CPU e a RAM.

Figura 21 – Diagrama do barramento.



Fonte: Adaptado de Intel (2024).

Pelo diagrama, percebe-se que existem três níveis de hierarquia para a memória *cache*, sendo elas L1, L2, e L3. Elas possuem tempo de acesso diferente, e capacidades de armazenamento também diferentes. A *cache* L1 é a mais rápida dentre as três, sendo que cada núcleo possui sua própria *cache* L1, além de ser dividida em duas partes, uma para armazenar dados (L1D), e outra para instruções (L1I). Por ser uma memória exclusiva, sua capacidade de armazenamento é a mais baixa dentre as três, com 256 KB no total.

A *cache* L2 também se encontra em cada núcleo da CPU, com tempo de acesso muito baixo, mas ainda superior comparado à L1. Outra diferença importante é o fato da L2 armazenar tanto dados quanto instruções, sem dividir a memória para esses conjun-

tos. Trata-se de uma memória com mais capacidade do que a L1, totalizando 1 MB de armazenamento.

Por outro lado, a L3 é o último nível da memória *cache*, com tempo de acesso superior comparado à L1 e L2, tendo em vista que ela é compartilhada entre os núcleos da CPU, e consequentemente a latência de acesso é maior. Dentre as três, a *cache* L3 possui maior capacidade de armazenamento, com 6 MB. Na Figura 21 ainda é representada a GPU integrada ao processador, Intel® UHD Graphics 620. A GPU não possui acesso às *caches* L1 e L2, e sim à L3. Ressalta-se que a GPU não foi utilizada para os testes que serão abordados nos próximos Capítulos, sendo esta uma sugestão para futuros trabalhos.

2.6 Considerações parciais

Neste Capítulo abordou-se a teoria por trás dos assuntos que serão novamente apresentados nos Capítulos seguintes, incluindo a descrição do ensaio, os algoritmos implementados, e os resultados obtidos. Foram descritas as principais informações (no que diz respeito a esse trabalho) envolvendo ondas acústicas, transdutores piezoelétricos, sistemas de excitação e aquisição, e a abstração envolvendo os algoritmos para formação de imagens que se pretende implementar.

O referencial teórico envolvendo PA, sistemas de excitação e aquisição, método de medição em pulso eco e o funcionamento dos algoritmos, seja sequencial ou paralelo, servirão como apoio para a descrição da matriz de pontos construída para geração das imagens, do ensaio realizado, e das implementações feitas.

3 Metodologia

Este Capítulo abordará essencialmente as características do ensaio (como o sistema de aquisição e matriz de pontos), as principais características da plataforma usada para testes, e as implementações feitas das técnicas para formação de imagens acústicas. A compreensão desses tópicos depende diretamente de alguns pontos abordados no Capítulo anterior de revisão teórica.

Assuntos como *phased arrays*, medição em pulso-eco, princípio de formação de imagem acústica (abordagens SAFT, SAFT-TFM, e 2R-SAFT), bem como as estratégias utilizadas para reduzir o tempo de execução, serão essenciais para os tópicos de estudo deste Capítulo que, por sua vez, servirá como base para o que será tratado posteriormente.

3.1 Descrição do ensaio

Para o ensaio ultrassônico, utilizou-se os dados fornecidos por Romero-Laorden et al. (2011). Foi utilizado um *array* linear com 128 elementos, e uma peça quadrada de alumínio com 100 mm em cada lado, contendo 9 pares de furos, cada qual com um diâmetro de 0,50 mm. A velocidade do som ao se propagar pela peça, é de 6300 m/s. A Figura 4 ajuda a compreender o modo como este tipo de ensaio é feito com um sistema de aquisição e um computador.

A Figura 22 ilustra o bloco de alumínio descrito, com os pares de defeitos destacados em vermelho. Ainda sobre o transdutor usado, o equipamento possui um espaçamento de 0,60 mm entre cada elemento, frequência do transdutor de 5 MHz, e frequência de amostragem de 40 MHz.

Figura 22 – Peça utilizada para o ensaio.

Fonte: Adaptado de Romero-Laorden et al. (2011).

Os sinais adquiridos (ondas acústicas refletidas pelo objeto de interesse) foram organizados em uma matriz tridimensional, de ordem 128x1137x128, onde a 1ª e 3ª dimensão referem-se a relação entre os elementos (emissor x receptor), considerando o modo de operação pulso-eco; e a segunda dimensão refere-se ao número de pontos adquiridos para cada combinação emissor-receptor utilizada, que depende diretamente da frequência de amostragem do sistema.

3.2 Descrição da plataforma utilizada para os testes

O computador utilizado para as implementações e testes aqui apresentados é o modelo Lenovo ideapad 330, com CPU Intel Core i5-8250U, 1,6 GHz, 4 núcleos e 8 *threads*. Possui memória RAM DDR 4 *dual channel* com capacidade de 8 GB e largura de banda de 2400 MHz.

A placa mãe do computador é do modelo Lenovo LNVNB 161216, com barramento PCI-E 3.0, e sistema operacional *Windows* 11 *Home Single Language*. O ambiente de desenvolvimento dos algoritmos foi o Matlab.

Importante mencionar que o sistema operacional *Windows* executa tarefas em segundo plano, que não são gerenciadas pelo usuário como, por exemplo, o *windows update*. Como essas tarefas podem influenciar nos tempos de processamento medidos, foi adotada a prática de inicializar previamente somente o *software* Matlab em primeiro plano, considerando o dispositivo em modo avião.

3.3 Implementações em Matlab

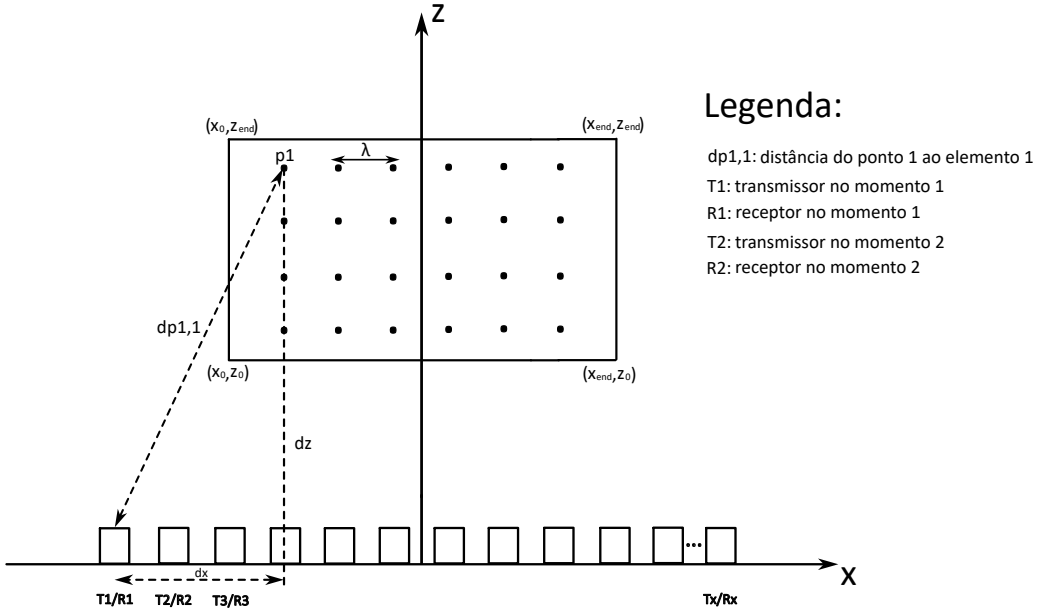
Esta seção descreve os procedimentos utilizados para implementações sequenciais e concorrentes dos algoritmos de focalização no Matlab. Foram utilizadas as técnicas de SA: SAFT, 2R-SAFT, e SAFT-TFM. Para cada caso é feito um pseudocódigo, e apresentado uma imagem ilustrativa dos transmissores e receptores para cada abordagem, permitindo a visualização das principais diferenças. É discutida também a ideia por trás da definição do tamanho e construção da matriz de pontos, além da influência da dimensão dessa estrutura no tempo de execução em cada abordagem. Para as estratégias otimizada e concorrente, ressalta-se que a implementação foi feita somente para a técnica SAFT-TFM, por se tratar do pior caso.

3.3.1 Implementações sequenciais

3.3.1.1 Algoritmo SAFT

Inicialmente, é importante compreender como são calculados os pontos para a formação da imagem da peça a ser analisada, tendo em vista que este processo é feito nas três abordagens. Para isso, utilizam-se algumas informações do experimento, como a frequência de amostragem, velocidade e, consequentemente, comprimento de onda. Um critério muito utilizado para resolução é do comprimento de onda λ, sendo calculado a partir da divisão entre a velocidade da onda e a frequência de amostragem. A Figura 23 ilustra a dinâmica transmissor/receptor nas medições pulso-eco da abordagem SAFT, juntamente com a representação dos procedimentos para os cálculos das distâncias dos elementos para cada ponto definido, onde a distância $d_{p1,1}$ é calculada duas vezes (onda transmitida e recebida).

Figura 23 – Técnica SAFT e matriz de pontos.



Fonte: Do autor.

O valor da resolução lateral λ, que significa o passo entre cada ponto, implica em quantos destes irão compor a imagem. Resoluções maiores implicam em mais pontos na composição da matriz. Nessas implementações, o tamanho da matriz é um fator crucial no desempenho, seja em questão de tempo, seja se tratando de resolução lateral.

Isso se deve à dinâmica do algoritmo que, para cada ponto, deve ser calculado sua distância entre o elemento emissor e receptor. Quanto maior o número de pontos da matriz, mais cálculos de distâncias serão feitos e, dependendo da configuração emissor/receptor (principal diferença entre as abordagens), mais tempo será gasto para produzir a imagem.

Pela Figura 23 percebe-se que em SAFT, a cada iteração um elemento emite e recebe simultaneamente. Isso, como será apresentado no Capítulo 4, representa uma vanta-

gem no tempo de execução, tendo em vista que não são necessários dois laços de repetição, sendo um para os emissores, e um para os receptores. Em contrapartida, a imagem obtida apresenta contrastes piores, causados pelos lóbulos secundários, resultando em artefatos com maior intensidade na sua composição, como será apresentado no Capítulo seguinte.

O Algoritmo 8 apresenta o pseudocódigo referente ao processamento da matriz de pontos. No início da implementação (e para as restantes que serão apresentadas neste trabalho) são feitas as considerações iniciais, que incluem a declaração dos dados do experimento, como velocidade, frequência do transdutor e de amostragem, elementos do *array*, e a distância entre cada elemento. Inicialmente, são feitas também as declarações para dimensionamento da imagem, como o tamanho da placa (são desprezados os primeiros e últimos 5 cm por conta de efeitos do campo próximo), e a definição dos eixos x e z. Em um estágio posterior, realiza-se o cálculo da posição dos transdutores, e a resolução, discutida anteriormente.

