



UFAM

UNIVERSIDADE FEDERAL DO AMAZONAS
PRÓ-REITORIA DE PESQUISA E PÓS-GRADUAÇÃO
DEPARTAMENTO DE PESQUISA

PROGRAMA INSTITUCIONAL DE BOLSAS DE INICIAÇÃO CIENTÍFICA - PIBIC
PROGRAMA DE APOIO À INICIAÇÃO CIENTÍFICA - PAIC

RELATÓRIO FINAL

EDIÇÃO: PIBIC/PAIC 2023/2024	
RECURSOS HUMANOS	
Nome do(a) orientador(a): JOÃO MARCOS BASTOS CAVALCANTI	
Nome do(a) aluno(a) SEBASTIAO BICHARRA NETO	Bolsa: () CNPQ () UFAM () FAPEAM (X) VOLUNTÁRIO
IDENTIFICAÇÃO DO PROJETO	
Título: Análise comparativa de desempenho entre algoritmos paralelos e tradicionais para processamento de imagens.	Código do Projeto: PIB-E/0141/2023
Área de Conhecimento: (X) Exatas e da Terra () Agrárias () Biológicas () Sociais Aplicadas () Engenharias () Saúde () Ciências Humanas () Linguística, Letras e Artes () Multidisciplinar	
COMITÊ DE ÉTICA EM PESQUISA COM HUMANOS (CEP) OU ANIMAIS (CEUA)	
() Aprovado - Número do protocolo: _____ (X) Não se aplica Caso o projeto ainda não esteja aprovado, justifique:	

* O Relatório deve ser apresentado abaixo deste formulário em no máximo 20 páginas

* O Relatório deverá estar de acordo com as normas atualizadas da ABNT para trabalhos acadêmicos.

RESUMO

Neste projeto, abordamos a análise comparativa do desempenho entre algoritmos paralelos e tradicionais no campo de processamento de imagens. A programação paralela permite a execução simultânea de processos em múltiplos núcleos ou processadores, é posta à prova contra técnicas de programação convencionais, onde as instruções são executadas em sequência. A pesquisa foca em aspectos como tempo de processamento, eficiência no uso de recursos e qualidade dos resultados em operações comuns de processamento de imagens, como transformações radiométricas, detecção de bordas e filtragem espacial. O estudo é estruturado em uma série de experimentos práticos, onde diferentes conjuntos de imagens são submetidos às operações usando algoritmos paralelos e convencionais. Isso permite uma avaliação direta e objetiva das vantagens e limitações de cada abordagem. A investigação não se limita apenas a medir o desempenho, mas também busca entender as implicações da complexidade de implementação e as possíveis aplicações práticas da programação paralela para processamento de imagens. Através desta análise comparativa, o projeto visa fornecer uma compreensão da programação paralela em relação aos métodos tradicionais, esclarecendo seu papel potencial no avanço da eficiência e eficácia do processamento de imagens. Este estudo é relevante no contexto atual, onde a demanda por processamento rápido e eficiente de grandes volumes de dados de imagens vem crescendo.

1. INTRODUÇÃO

Com a popularização do uso de processadores com vários núcleos, desde computadores servidores de alto desempenho até smartphones, a possibilidade do uso da programação paralela para a solução de diversos problemas tornou-se uma realidade.

A programação paralela caracteriza-se pelo uso de vários recursos computacionais para resolver um problema. Esses problemas são divididos em várias partes, em que cada parte é executada em simultâneo pelos vários processadores. A exploração do paralelismo na execução das instruções é a chave para a programação paralela. Em princípio, utilizar mais recursos para resolver um problema vai diminuir o seu tempo de execução [2, 4].



UNIVERSIDADE FEDERAL DO AMAZONAS
PRÓ-REITORIA DE PESQUISA E PÓS-GRADUAÇÃO
DEPARTAMENTO DE PESQUISA
PROGRAMA INSTITUCIONAL DE BOLSAS DE INICIAÇÃO CIENTÍFICA - PIBIC
PROGRAMA DE APOIO À INICIAÇÃO CIENTÍFICA - PAIC

Algoritmos de Processamento de Imagens em grande parte tratam do processamento de matrizes com valores numéricos, que representam a intensidade dos pixels da imagem [3]. Este tipo de estrutura de dados permite soluções por meio de algoritmos paralelos, facilitando a divisão de tarefas de processamento de imagens em várias partes menores, que são executadas simultaneamente em diferentes processadores. Desta forma, determinadas técnicas e operações de processamento de imagens podem ser aceleradas significativamente permitindo sua aplicação em tempo real ou em larga escala, dependendo do contexto. Por exemplo, em uma aplicação de reconhecimento facial em tempo real, o processamento de imagens pode ser dividido em várias tarefas menores, como detecção de rosto, detecção de olhos, identificação de características faciais, reconhecimento de expressões faciais, entre outras. Cada uma dessas tarefas (ou mesmo partes menores delas) pode ser executada em um núcleo de processamento diferente, permitindo que o sistema processe as imagens em tempo real com desempenho adequado à aplicação.