Algoritmo 8 – SAFT

```
for z1 = 1 : Nz do                                   ▷ Variação dos pontos na dimensão z
    for x1 = 1 : Nx do                               ▷ Variação dos pontos na dimensão x
        sinal = 0;                                   ▷ Variável que preenche a matriz de pontos
        for Ne = 1 : emissor do                      ▷ Variação dos emissores. 'Emissor' = 128
            Nr = Ne;                                 ▷ Elemento receptor/emissor é igual (SAFT)
            De = sqrt((x(x1) − xn(Ne))^2 + z(z1)^2);
            Dr = 2 · De                              ▷ Cálculo da distância total
            k = round(Dr/resolucao);
            indice = k − tbegin − 1;                 ▷ Obtenção do índice da matriz de dados
            sinal = sinal + Data(Nr, indice, Ne);    ▷ Soma coerente
        end for
        pontos(z1,x1) = sinal;
    end for
end for
```

No Algoritmo 8, os termos z e x são os *arrays* com os valores dos eixos z e x, respectivamente. As posições dos elementos do transdutor são representadas pela variável x_n, N_x e N_z são os tamanhos dos vetores x e z respectivamente, t_{begin} é o índice de tempo inicial definido a partir da frequência de amostragem f_s e t_{min} (igual a D_{min}/c), a matriz de dados é representada pela variável *Data*, e a matriz dos resultados é a variável *pontos*.

O cálculo do índice de tempo inicial é feito a partir da frequência de amostragem, dimensão mínima da imagem e velocidade de propagação c. O tamanho da malha gerada, ou da matriz de pontos, é obtido a partir das dimensões da imagem (que pode ser reduzida, caso desejado) e com a resolução definida de antemão. Após essa sequência de atribuições e operações, há três laços de repetição *for* aninhados, sendo os dois primeiros para variar a posição dos pontos, em termos das dimensões z e x, e o terceiro para variar o elemento emissor (do elemento 1 ao 128) e receptor, caso aplicado somente na abordagem SAFT.

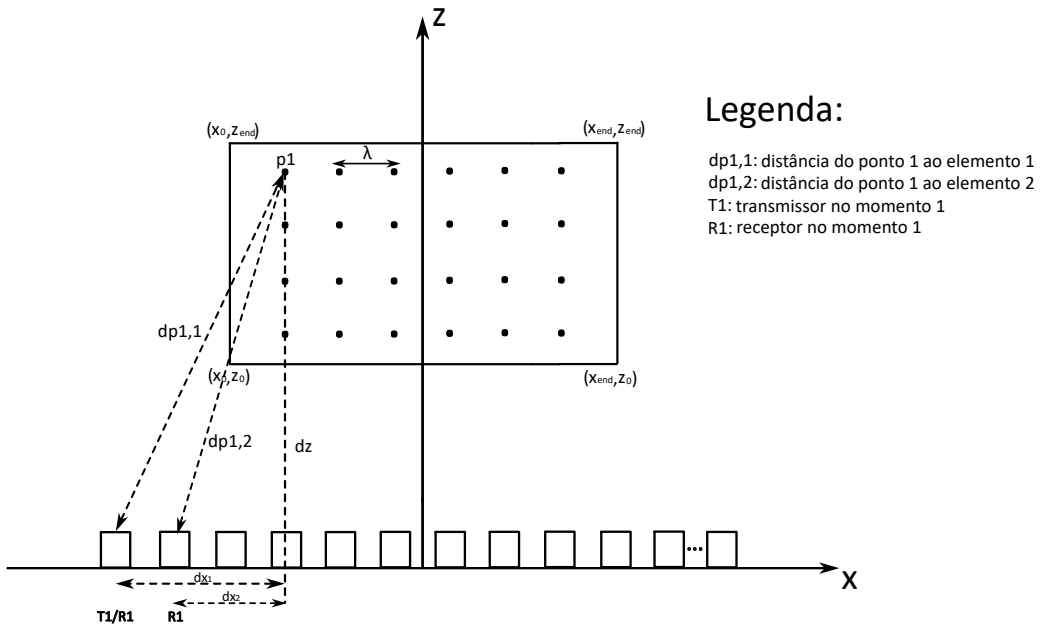
Dentro deste 3° laço mais interno, atribui-se o número do elemento emissor ao receptor, procedido pelo cálculo da distância do ponto ao elemento emissor/receptor. Esse cálculo é importante para determinar o índice presente no arquivo de dados que corresponde à distância medida. As informações presentes na matriz de dados são somadas a cada variação do elemento emissor/receptor, sofrendo atualização a cada novo ponto calculado, por meio de uma variável atribuída com o nome de "sinal".

Essa variável também é utilizada para preencher uma matriz com dimensões iguais à quantidade de pontos presentes na imagem para, em seguida, normalizá-los em torno do valor máximo, calcular a transformada de Hilbert, para extrair a envoltória do sinal (que representa as variações de amplitude do sinal ao longo do tempo) e, por fim, gerar a imagem.

3.3.1.2 Algoritmo 2R-SAFT

A Figura 24 ilustra a dinâmica emissor/receptor para esta técnica, onde em cada iteração um elemento emite e dois recebem.

Figura 24 – Técnica 2R-SAFT e matriz de pontos.



Fonte: Do autor.

A implementação do método 2R-SAFT apresenta como principal diferença o fato de o laço de repetição para os receptores ser executado duas vezes, exceto quando se chega ao último emissor, caso em que há apenas um receptor, conforme ilustrado na Figura 14. Ao final dos cálculos, os mesmos procedimentos de normalização e definição da envoltória dos sinais devem ser realizados antes da apresentação da imagem.

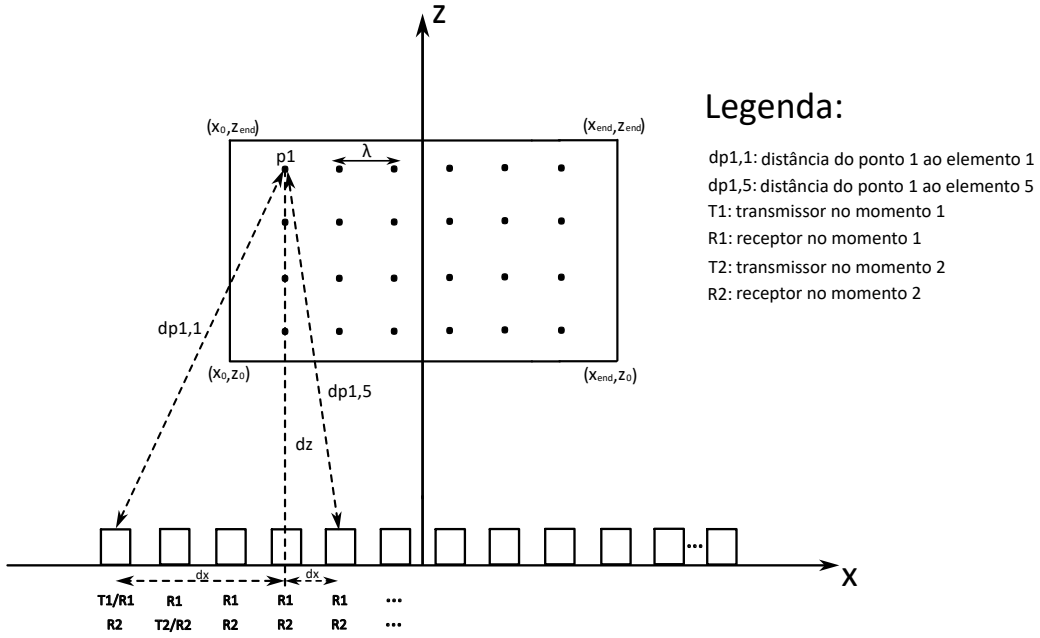
Isso pode ser visto também no Algoritmo 10 Apêndice A, onde o laço dos receptores se inicia do próprio emissor e termina no elemento seguinte, com exceção ao último

emissor, onde o receptor da onda refletida é o próprio elemento que emitiu.

3.3.1.3 Algoritmo SAFT-TFM

A Figura 25 apresenta a dinâmica para o método SAFT-TFM, em que todos os elementos do transdutor emitem e recebem, para cada iteração. Esta característica possibilita a focalização total.

Figura 25 – Técnica SAFT-TFM e matriz de pontos.



Fonte: Do autor.

A estratégia para implementação SAFT-TFM é parecida em partes com as que foram descritas para as abordagens SAFT e 2R-SAFT. A diferença dessa abordagem com relação às anteriores, é a forma dos laços de repetição dos elementos emissores e receptores, tendo em vista que no SAFT-TFM, para cada ponto, todos os elementos do transdutor emitem, e todos recebem.

Sendo assim, para esse caso haverá quatro laços de repetição *for* em sequência, dois para variação das dimensões z e x dos pontos, e outros dois para variação dos emissores e receptores.

O Algoritmo 9 também demonstra em linhas gerais o que foi citado. No laço responsável por variar os elementos emissores há o cálculo da distância do ponto ao emissor, e no laço mais interno, que varia os receptores, o valor da distância do emissor é somada com a distância ao receptor, para obtenção da distância total.

Ao final, assim como feito anteriormente em SAFT e 2R-SAFT, após essa série de instruções há a normalização da matriz de pontos, cálculo da transformada de Hilbert, e geração da imagem.

Algoritmo 9 – SAFT-TFM

```
for z1 = 1 : Nz do
    for x1 = 1 : Nx do
        sinal = 0;
        for Nr = 1 : receptor do                          ▷ Todos os elementos recebem
            Dr = sqrt((x(x1) − xn(Nr))^2 + z(z1)^2);
            for Ne = 1 : emissor do                       ▷ Todos os elementos emitem
                De = Dr + sqrt((x(x1) − xn(Ne))^2 + z(z1)^2);
                k = round(De/resolucao);
                indice = k − t_begin − 1;
                sinal = sinal + Data(Nr, indice, Ne);
            end for
        end for
        pontos(z1,x1) = sinal;
    end for
end for
```

3.3.1.4 Abordagem otimizada para o SAFT-TFM

A eliminação de cálculos redundantes neste ensaio parte da simetria entre os eixos x e z com a imagem a ser obtida. Na Figura 26 o ponto p_1, por exemplo, é obtido a partir do cálculo das distâncias aos elementos do transdutor, utilizando para isso o teorema de Pitágoras. Por sua vez, o ponto p_{1-sim} pode ser obtido sem o cálculo matemático, aproveitando o que foi feito previamente para o ponto p_1.

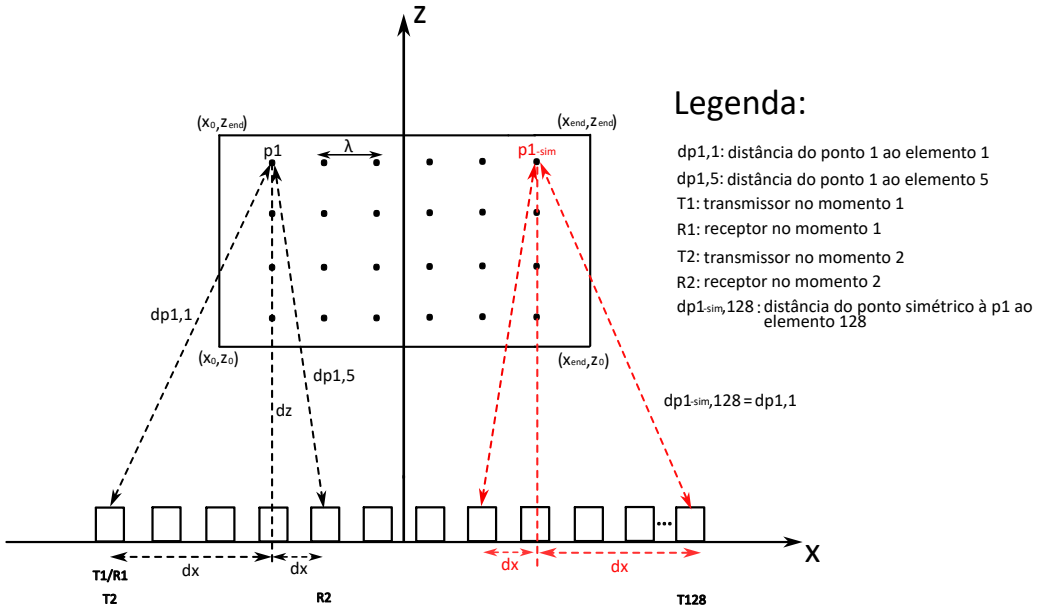
Para isso também deve ser levado em conta o elemento do transdutor, que também deve ser simétrico ao elemento emissor/receptor de p_1. A simetria entre os elementos do *array* se dá a partir de um "espelhamento" em torno do eixo z, onde os 64 primeiros elementos são simétricos aos 64 últimos. Sendo assim, o elemento 1 é simétrico ao elemento 128, 2 ao 127, 3 ao 126, e assim por diante. Na Figura 26, as distâncias simétricas que não precisam ser calculadas, e simplesmente atribuídas ao cálculo anteriormente realizado, estão na cor vermelha.

Essa dinâmica é representada nos algoritmos por meio de duas variáveis auxiliares (uma para o emissor simétrico e outra para o receptor simétrico), que servirão como forma de obter o elemento simétrico em cada iteração no *loop*. Essas variáveis são inicializadas em 0, e utilizadas para decrementar o valor do elemento correspondente, que se inicia no 128.

Essa alteração possibilita uma redução do número de iterações, tendo em vista que o laço *for* da variável x é diminuído pela metade. De forma resumida, nesta abordagem uma metade da imagem é efetivamente calculada, enquanto a outra é simplesmente atribuída. Essa abordagem pode ser vista por meio do Algoritmo 11 presente no Apêndice A, onde está implementado o método SAFT-TFM. As principais diferenças com relação ao Algoritmo 9 estão nas variáveis para o cálculo do ponto simétrico, bem como a indexação

para a matriz de pontos, onde a primeira metade recebe o que é calculado, enquanto a segunda metade é atribuída.

Figura 26 – Simetria entre a matriz de pontos e os eixos z e x.



Fonte: Do autor.

3.3.2 Implementação concorrente com a *Parallel Computing Toolbox*

Para a implementação paralelizada em Matlab, a base anteriormente descrita para a abordagem sequencial foi mantida, com a mudança do laço mais externo *for* para *parfor*. Com essa alteração, foram necessárias outras modificações devido à forma como o Matlab trabalha a paralelização de tarefas, conforme tratado no Capítulo 2. Dentre os casos citados no Capítulo 2 envolvendo a paralelização de um algoritmo utilizando a *toolbox* de paralelização, para a implementação aqui tratada, e apresentada no Algoritmo 12 Apêndice A, merece destaque a paralelização de laços aninhados, a definição dos limites superiores, e a utilização da variável de índices dos *loops*.

Na implementação realizada, para o caso da versão sem os cálculos redundantes, o cálculo do limite superior de x deve ser feito externamente ao laço *parfor*, atribuindo o resultado a uma variável utilizada como limite superior do *loop*. Portanto, a estrutura padrão desta implementação é modificada conforme as recomendações apresentadas nos Algoritmos 2, 4, e 6. Ressalta-se também uma pequena modificação no cálculo das distâncias para o eixo z, sendo que nesta abordagem ela foi fixada a cada iteração, de modo a diminuir cálculos repetitivos.

Além dos pontos citados e destacados, existem também outros detalhes que devem ser mencionados na implementação do *parfor*, que ficam bem exemplificadas pelo Algoritmo 12 no Apêndice A. Um desses detalhes é a utilização de variáveis "locais" para x, e

x_n dentro do laço. Isso se aplica em *arrays* que não têm seus valores alterados dentro do *loop*, conhecido como variáveis *broadcast*.

Este ponto é importante, pois quando se executa uma implementação com *parfor*, todas as variáveis indispensáveis à execução do *loop* são copiadas para cada *worker*, garantindo o acesso independente a essas variáveis. A ideia é evitar que os núcleos tentem acessar, de forma concorrente, a memória RAM, alocando previamente os dados necessários na memória *cache* do processador. Destaca-se que este acesso, nos algoritmos implementados, é feito exclusivamente pelo sistema operacional.

Também pode ser visualizado pelo Algoritmo 12 no Apêndice A, que existem duas matrizes *Data*: $Data_1$, e $Data_2$. Essa divisão da matriz de dados principal entre duas de mesmas dimensões se deu pela tentativa de utilizar o acesso *dual-channel* da memória RAM, o que não foi possível confirmar tal fato devido à falta de informações. Importante destacar que os tempos de acesso diminuíram, o que pode ser justificado pela memória *dual-channel*, ou pelo efeito da redução do tamanho da matriz.

Além da matriz de dados, outra que também foi dividida em duas é a matriz de pontos, com suas parcelas nomeadas por $pontos_1$ e $pontos_2$. Essa divisão se fez necessária devido ao fato do Matlab restringir a maneira como as variáveis são indexadas na paralelização, de modo a garantir a quantidade mínima de dados a serem enviados. Dessa forma, a indexação da matriz de pontos apresentada no Algoritmo 11 é incompatível quando se utiliza *parfor*. Posteriormente, as variáveis $pontos_1$ e $pontos_2$ são unidas em uma única matriz para a apresentação da imagem.

3.4 Implementações em *C* usando funções MEX

Serão apresentados, nesta seção, os algoritmos escritos em linguagem *C*. Destaca-se que foi utilizada somente a técnica SAFT-TFM, por se tratar da abordagem com pior tempo de execução.

3.4.1 Implementação sequencial em *C*

Utilizando funções MEX, foram realizadas modificações nos algoritmos apresentados anteriormente. Essa abordagem passa a ser muito vantajosa neste trabalho, pois permite que o processamento seja feito em linguagem de nível intermediário, e o resultado seja apresentado com a utilização das ferramentas de visualização do Matlab.

A implementação em *C* exigiu que algumas bibliotecas específicas fossem adicionadas, tais como "*stdio.h*" e "*stdlib.h*", padrões dos algoritmos em *C*. Foi incluída também a "*math.h*"para os cálculos necessários, além das bibliotecas "*mex.h*" e "*matrix.h*". A primeira é usada para gerenciar a execução da função MEX, possibilitando dentre outras funcionalidades, a definição do ponto de entrada do algoritmo "*mexFunction*". Por outro

lado, a segunda permite a interação com matrizes definidas como variáveis no Matlab, fornecendo funções importantes para leitura e manipulação de dados matriciais do Matlab como, por exemplo:

- ***"mxGetData"*** e ***"mxGetPr"***: acessam dados em *arrays* do Matlab. Utilizadas principalmente para ler a matriz de dados de entrada da função;
- ***"mxGetDimensions"***: obtém as dimensões da matriz de entrada;
- ***"mxCreateDoubleMatrix"***: função para criar matriz de saída. No algoritmo implementado, a matriz de saída é a matriz de pontos utilizada para a representação da imagem.

As funções citadas anteriormente, em conjunto com os vetores de entrada e saída da função MEX citados no Capítulo 2, permitem a leitura de variáveis do próprio Matlab, e a organização dos resultados também nessa estrutura. Para o algoritmo escrito usando linguagem C implementado, definiu-se como entrada para a função os vetores x e z, além da matriz de dados *Data*. A representação da imagem é feita com a matriz de pontos retornada pela função.

Um detalhe importante a ser destacado na implementação em C, é como a indexação das matrizes e dos *arrays* é feita, sendo diferente do formato em Matlab. Nas implementações em C com funções MEX, a indexação de matrizes é feita de forma linear, como um *array*. Esse formato segue o padrão coluna por coluna da matriz. Como exemplo desta indexação linear, a equação 3.1 apresenta uma matriz 2D m sendo transformada em um *array* linear

$$m = \begin{bmatrix} 1 & 4 & 2 \\ 3 & 5 & 8 \\ 6 & 9 & 7 \end{bmatrix} = \begin{bmatrix} 1 \\ 3 \\ 6 \\ 4 \\ 5 \\ 9 \\ 2 \\ 8 \\ 7 \end{bmatrix}. \tag{3.1}$$

Em se tratando de algoritmo, uma matriz 2D no formato $m(z,x)$ será indexada linearmente como apresentado na equação 3.2, sendo que a variável i varia os índices das linhas, j o índice das colunas, e z é o total de linhas

$$m(z,x) = m[i + z \cdot j]. \tag{3.2}$$

Para matrizes 3D, que é um caso também aplicado nas implementações feitas, este padrão de estrutura é mantido, com cada sub-matriz sendo transformada em sequência, como apresentado na equação 3.3

$$\left.\begin{array}{l} m(:,:,1) = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \\ m(:,:,2) = \begin{bmatrix} 5 & 6 \\ 7 & 8 \end{bmatrix} \end{array}\right\} \quad \begin{bmatrix} 1 \\ 3 \\ 2 \\ 4 \\ 5 \\ 7 \\ 6 \\ 8 \end{bmatrix}. \tag{3.3}$$

Essa alteração reflete no modo como são indexadas as matrizes no algoritmo, como apresentado na equação 3.4

$$m(R,indice,E) = m[N_r + R \cdot j + E \cdot indice \cdot N_e], \tag{3.4}$$

sendo N_e o parâmetro que varia a dimensão z da matriz, R é o número total de linhas, j varia a dimensão y, E é o total de sub-matrizes na dimensão z, *indice* é o total de colunas, e N_r varia as linhas da matriz.

Como a memória é organizada dessa forma, a matriz é apenas um artifício para facilitar a interpretação dos dados, porém com um custo no tempo de acesso, como será melhor evidenciado posteriormente. Portanto, trata-se de uma transformação bastante eficiente para a proposta aqui tratada. Para maiores detalhes do algoritmo implementado, consultar Apêndice B, Algoritmo 14.