Embora as arquiteturas heterogêneas, incluindo o uso de Unidades de Processamento Gráfico (GPUs), ofereçam potencias vantagens para o processamento de imagens através da execução paralela de tarefas, não dependendo somente da Unidade de Processamento Central (CPU), este projeto não se concentra em explorar tais arquiteturas. Em vez disso, focamos em demonstrar experimentalmente as vantagens da programação paralela e como sua implementação pode oferecer melhorias significativas em eficiência e velocidade de processamento, mesmo quando restrita a ambientes computacionais baseados somente em CPUs, no contexto das operações sobre imagens: transformadas radiométricas, filtragens espaciais e detecção de bordas.

Partindo da hipótese de que a programação paralela pode ser aplicada em técnicas de processamento de imagens com uma melhoria significativa no desempenho, propomos neste trabalho uma investigação empírica por meio de experimentos com algoritmos em suas versões paralela e tradicional (também chamado de sequencial ou convencional). O objetivo é explorar até que ponto a programação paralela pode superar os métodos convencionais em aspectos como tempo de processamento, eficiência no uso de recursos e qualidade dos resultados finais.

2. OBJETIVOS (geral e específicos)

2.1 Geral:

- Realizar pesquisa experimental sobre o desempenho de algoritmos tradicionais e paralelos no processamento de imagens, investigando sua aplicabilidade a fim de mapear resultados que orientem a escolha de técnicas e ferramentas para o melhoramento e aumento de eficiência de processamento de imagens.

2.2 Específicos:

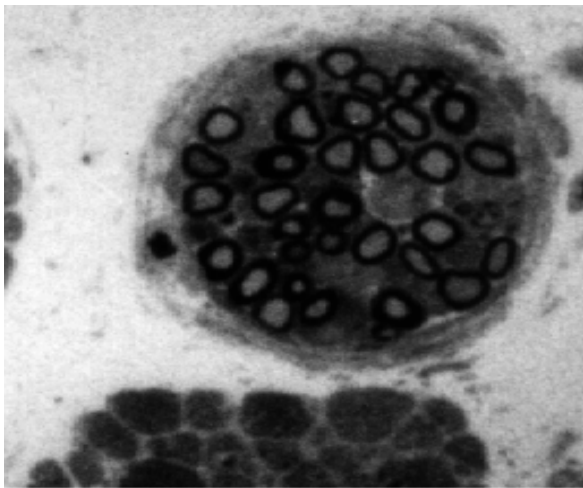
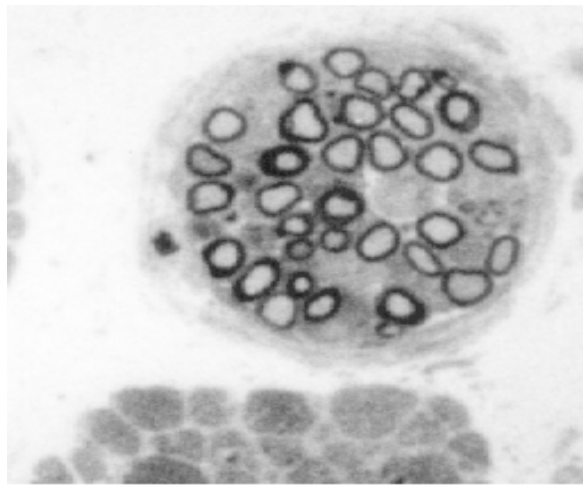


- Levantamento bibliográfico das técnicas mais recentes do uso de programação paralela em processamento de imagens;
- Coletar dados e informações sobre o desempenho dos algoritmos em diferentes cenários e situações;
- Analisar o desempenho dos algoritmos em termos de tempo de resposta e uso de recursos de hardware;
- Identificar algumas questões como facilidade de uso, ferramentas disponíveis, de cada tipo de algoritmo e avaliar a sua aplicabilidade em diferentes problemas de processamento de imagens.

3. METODOLOGIA

Este projeto foi conduzido com o objetivo de realizar uma análise comparativa do desempenho entre algoritmos tradicionais e paralelos no processamento de imagens. Infelizmente, projetar cronogramas de alto desempenho para pipelines complexos de processamento de imagens requer conhecimento substancial de arquitetura de hardware moderna e técnicas de otimização de código [1]. Problemas que foram sanados com a utilização de ferramentas modernas para processamento de imagens e programação paralela que será tratado mais adiante. A pesquisa baseia-se em um método quantitativo de análise que levou em consideração o tempo de resposta e de processamento, com análises adicionais sobre uso de memória e quantidade de núcleos, baseando-se na execução dos seguintes algoritmos selecionados, típicos em processamento de imagens:

1. Transformadas Radiométricas: Expansão de contraste linear, Logaritmo e Dente de serra.
2. Filtragens Espaciais: Filtro da Média, Mediana e K vizinhos mais próximos.
3. Detecção de bordas: Operador Sobel e Operador Roberts.