3.4.2 Implementação concorrente em *C*

A estratégia para essa implementação é executar, em linguagem *C*, o que o *parfor* faz no Matlab, a partir da *Parallel Computing Toolbox*. Nesta versão há o acréscimo das funções de paralelização de tarefas com a biblioteca "*windows.h*", que foram anteriormente apresentadas no Capítulo 2. Válido destacar que para a paralelização, foi criado inicialmente uma *struct* com todos os parâmetros que cada *thread* recebe, sendo eles:

- *Data*: cada *thread* recebe uma cópia deste parâmetro para facilitar o acesso aos dados durante o processamento;
- z: *array* com os valores do eixo z, sendo este parâmetro dividido entre as *threads*. A justificativa da divisão é pelo fato de estar associado ao laço mais externo da estrutura;

- x: *array* com os valores do eixo x. Ele também é passado como parâmetro para cada *thread* como forma de garantir o acesso de cada uma a esta variável que, assim como as outras, é necessária para o cálculo;

- *numel*: trata-se de uma variável escalar, sendo o número de pontos do *array* z a ser divido entre cada *thread*. O cálculo dessa divisão é feito entre o total de elementos de z e o número de *threads* disponível;

- *pontos*: matriz de saída do algoritmo. Cada *thread* recebe uma parte desta variável, sendo definida pela divisão entre o número de elementos de z e o número de *threads*. Essa divisão é importante para que cada parte da matriz de saída seja feita de forma independente, melhorando o tempo de execução;

- N_x: tamanho do vetor x. A justificativa deste parâmetro é pelo fato dele ser usado como limite superior do laço de x, e também como critério para indexação de matrizes, operação que cada *thread* realiza durante o processamento;

- N_z: comprimento do vetor z. A justificativa da sua presença é semelhante ao caso da variável N_x, exceto pelo fato que N_z não será mais o limite superior do laço de z, e sim o resultado da razão entre o tamanho do vetor com as *threads* disponíveis.

A função *mexCallMatlab* foi utilizada para identificar o número de *threads* disponíveis, que por padrão será igual ao número de *cores* do processador, como forma de garantia que cada núcleo físico tenha apenas um conjunto de tarefas, reduzindo o risco de sobrecarga de trabalho.

Destaca-se que os parâmetros anteriormente citados são fornecidos como argumentos a cada *thread*, através de um laço de repetição. Ressalta-se que as variáveis z e *pontos* são divididas entre cada *thread* por meio de uma variável "*offset*", obtida por meio da equação 3.5

$$offset = ceil(numel/N_{threads}), \tag{3.5}$$

sendo *numel* o número de pontos do *array* z, e $N_{threads}$ o número de *threads* disponíveis. A função *ceil* é utilizada para arredondar o resultado para o inteiro mais próximo que seja maior ou igual ao resultado da divisão. Esse arredondamento se faz necessário, pois o resultado deste cálculo é usado como índice superior do laço z (definido como a variável chamada "*imax*").

Assim como em outras implementações, um pseudocódigo (15) da implementação pode ser encontrado no Apêndice B. Para mais detalhes envolvendo o procedimento utilizado para implementação do algoritmo, recomenda-se a leitura do trabalho apresentado por Oreoman (2024).

3.5 Considerações parciais

Neste Capítulo foram feitas as descrições do ensaio, da plataforma utilizada para testes, e das implementações realizadas, tanto na forma sequencial, quanto na forma paralelizada. Pode ser destacado, por exemplo, o detalhamento do sistema de aquisição, especificidades da peça, e formato da matriz de pontos usada. As principais características do computador também foram pontuadas, como forma de comparação com outros resultados. A representação das abordagens SAFT, 2R-SAFT, e SAFT-TFM em pseudocódigos destaca os principais métodos feitos. As estratégias para a diminuição do tempo de execução foram tratadas de forma descritiva, com as principais diferenças entre cada uma sendo destacadas.

Os conteúdos abordados neste Capítulo serão necessários para uma melhor interpretação dos tópicos a serem discutidos a seguir, que apresentará, e comparará os resultados obtidos. A importância dos assuntos que foram tratados são essenciais nas métricas utilizadas para avaliação do desempenho dos algoritmos, sendo a resolução lateral de cada técnica SA, e o tempo de execução de cada implementação feita.

4 Resultados

Este Capítulo apresenta os principais resultados obtidos neste trabalho. Nesta etapa, serão realizadas análises comparativas entre as resoluções laterais das imagens ao aplicar cada abordagem SA, além de apresentar os tempos de execução dos algoritmos propostos anteriormente, considerando implementações tanto na forma sequencial, quanto na forma paralela. Foram realizados testes com imagens de dimensões 20 mm x 20 mm, com uma resolução igual a λ.

A decisão de fixar os testes nestes dois parâmetros se dá pelo motivo da repetição do conteúdo. E para fins de comparação, principal objetivo deste trabalho, a fixação destes dois indicadores para todos os testes é essencial para garantir a consistência e validade dos resultados obtidos. Outro ponto a se destacar é a definição da abordagem SAFT-TFM para os testes feitos com a versão otimizada, com a *Parallel Computing Toolbox*, e com a linguagem *C*. Como será apresentado, trata-se da técnica mais exigente computacionalmente (N^2 cálculos, sendo N o número de elementos do transdutor), com tempos de execução mais longos.

Dessa forma, a comparação e análise de escalabilidade das novas implementações é melhor evidenciada com esta abordagem. A medição dos tempos de execução levou em conta a média de 4 amostras, com o computador ativado no modo avião, e apenas o Matlab aberto em primeiro plano, tendo em vista que o sistema operacional gerencia programas em camadas secundárias.

Por fim, a organização deste Capítulo é feita com a separação dos resultados obtidos em Matlab e em *C*, e uma comparação entre essas duas abordagens. As imagens obtidas a partir de cada técnica SA são apresentadas somente uma vez, levando em conta que são equivalentes para as estratégias de implementação abordadas. Os tempos de execução são comparados e discutidos em três momentos, sendo um para Matlab, outro para *C*, e um último entre essas duas estratégias, ponderando a vantagem ou desvantagem que uma tem sobre a outra.

4.1 Implementações em Matlab

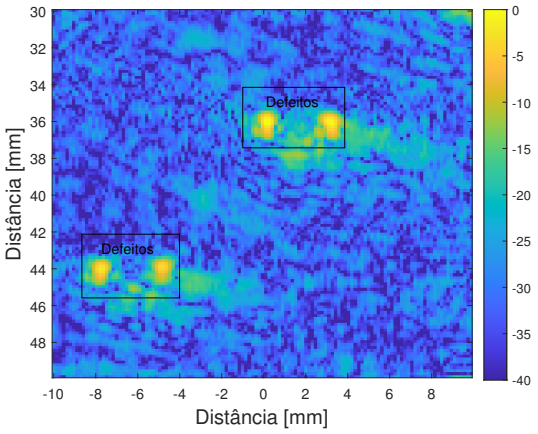
Nesta seção serão apresentados os resultados obtidos com as implementações em Matlab. Sendo assim, serão comparadas e discutidas as imagens obtidas com cada um dos três métodos de focalização, e os tempos de processamento para cada abordagem.

4.1.1 Comparação entre cada abordagem SA

4.1.1.1 SAFT

A imagem obtida com a técnica SAFT é apresentada na Figura 27, com os defeitos da placa sendo destacados. A apresentação das imagens obtidas levou-se em consideração um limite de magnitude de -40 dB.

Figura 27 – Imagem obtida com SAFT.



Fonte: Do autor.

Pela Figura 27 observa-se que os defeitos estão bem destacados na imagem, apesar da presença de artefatos principalmente em regiões próximas aos defeitos. O artefato é uma representação na imagem que se assemelha ao defeito, podendo resultar em diagnósticos incorretos. Seu surgimento se dá pela aquisição de um sinal com certa amplitude que provoca essa representação nas proximidades do local. Destaca-se também que ele pode ser causado por valores incorretos relacionados ao ensaio, como a distância entre os elementos do *array*, por exemplo.

A abordagem SAFT demandou um tempo de execução igual a 0,15 s, levando em consideração que ela foi implementada apenas em Matlab, no formato sequencial. Esse tempo de execução, comparado às outras técnicas SA, é baixo. Isso se deve ao número de cálculos (N), que é menor devido à presença de um único elemento como emissor/receptor.

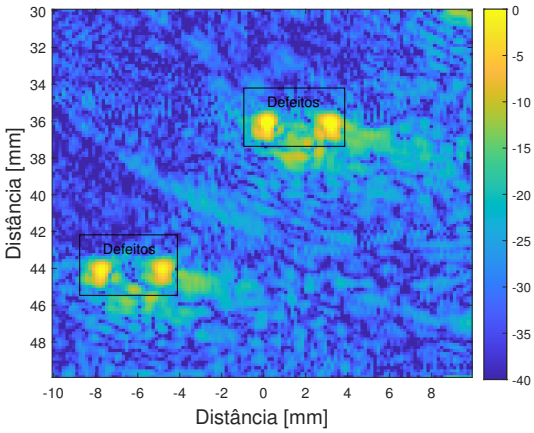
4.1.1.2 2R-SAFT

A Figura 28 apresenta a imagem obtida com a técnica 2R-SAFT. Comparada à abordagem SAFT, esse resultado é muito parecido, inclusive na representação dos defeitos. Por serem feitos mais cálculos em 2R-SAFT ($2N - 1$), obtém-se uma ligeira melhoria no contraste da imagem, sendo perceptível principalmente nas proximidades da região superior esquerda, pois apresenta artefatos com menos intensidade.

No caso aqui tratado, como pôde ser visualizado, 2R-SAFT não apresentou grandes vantagens, em termos de resolução lateral, sobre o método SAFT. Seu tempo de

execução, na implementação sequencial no Matlab, foi de 0,28 s. Um leve acréscimo de 0,13 s comparada à abordagem SAFT.

Figura 28 – Imagem obtida com 2R-SAFT.

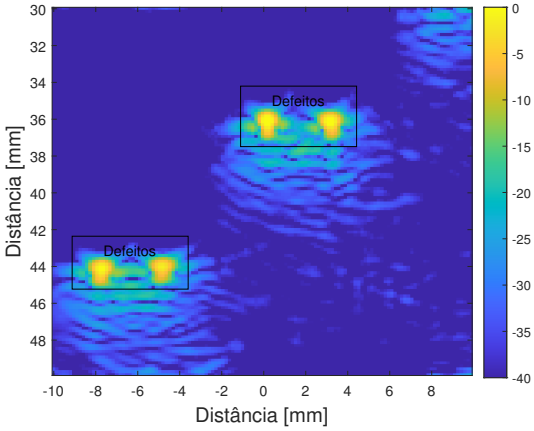


Fonte: Do autor.

4.1.1.3 SAFT-TFM

A imagem obtida com a técnica SAFT-TFM é apresentada pela Figura 29.

Figura 29 – Imagem obtida com SAFT-TFM.



Fonte: Do autor.

Comparada às outras técnicas apresentadas anteriormente, percebe-se claramente uma menor intensidade de artefatos de modo geral, e um aumento no contraste da imagem. Esse resultado é justificado pelo grande aumento no número de cálculos, sendo da ordem de N^2 para o SAFT-TFM. A presença dos artefatos se dá predominantemente nas regiões próximas aos defeitos da peça.

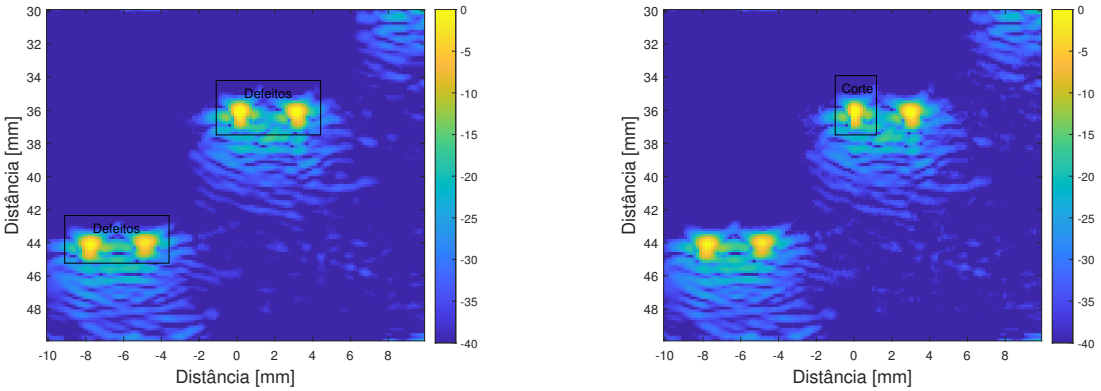
Como a abordagem SAFT-TFM foi implementada em todas as estratégias aqui descritas, a apresentação dos seus tempos de execução se dará na forma de gráfico, sendo comentado a partir da seção 4.1.3.

4.1.2 Implementação otimizada

A Figura 30 apresenta e compara as imagens obtidas para SAFT-TFM. Na Figura 30 (a) é exibida a versão com cálculos redundantes, que é a mesma imagem apresentada anteriormente. A Figura 30 (b), por sua vez, apresenta o resultado obtido para a versão otimizada, sem os cálculos redundantes.

Figura 30 – Resultados com e sem cálculos redundantes. SAFT-TFM.

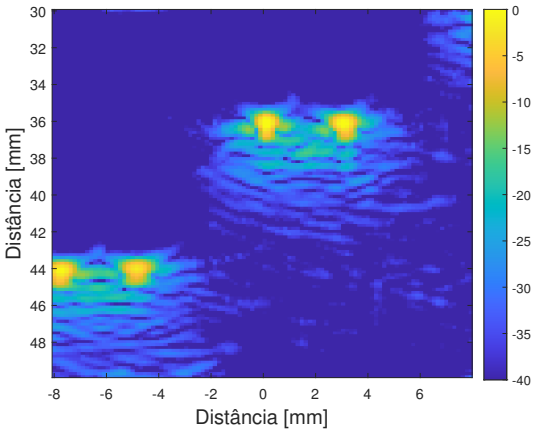
(a) Com cálculos redundantes. (b) Sem cálculos redundantes.



Fonte: Do autor.

Essa comparação foi feita para evidenciar, de forma mais fácil, um pequeno corte existente no meio da imagem obtida com a versão otimizada. Embora seja um detalhe sutil, é importante destacar que ele é causado pela divisão ao meio do *array* x. Quando o número de elementos desse vetor é ímpar, o resultado da divisão não é um inteiro, sendo assim arredondado para poder ser utilizado como índice. Porém, esse cálculo faz com que os pontos no centro da imagem não sejam contemplados, resultando neste corte. A Figura 31 ilustra um caso onde isso não acontece, pois o tamanho do vetor x é par.

Figura 31 – Imagem com número de pontos de x par.



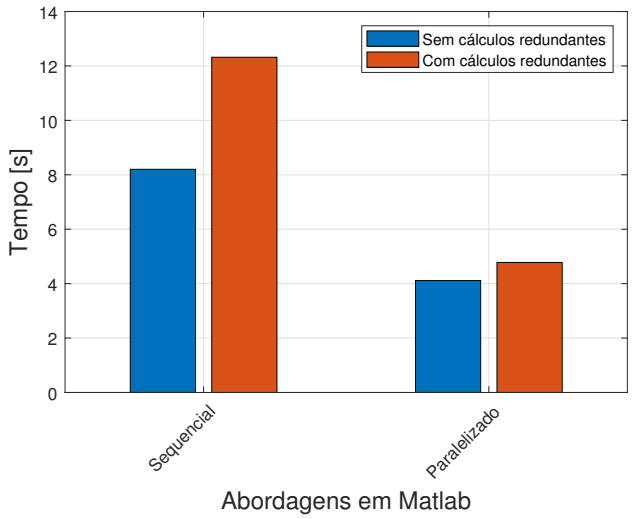
Fonte: Do autor.

De forma geral, as imagens obtidas nas versões com ou sem cálculos redundantes são muito parecidas, com o detalhe do recorte no centro podendo até mesmo ser insignificante. Importante deixar claro que essa abordagem apresentada oferece resultados confiáveis apenas quando o eixo x está dividido igualmente em relação ao eixo z. Caso a divisão da matriz resulte em partes assimétricas, com dimensões diferentes, esta técnica não é viável, pois apresentará resultados incorretos.

4.1.3 Tempos de execução para implementações em Matlab

A Figura 32 apresenta os tempos de execução medidos para as estratégias implementadas em Matlab, considerando apenas a abordagem SAFT-TFM.

Figura 32 – Tempos de execução em Matlab.



Fonte: Do autor.

Tanto na implementação sequencial, quanto na concorrente, a versão otimizada apresentou tempos melhores, com um ganho de 33,3 % para a abordagem sequencial, e 13,9 % para a versão paralela. O decréscimo pela metade do laço de x resultou em uma redução pela metade das iterações. Porém, o tempo de processamento não seguiu esta ordem, pois na versão otimizada existem outras operações, principalmente o acesso à matriz de dados, que ocorre duas vezes.

A eliminação de cálculos redundantes, conforme a Figura 32 apresenta, resultou em um ganho maior para a versão sequencial, comparada à abordagem paralela. Isso pode ser justificado pela sobrecarga no algoritmo paralelizado, que envolve o gerenciamento dos *workers* e a comunicação entre eles. Na implementação sequencial, a redução dos cálculos afeta diretamente o tempo de execução, de forma mais eficaz. Na versão concorrente, por sua vez, esse ganho é mais limitado devido ao sincronismo entre os *workers*, e a distribuição e organização dos resultados em diferentes iterações do *loop parfor*.

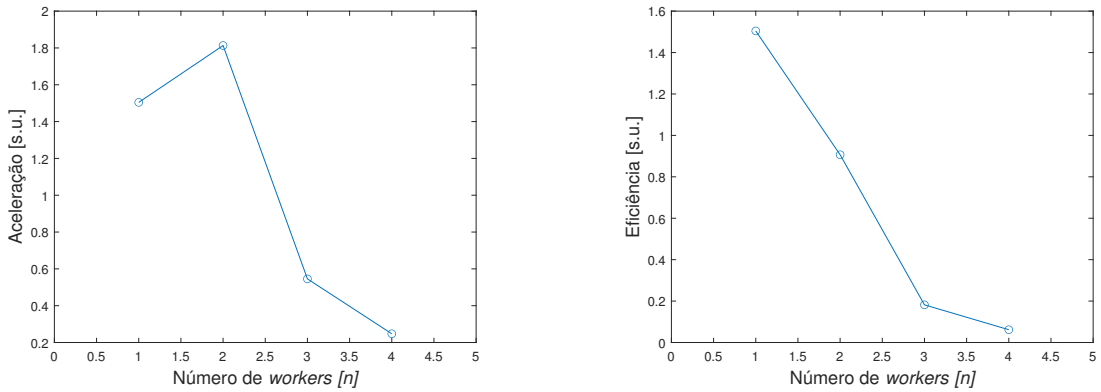
Com relação às abordagens sequencial e paralela de modo geral, a versão concorrente apresenta resultados melhores, como esperado. Considerando todos os cálculos, foi obtido um ganho de 60,0 % no tempo de execução. Para as versões otimizadas, o ganho foi menor, sendo da ordem de 48,6 %.

Conforme apresentado, a *Parallel Computing Toolbox* apresenta resultados satisfatórios, o que justifica sua utilização. Entretanto, seus ganhos são obtidos com a inicialização prévia dos *workers*, pois caso contrário a versão paralela pode apresentar resultados insatisfatórios, de acordo com a MathWorks (2024d).

Como forma de confirmar tal questão, a Figura 33 apresenta a relação de aceleração e eficiência durante a inicialização dos *workers*. Os gráficos foram obtidos por meio da execução do processamento para cada número de *workers* (1 a 4) inicializados, realizando a coleta do tempo de execução e o cálculo da razão de aceleração e eficiência.

Figura 33 – Aceleração e eficiência em função do número de *workers* inicializados.

(a) Relação de aceleração. (b) Eficiência na inicialização dos *workers*.



Fonte: Do autor.

A razão de aceleração foi calculado conforme equação 4.1

$$speed_{ratio} = t_{sequencial}/t_{paralelo}, \tag{4.1}$$

sendo $t_{sequencial}$ o tempo em $[s]$ para o algoritmo sequencial (mantido fixo como sendo 8,2 s), e $t_{paralelo}$ o tempo também em $[s]$ para as implementações concorrentes, coletadas em cada processamento com o número de *workers* utilizado. Pela equação 4.1, como $t_{sequencial}$ é mantido constante, o resultado é alterado apenas por $t_{paralelo}$, onde à medida que aumenta diminui a razão, e à medida que é reduzido de valor a razão de aceleração aumenta. Com a inicialização de mais *workers*, é perceptível o aumento no tempo de processamento, e a diminuição da relação de aceleração.

A eficiência, por sua vez, é obtida diretamente com o valor calculado para a razão de aceleração, conforme apresenta a equação 4.2

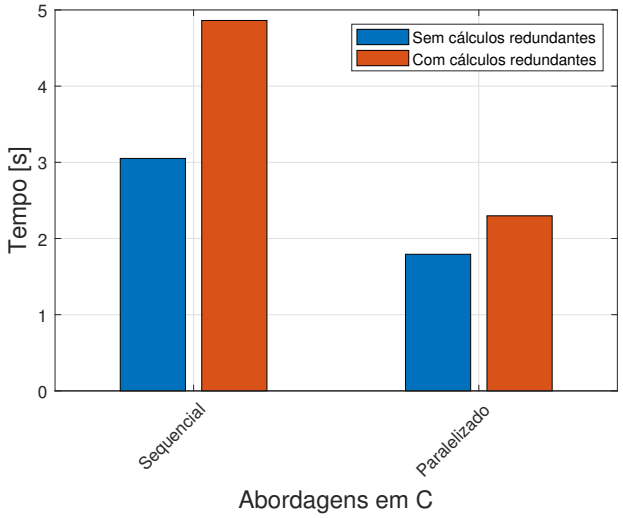
$$eficiencia = speed_{ratio}/N_{workers}, \tag{4.2}$$

sendo $speed_{ratio}$ a razão explicada anteriormente, e $N_{workers}$ o número de *workers*, que varia de 1 a 4. Para maiores detalhes, consultar Algoritmo 13, Apêndice A.

4.2 Implementações em *C* com funções MEX

Nesta seção serão discutidos somente os resultados relacionados aos tempos de execução para os algoritmos em *C*, pois as imagens obtidas foram as mesmas anteriormente destacadas, com a técnica SAFT-TFM. A Figura 34 apresenta os tempos de execução obtidos para as implementações em *C*, considerando as versões com e sem otimização, nas abordagens sequencial e paralela.

Figura 34 – Tempos de execução em *C*.



Fonte: Do autor.

Assim como os resultados em Matlab apresentaram, a versão otimizada em *C* também se mostrou mais eficiente, com tempos de execução menores. Considerando a implementação sequencial, foi possível obter um ganho de 37,2 % no tempo de processamento, enquanto para a abordagem paralela, o ganho novamente foi menor, sendo aproximadamente de 22,0 %. A justificativa do ganho não ser proporcional à redução do número de iterações é a mesma anteriormente destacada, pois na implementação otimizada há acesso duplo ao *array* de dados.

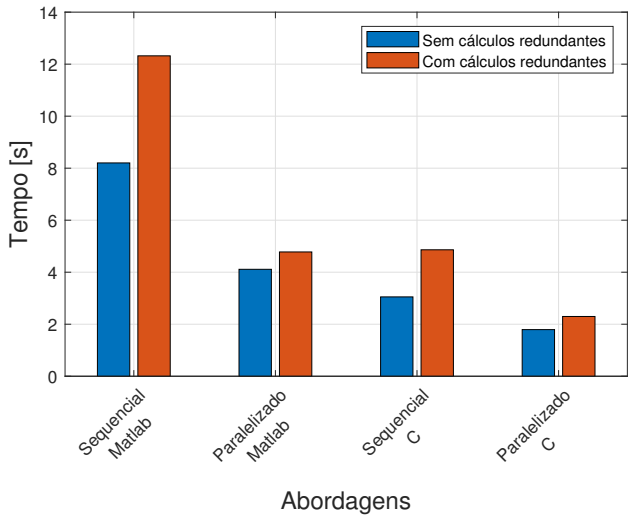
Os resultados para a versão otimizada também apresentaram maior ganho na implementação sequencial, em virtude do gerenciamento das *threads* realizado pelo sistema operacional, que resulta em um ganho menor quando se elimina os cálculos redundantes. De modo geral, comparando as implementações sequencial e paralela, os algoritmos concorrentes apresentaram, como esperado, considerável ganho no tempo de processamento.

Com todos os cálculos realizados, a paralelização apresentou um tempo de execução 52,7 % melhor, o que é um pouco inferior ao ganho de 60,0 % que Oreoman (2024) obteve para o caso que tratou. Esse resultado depende diretamente do que é realizado no algoritmo, e da carga de trabalho resultante dos cálculos realizados. Considerando a abordagem otimizada, por sua vez, a paralelização apresentou um ganho de 41,2 % no tempo de execução, comparada à abordagem sequencial.

4.3 Comparação entre os tempos de execução em Matlab e *C*

Nesta seção serão apresentados e comparados os tempos de execução em Matlab e *C*, com a técnica SAFT-TFM. A Figura 35 apresenta os resultados obtidos.

Figura 35 – Tempos de execução em Matlab e *C*.



Fonte: Do autor.

Pelos resultados, é perceptível a grande diferença que existe nas implementações em *C*, sendo que sua pior abordagem (sequencial com todos os cálculos) apresentou um tempo de execução semelhante ao obtido na melhor versão em Matlab, com a otimização e paralelização com *parfor*, apresentando uma ligeira diferença de 0,75 s. A justificativa se baseia, dentre outros fatores, na diferença entre um código interpretado e compilado, sendo o segundo mais eficiente, pois o código de máquina é traduzido pelo compilador antes de ser executado, o que não acontece para o primeiro caso.

Uma comparação importante envolve o ganho obtido com a otimização em Matlab e *C*, principalmente com a paralelização, pois é um indicativo da eficiência no gerenciamento de tarefas paralelas. No algoritmo concorrente implementado em Matlab, foi obtido um ganho de aproximadamente 13,9 %, enquanto para a versão equivalente em *C* o fator foi de 22,0 % . Esse resultado demonstra que em *C* o gerenciamento dos *threads* a partir da

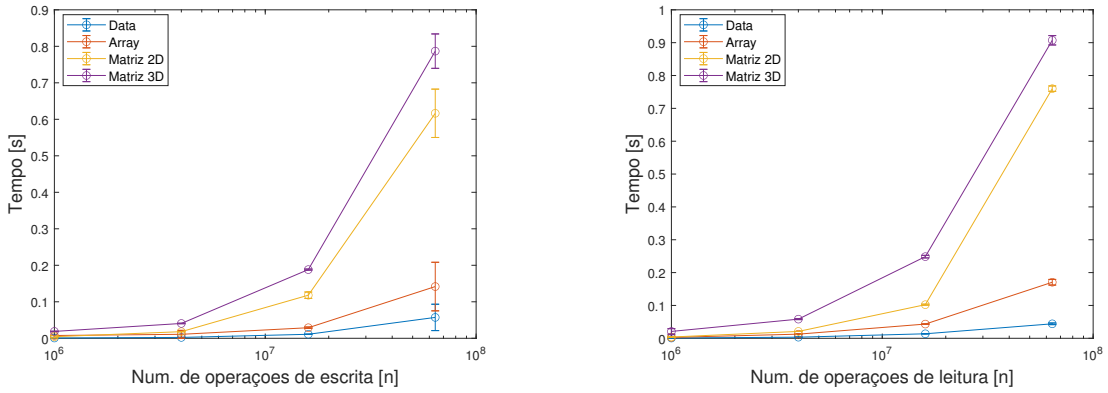
biblioteca *Windows.h* é feito de forma mais eficaz comparado ao Matlab, com a utilização da *Parallel Computing Toolbox*.

Comparando o pior (sequencial em Matlab com todos os cálculos) e o melhor caso (paralelizado e otimizado em *C*) dentre todos, a diferença obtida foi de aproximadamente 10,5 s, representando um ganho de 85,4 % no tempo de processamento. Trata-se de uma redução drástica, considerando a pior abordagem SA em termos de tempo de processamento.

Essa redução também pode ser explicada pelo formato em que os dados se encontram organizados. A Figura 36 compara o tempo, para operações de leitura e escrita, demandado por diferentes tipos de variáveis, desde escalar à matriz 3D. Nas implementações em Matlab os dados foram representados em matriz 3D. Em *C*, por sua vez, a representação seguiu o formato de *array* linear.

Figura 36 – Tempos de execução em escrita e leitura para diferentes tipos de variáveis.

(a) Operação de escrita. (b) Operação de leitura.



Fonte: Do autor.

A diferença no tempo de leitura para essas variáveis sustenta a discrepância para os resultados apresentados. Por mais que a matriz 3D seja apenas um artifício para facilitar a interpretação dos dados, suas operações são mais complexas e exigem maior tempo de operação.

4.4 Considerações parciais

Este Capítulo apresentou os resultados obtidos, e os casos estudados e analisados para cada uma das três técnicas de formação de imagens acústicas, em conjunto com as abordagens para diminuição do tempo de execução. Em relação à resolução lateral, SAFT-TFM se sobressai comparado aos métodos SAFT e 2R-SAFT, demandando, entretanto, tempos de execução consideravelmente superiores.

Com relação às abordagens para diminuição do tempo de processamento, foi possível obter um ganho razoável com as implementações otimizadas. Os algoritmos concor-

rentes, por sua vez, apresentaram resultados significativos, com a redução no tempo de execução em ambos os casos, tanto em Matlab, quanto em *C*.

O destaque principal se dá pelas implementações em linguagem *C*, principalmente com a versão paralelizada e sem cálculos redundantes, apresentando uma redução no tempo de quase 7 vezes comparado à implementação sequencial e com todos os cálculos no Matlab. Aplicando esta estratégia com a abordagem SAFT, por exemplo, seria possível obter imagens equivalentes em tempo real, com um *hardware* de baixo custo.

5 Considerações finais

Este trabalho apresentou e comparou implementações de técnicas para geração de imagens acústicas, com aplicação em avaliações não destrutivas para análise da integralidade de uma peça, material, etc. A apresentação de trabalhos consolidados, bem como áreas de pesquisa dentro deste assunto permitiu a constatação da consolidação dessa área, além da ponderação dos atuais desafios.

Ressalta-se também o desenvolvimento feito para assuntos teóricos envolvendo acústica e processamento de imagem (como ondas ultrassônicas, transdutores, e sistemas de excitação e aquisição), a descrição do ensaio, e o detalhamento das implementações feitas, o que possibilitou alcançar e comparar os resultados obtidos.

É evidenciada a superioridade, em termos de tempo de execução, para as implementações otimizadas e concorrentes, tanto em Matlab quanto em *C*. A partir dos resultados obtidos, evidencia-se a eficácia da *Parallel Computing Toolbox* para a paralelização em Matlab, e da biblioteca *Windows.h* para a linguagem *C*, bem como os desafios que cada abordagem apresenta.

Os algoritmos concorrentes implementados no Matlab oferecem uma alternativa mais fácil e prática para implementações concorrentes, sem a exigência do mesmo nível de detalhamento e controle feito em *C*. O ponto negativo fica por conta da inicialização das instâncias de paralelização, que demandam um tempo considerável principalmente para o caso aqui tratado, em que há uma carga de trabalho exaustiva.

Os resultados obtidos permitiram também comparar o gerenciamento das tarefas paralelizadas em cada caso, com mais eficácia por parte da biblioteca *Windows.h* sobre a *Parallel Computing Toolbox*. Dentre os resultados, destaca-se a diferença de 10,53 s obtida no tempo de execução entre o pior cenário em Matlab para o melhor caso em *C*. A implementação em uma linguagem de nível inferior, bem como a organização dos dados em um *array* linear justificam a diferença obtida, especialmente em operações intensivas com matrizes 3D.

Portanto, implementações concorrentes que permitem melhor uso da CPU, bem como organizações de dados em formato que exigem operações menos complexas, são totalmente válidas na geração de imagens acústicas. Levando em consideração o contínuo aumento do poder computacional, abordagens como as que foram apresentadas são amplamente eficientes em termos de escalabilidade e desempenho. A fundamentação é apoiada pela redução de quase sete vezes no tempo de execução, ao manter o mesmo *hardware* e alterar apenas a metodologia de implementação. Em termos práticos, seria possível, nas devidas proporções, obter imagens em tempo real com um *hardware* de baixo custo e fácil acesso.

5.1 Propostas para trabalhos futuros

Como continuação para a linha de pesquisa apresentada neste trabalho, a complementação pode ser feita, por exemplo, com a utilização da GPU integrada ao processador. A capacidade de incorporar esse componente ao processamento pode reduzir os tempos de execução, considerando o ganho de escalabilidade computacional.

Outra possibilidade, que não foi abordada, seria a virtualização de processos na paralelização de algoritmos. Podendo estar em um único processador, ou espalhados por diferentes máquinas, a virtualização oferece uma camada de abstração do *hardware*, podendo resultar em uma melhor capacidade computacional, ou apresentar resultados não tão satisfatórios, devido à sobrecarga adicional e acessos concorrentes à memória.

Uma última proposta seria a implementação de abordagens equivalentes em sistema operacional *Linux*, pois ele permite ao usuário maior liberdade no gerenciamento e customização das operações a serem feitas. No *Windows*, tal gerenciamento não é possível, além de ser um sistema operacional que trabalha (principalmente versões mais novas) com muitas interfaces gráficas de usuário, aumentando as operações em segundo plano.

Coordenar o sistema operacional para operar apenas em tarefas específicas, e gerenciar o acesso dos *cores* à memória por meio de implementações em nível de *Kernel* certamente é mais desafiador, porém com resultados possivelmente melhores aos que foram aqui apresentados.

Referências

ADAMOWSKI, J. C. et al. Caracterização de líquidos por ultrassom. In: *Sensores - Tecnologia e Aplicações*. São Paulo: Universidade De São Paulo, 2004. p. 50.

BIRK, M. et al. Gpu-based iterative transmission reconstruction in 3d ultrasound computer tomography. *Journal of Parallel and Distributed Computing*, v. 74, n. 1, p. 1730–1743, 2014. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0743731513002037>. doi: https://doi.org/10.1016/j.jpdc.2013.09.007.

BONANNELLA, L. *A Guide to Python Multiprocessing and Parallel Programming*. 2022. Disponível em: <https://www.sitepoint.com/python-multiprocessing-parallel-programming/>. Acesso em: 28 de jul. de 2024.

COSTA, E. T. et al. Transdutores de ultrassom: Modelagem, construção e caracterização. In: *Sensores - Tecnologia e Aplicações*. São Paulo: Universidade De São Paulo, 2004. p. 97.

DRINKWATER, B. W.; WILCOX, P. D. Ultrasonic arrays for non-destructive evaluation: A review. *NDT & E International*, v. 39, n. 7, p. 525–541, 2006. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0963869506000272>. doi: https://doi.org/10.1016/j.ndteint.2006.03.006.

GEORGE, A. D.; HAWKES, L. W. *Microprocessor-Based Parallel Architecture for Reliable Digital Signal Processing Systems*. Boca Raton, Florida, USA: CRC Press, 1992.

GUAN, X.; HE, J.; RASSELKORDE, E. M. A time-domain synthetic aperture ultrasound imaging method for material flaw quantification with validations on small-scale artificial and natural flaws. *Ultrasonics*, Elsevier, v. 56, p. 487–496, 2015.

HOLMES, C.; DRINKWATER, B. W.; WILCOX, P. D. Post-processing of the full matrix of ultrasonic transmit–receive array data for non-destructive evaluation. *NDT & E International*, v. 38, n. 8, p. 701–711, 2005. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0963869505000721>. doi: https://doi.org/10.1016/j.ndteint.2005.04.002.

INTEL. *Intel® 64 and IA-32 Architectures Software Developer's Manual*. Volume 1: Basic architecture. Santa Clara, California, USA, 2024.

IRIARTE, J. M.; COSARINSKY, G.; BRIZUELA, J. Synthetic aperture ultrasound imaging using gpus. In: IEEE. *IEEE CACIDI 2016 - IEEE Conference on Computer Sciences*. Piscataway, New Jersey, USA, 2016. p. 1–5. doi: 10.1109/CACIDI.2016.7785996.

JENSEN, J. A. et al. Synthetic aperture ultrasound imaging. *Ultrasonics*, v. 44, p. e5–e15, 2006. Proceedings of Ultrasonics International (UI'05) and World Congress on Ultrasonics (WCU). Disponível em: <https://www.sciencedirect.com/science/article/pii/S0041624X06003374>. doi: https://doi.org/10.1016/j.ultras.2006.07.017.

KARAMAN, M.; BILGE, H.; O'DONNELL, M. Adaptive multi-element synthetic aperture imaging with motion and phase aberration correction. *IEEE Transactions on Ultrasonics, Ferroelectrics, and Frequency Control*, v. 45, n. 4, p. 1077–1087, 1998. doi: 10.1109/58.710591.

KARAMAN, M.; LI, P.-C.; O'DONNELL, M. Synthetic aperture imaging for small scale systems. *IEEE Transactions on Ultrasonics, Ferroelectrics, and Frequency Control*, v. 42, n. 3, p. 429–442, 1995. doi: 10.1109/58.384453.

KJELDSEN, T. et al. Synthetic aperture sequential beamforming implemented on multi-core platforms. In: IEEE. *2014 IEEE International Ultrasonics Symposium*. Piscataway, New Jersey, USA, 2014. p. 2181–2184. doi: 10.1109/ULTSYM.2014.0543.

MATHWORKS. *Call MEX Functions*. 2024. Disponível em: <https://www.mathworks.com/help/matlab/call-mex-functions.html>. Acesso em: 21 de ago. de 2024.

MATHWORKS. *Decide When to Use parfor*. 2024. Disponível em: <https://www.mathworks.com/help/parallel-computing/decide-when-to-use-parfor.html>. Acesso em: 19 de jul. de 2024.

MATHWORKS. *Nested parfor and for-Loops and Other parfor Requirements*. 2024. Disponível em: <https://www.mathworks.com/help/parallel-computing/nested-parfor-loops-and-for-loops.html>. Acesso em: 19 de jul. de 2024.

MATHWORKS. *Quick Start Parallel Computing in MATLAB*. 2024. Disponível em: <https://www.mathworks.com/help/parallel-computing/quick-start-parallel-computing-in-matlab.html#mw_70c1fc86-9b3f-42d1-b0a5-2a568c449314>. Acesso em: 27 de jul. de 2024.

MATHWORKS. *What Is Parallel Computing?* 2024. Disponível em: <https://www.mathworks.com/help/parallel-computing/what-is-parallel-computing.html>. Acesso em: 28 de jul. de 2024.

OLAH, M. J. *MexIFace - A C++/Matlab object-based interface library and cross-platform CMake build system for creating interactive Matlab MEX modules with persistent state and complex behavior*. 2019. Disponível em: <https://github.com/markjolah/MexIFace>. Acesso em: 18 de jul. de 2024.

OLYMPUS. *Introduction to Phased Array Ultrasonic Technology Applications: R/D Tech Guideline*. Waltham, MA, USA: Olympus NDT, 2004.

OLYMPUS. *Phased Array Testing Basic Theory for Industrial Applications*. Waltham, MA, USA: Olympus NDT, 2010.

OREOMAN. *Simpler MEX Multi-Threading w/ a Persistent Thread Pool*. 2024. Disponível em: <https://www.mathworks.com/matlabcentral/fileexchange/71532-simpler-mex-multi-threading-w-a-persistent-thread-pool>. Acesso em: 26 de jul. de 2024.

PRæSIUS, S. K. et al. Real-time super-resolution ultrasound imaging using gpu acceleration. In: IEEE. *2022 IEEE International Ultrasonics Symposium (IUS)*. Piscataway, New Jersey, USA, 2022. p. 1–4. doi: 10.1109/IUS54386.2022.9957589.

RASMUSSEN, M. F.; JENSEN, J. A. Comparison of 3-d synthetic aperture phased-array ultrasound imaging and parallel beamforming. *IEEE Transactions on Ultrasonics, Ferroelectrics, and Frequency Control*, v. 61, n. 10, p. 1638–1650, 2014. doi: 10.1109/TUFFC.2014.006345.

ROMERO, D. et al. Using gpus for beamforming acceleration on saft imaging. In: IEEE. *2009 IEEE International Ultrasonics Symposium*. Piscataway, New Jersey, USA, 2009. p. 1334–1337. doi: 10.1109/ULTSYM.2009.5441790.

ROMERO-LAORDEN, D. et al. Paralelización de los procesos de conformación de haz para la implementación del total focusing method. In: *12 Congreso Nacional de Ensayos No Destructivos Valencia*. Valencia, ES: Asociación Española de Ensayos No Destructivos, 2011. p. 1–10.

SLOUN, R. J. van et al. Deep learning for super-resolution vascular ultrasound imaging. In: IEEE. *ICASSP 2019 - 2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. Piscataway, New Jersey, USA, 2019. p. 1055–1059. doi: 10.1109/ICASSP.2019.8683813.

YANG, C.; QIN, K.; LI, Y. Real-time ultrasonic imaging for multi-layered objects with synthetic aperture focusing technique. In: IEEE. *2014 IEEE International Instrumentation and Measurement Technology Conference (I2MTC) Proceedings*. Piscataway, New Jersey, USA, 2014. p. 561–566. doi: 10.1109/I2MTC.2014.6860807.

YIU, B. Y. S.; TSANG, I. K. H.; YU, A. C. H. Gpu-based beamformer: Fast realization of plane wave compounding and synthetic aperture imaging. *IEEE Transactions on Ultrasonics, Ferroelectrics, and Frequency Control*, v. 58, n. 8, p. 1698–1705, 2011. doi: 10.1109/TUFFC.2011.1999.

ZHANG, J.; DRINKWATER, B. W.; WILCOX, P. D. Comparison of ultrasonic array imaging algorithms for nondestructive evaluation. *IEEE Transactions on Ultrasonics, Ferroelectrics, and Frequency Control*, v. 60, n. 8, p. 1732–1745, 2013. doi: 10.1109/TUFFC.2013.2754.

Apêndices

Apêndice A – Implementações em Matlab

A.1 Método 2R-SAFT

O método 2R-SAFT é descrito pelo Algoritmo 10. A base apresentada para as abordagens SAFT e SAFT-TFM é mantida, alterando somente a dinâmica dos elementos emissores e receptores, considerando que nesta abordagem, a cada ponto calculado, um elemento emite e dois recebem.

Algoritmo 10 – 2R-SAFT

```
for z1 = 1 : Nz do
    for x1 = 1 : Nx do
        sinal = 0;
        for Ne = 1 : emissor do
            De = sqrt((x(x1) − xn(Ne))^2 + z(z1)^2);
            if Ne == emissor then                ▷ Verifica se é o último elemento emissor
                Dr = De + sqrt((x(x1) − xn(Nr))^2 + z(z1)^2);
                k = round(De/resolucao);
                indice = k − tbegin − 1;
                sinal = sinal + Data(Nr, indice, Ne);
            else
                for Nr = Ne : (Ne + 1) do        ▷ Varia os 2 receptores (2R-SAFT)
                    Dr = De + sqrt((x(x1) − xn(Nr))^2 + z(z1)^2);
                    k = round(De/resolucao);
                    indice = k − tbegin − 1;
                    sinal = sinal + Data(Nr, indice, Ne);
                end for
            end if
        end for
        pontos(z1,x1) = sinal;
    end for
end for
```

A.2 Método SAFT-TFM otimizado

A metodologia adotada para a otimização do método SAFT-TFM é descrito no Algoritmo 11. A redução dos cálculos é feita no laço que varia os elementos do *array* x, onde o índice superior é apenas a metade do número de pontos, e não o total. Neste algoritmo, a primeira metade da matriz de pontos é efetivamente calculada, e a segunda metade, por sua vez, é simplesmente atribuída com os valores já obtidos, por meio da simetria. A principal vantagem observada é a redução no tempo de execução, que não é proporcional ao número de iterações reduzidas. Esse resultado se deve principalmente ao acesso duplo à matriz de dados *Data*.

Algoritmo 11 – Otimização para o SAFT-TFM

for $z_1 = 1 : N_z$ **do**
  for $x_1 = 1 :$ round$(N_x/2)$ **do** ▷ Aqui o *array* x é dividido ao meio
    sinal = 0;
    sinal_sim = 0; ▷ Variável para sinal simétrico
    for $N_r = 1 :$ receptor **do**
      $D_r = \sqrt{(x(x_1) - x_n(N_r))^2 + z(z_1)^2}$;
      N_r_sim = length(x_n) - aux_1; ▷ Receptor simétrico
      for $N_e = 1 :$ emissor **do**
        N_e_sim = length(x_n) - aux_2; ▷ Emissor simétrico
        $D_e = D_r + \sqrt{(x(x_1) - x_n(N_e))^2 + z(z_1)^2}$;
        k = round(D_e/resolucao);
        indice = $k - t_{begin} - 1$;
        sinal = sinal + Data(N_r, indice, N_e);
        sinal_sim = sinal_sim + Data(N_r_sim, indice, N_e_sim);
        $aux_2 = aux_2 + 1$; ▷ Operação para obtenção do emissor simétrico
      end for
      $aux_2 = 0$; ▷ Elemento 128 como simétrico para a próxima iteração
      $aux_1 = aux_1 + 1$; ▷ Operação para obtenção do receptor simétrico
    end for
    $aux_1 = 0$;
    pontos(z_1,x_1) = sinal;
    pontos(z_1,$N_x - x_1 + 1$) = sinal_sim; ▷ Armazena na 2ª metade da matriz
  end for
end for

A.3 Implementação concorrente em Matlab com o método SAFT-TFM

A implementação concorrente utilizando a *Parallel Computing Toolbox* é apresentada pelo Algoritmo 12. Uma das principais diferenças é a divisão da matriz de dados em duas matrizes. Foi obtido um ganho no tempo de execução, sendo justificado pela diminuição da escalabilidade, ou utilização da memória RAM *dual channel*, detalhe que não foi possível confirmar. A paralelização seguiu as práticas descritas como, por exemplo, a utilização do *parfor* somente no laço mais externo, a definição dos limites superiores por meio de variáveis e não funções, e a utilização de duas matrizes *pontos*, devido à limitação de indexação que o laço paralelizado impõe.

Algoritmo 12 – Implementação com *parfor*

```
parfor z1 = 1 : Nz do                                   ▷ Laço paralelizado com parfor
    za = z(z1)^2;                                       ▷ Essa distância é fixada
    x_local = x;                                        ▷ Variável broadcast
    xn_local = xn;                                      ▷ Variável broadcast
    aux1 = 0; aux2 = 0;
    for x1 = 1 : end_x do
      sig1 = 0;                                         ▷ Referente à 1ª metade da matriz Data
      sig2 = 0;                                         ▷ Referente à 2ª metade da matriz Data
      sig1_sim = 0;
      sig2_sim = 0;
      for Nr = 1 : receptor do
        Dr = sqrt((x_local(x1) - xn_local(Nr))^2 + za);
        Nr_sim = 64 - aux2;                             ▷ É considerado 64 devido à divisão da matriz
        for Ne = 1 : emissor do
          Ne_sim = length(xn) - aux2;
          De = Dr + sqrt((x_local(x1) - xn_local(Ne))^2 + za);
          k = round(De/resolucao);
          indice = k - t_begin - 1;
          sig1 = sig1 + Data1(Nr, indice, Ne);
          sig1_sim = sig1_sim + Data2(Nr_sim, indice, Ne_sim);
          sig2 = sig2 + Data2(Nr, indice, Ne);
          sig2_sim = sig2_sim + Data1(Nr_sim, indice, Ne_sim);
          aux1 = aux1 + 1;
        end for
        aux1 = 0;
        aux2 = aux2 + 1;
      end for
      aux2 = 0;
      pontos1(z1,x1) = sig1 + sig2;                     ▷ 2 matrizes devido à indexação em um parfor
      pontos2(z1,x1) = sig1_sim + sig2_sim;             ▷ Somam-se os resultados
    end for
end parfor
```

A.4 Razão de aceleração e eficiência dos *workers*

Os gráficos para a apresentação do problema na inicialização dos *workers* foram obtidos por meio do Algoritmo 13. A cada iteração do laço *for* é feito um processamento com o número de *workers* inicializados, por meio da função *parpool*. Após o término, os tempos decorridos são coletados para a realização dos cálculos descritos, que relacionam o tempo de execução para a implementação sequencial e paralela. A função *delete(gcp)* fecha todos os *workers* inicializados, garantindo que a próxima iteração esteja nas mesmas condições anteriores.

Algoritmo 13 – Eficiência na inicialização dos *workers*

```
numWorkersArray = 1:4;                                  ▷ Número de workers
speedupArray = zeros(size(numWorkersArray));
efficiencyArray = zeros(size(numWorkersArray));
elapsedTimeSeq = 8.202;
for k = 1 : length(numWorkersArray) do      ▷ Varia o número de workers utilizado
    parpool('local', numWorkersArray(k));                ▷ Inicializa os workers
    tic;
    parfor z_1 = 1 : N_z do
        .
        .                                   ▷ Realiza-se o processamento com parfor
        .
    end parfor
    elapsedTimePar = toc;                                ▷ Coleta o tempo demandado
    delete(gcp);                                         ▷ Fecha os workers inicializados
    speedupArray(k) = elapsedTimeSeq / elapsedTimePar;
    efficiencyArray(k) = speedupArray(k) / numWorkersArray(k);
end for
...                                                      ▷ Geração das figuras
```

Apêndice B – Implementações em *C*

B.1 Implementação sequencial em *C* com o método SAFT-TFM

A implementação sequencial em linguagem *C* considerando o método SAFT-TFM é apresentada no Algoritmo 14. As principais diferenças são a mudança de sintaxe específica para a linguagem de programação, e a indexação dos dados como *array* linear, e não matriz 3D. O ganho no tempo de execução, dentre outros fatores, foi viabilizada por essa transformação da matriz em *array*. Por apresentar operações menos complexas, o tempo de acesso à memória foi encurtado, quando comparado à utilização da matriz 3D.

Algoritmo 14 – SAFT-TFM em *C*

```
for (z1 = 0; z1 < Nz; z1++) do
  for (x1 = 0; x1 < round(Nx/2); x1++) do
    sinal = 0;
    sinal_simetrico = 0;
    for (Nr = 0; Nr < dim1; Nr++) do
      Dr = sqrt((x(x1) - xn(Nr))^2 + z(z1)^2);
      Nr_sim = 127 - aux1;                    ▷ 127 pois o laço se inicia no índice 0
      for (Ne = 0; Ne < dim3; Ne++) do
        Ne_sim = 127 - aux2;
        De = Dr + sqrt((x(x1) - xn(Ne))^2 + z(z1)^2);
        k = round(De/resolucao);
        indice = k - t_begin - 1;
        sinal += Data[Ne + N · indice + N · dim2 · Nr];
        sinal_simetrico+ = Data[Ne_sim + N · indice + N · dim2 · Nr_sim];
        aux2++;
      end for
      aux2 = 0;
      aux1++;
    end for
    aux1 = 0;
    pontos[z1 + Nz · x1] = sinal;
    pontos[z1 + Nz · (Nx - 1 - x1)] = sinal_simetrico;
  end for
end for
```

B.2 Implementação concorrente em *C* com o método SAFT-TFM

A paralelização feita em linguagem *C* é descrita pelo Algoritmo 15. Na estrutura apresentada, a divisão de tarefas ocorre no laço mais externo z. O limite superior é definido como sendo o resultado da razão do número de pontos do *array* z pela quantidade de núcleos disponíveis (igual a 4, considerando a plataforma utilizada). Portanto, cada *core* da CPU executa um conjunto de instruções contidas no laço de z. É interessante destacar também que neste caso não existe alguns dos problemas descritos para a paralelização em Matlab, como a definição de limites superiores por meio de funções (utilização do *round* no laço de x), e a indexação da matriz de pontos, pois neste caso é utilizada somente uma variável '*pontos*'.

Algoritmo 15 – Implementação paralelizada em *C*

for $(z_1 = 0;\ z_1 < imax;\ z_1 + +)$ **do** ▷ Laço paralelizado. $imax = ceil(N_z/N_{cores})$
  for $(x_1 = 0;\ x_1 < round(N_x/2);\ x_1 + +)$ **do**
    sinal $= 0$;
    $sinal_{simetrico} = 0$;
    for $(N_r = 0;\ N_r < 128;\ N_r + +)$ **do**
      $D_r = \sqrt{(x(x_1) - x_n(N_r))^2 + z(z_1)^2}$;
      $Nr_{sim} = 127 - aux_1$;
      for $(N_e = 0;\ N_e < 128;\ N_e + +)$ **do**
        $Ne_{sim} = 127 - aux_2$;
        $D_e = D_r + \sqrt{(x(x_1) - x_n(N_e))^2 + z(z_1)^2}$;
        $k = \text{round}(D_e/\text{resolucao})$;
        $indice = k - t_{begin} - 1$;
        sinal += Data$[N_e + N \cdot indice + N \cdot dim_2 \cdot N_r]$;
        $sinal_{simetrico} += Data[Ne_{sim} + N \cdot indice + N \cdot dim_2 \cdot Nr_{sim}]$;
        $aux_2 + +$;
      end for
      $aux_2 = 0$;
      $aux_1 + +$;
    end for
    $aux_1 = 0$;
    $pontos[z_1 + N_z \cdot x_1] = sinal$;
    $pontos[z_1 + N_z \cdot (N_x - 1 - x_1)] = sinal_{simetrico}$;
  end for
end for

MINISTÉRIO DA EDUCAÇÃO
Universidade Federal de Ouro Preto
Instituto de Ciências Exatas e Aplicadas
Colegiado do Curso de Engenharia Elétrica

TERMO DE RESPONSABILIDADE

O texto do trabalho de conclusão de curso intitulado Estratégias para redução dos tempos de execução de algoritmos SAFT para formação de imagens acústicas usando processadores quad-core é de minha inteira responsabilidade. Declaro que não há utilização indevida de texto, material fotográfico ou qualquer outro material pertencente a terceiros sem a devida citação ou consentimento dos referidos autores.

João Monlevade, 19 de setembro de 2024.

gov.br Documento assinado digitalmente
MATEUS HENRIQUE GONCALVES
Data: 19/09/2024 15:23:52-0300
Verifique em https://validar.iti.gov.br

Mateus Henrique Gonçalves