A tabela 1 ilustra o funcionamento e efeitos das técnicas aplicadas sobre algumas imagens selecionadas no projeto:

Transformada Radiométrica	
Transformada do Logaritmo	
Imagem Original	Imagem Processada
	
Expansão de Contraste Linear	
Imagem Original	Imagem Processada
	



UFAM

UNIVERSIDADE FEDERAL DO AMAZONAS
PRÓ-REITORIA DE PESQUISA E PÓS-GRADUAÇÃO
DEPARTAMENTO DE PESQUISA
PROGRAMA INSTITUCIONAL DE BOLSAS DE INICIAÇÃO CIENTÍFICA - PIBIC
PROGRAMA DE APOIO À INICIAÇÃO CIENTÍFICA - PAIC





Filtragem Espacial	
Filtro da Mediana	
	
Detecção de Bordas	
Operador Sobel	
Imagem Original	Imagem Processada
	

Tabela 1 - Exemplos de operações sobre imagens utilizadas no projeto



UFAM

UNIVERSIDADE FEDERAL DO AMAZONAS
PRÓ-REITORIA DE PESQUISA E PÓS-GRADUAÇÃO
DEPARTAMENTO DE PESQUISA

PROGRAMA INSTITUCIONAL DE BOLSAS DE INICIAÇÃO CIENTÍFICA - PIBIC
PROGRAMA DE APOIO À INICIAÇÃO CIENTÍFICA - PAIC

Os algoritmos de transformadas radiométricas caracterizam-se por ser uma operação ponto a ponto, ou seja, percorre-se a matriz de pixels de uma imagem e a função de transformação é aplicada em cada pixel sem considerar o valor dos pixels vizinhos. As operações de filtragem espacial e detecção de bordas caracterizam-se pela aplicação de uma máscara de convolução, que representa o filtro ou a operação para detectar as bordas. Neste caso, percorre-se a matriz de pixels, mas a operação considera pixels vizinhos na aplicação da operação, tendo naturalmente um custo de processamento maior do que as operações de transformações radiométricas.

Para realizar os testes dos algoritmos e verificar o ganho de desempenho em suas versões tradicionais e paralelas foram selecionadas um conjunto de trinta imagens diversificando o tamanho entre altura e largura (quantidade de pixels) que foram divididos em seis grupos contendo cinco imagens. O primeiro grupo possui entre 1.800.000 a 1.900.000 pixels, o segundo grupo entre 2.000.000 a 4.000.000 pixels, o terceiro grupo entre 4.000.000 a 8.000.000 pixels, o quarto grupo entre 9.000.000 a 12.200.000 pixels, o quinto grupo entre 12.200.000 a 19.000.000 pixels e o sexto grupo entre 22.000.000 a 33.000.000 pixels.

Para implementar os algoritmos foram utilizados três linguagens de programação, sendo duas para versões tradicionais, Python e C com suporte da biblioteca OpenCV para processamento de imagens. Para a programação paralela utilizamos a linguagem Halide. A linguagem Halide provou ser um sistema eficaz para autoria de código de processamento de imagens de alto desempenho. Os programadores Halide precisam apenas fornecer uma estratégia de alto nível para mapear um pipeline de processamento de imagem para uma máquina paralela (um cronograma), e o compilador Halide executa a tarefa de gerar código específico da plataforma que implementa o cronograma. Podemos observar as diferenças nas implementações quando analisamos trechos de códigos desenvolvidos, conforme as Figuras 2, 3 e 4 a seguir.

```
Var x("x"), y("y"), c("c");
Func transformada_dente_de_serra("transformada_dente_de_serra");

Expr valor = cast<float>(input(x, y, c));
Expr resultado = select(
    valor < 63, cast<uint8_t>(255 * valor / 62),
    valor < 127, cast<uint8_t>(255 * (valor - 63) / 63),
    valor < 191, cast<uint8_t>(255 * (valor - 127) / 63),
    cast<uint8_t>(255 * (valor - 191) / 64)
);

transformada_dente_de_serra(x, y, c) = cast<uint8_t>(resultado);
transformada_dente_de_serra.parallel(y).vectorize(x, 16);

Buffer<uint8_t> output = transformada_dente_de_serra.realize({ input.width(), input.height(), input.channels() });
```

Figura 2 - Transformada Dente de Serra na linguagem Halide

A Figura 1 apresenta um trecho de código implementado na linguagem de programação paralela Halide, em específico a transformada radiométrica dente de serra. As variáveis (Var x("x"), y("y") e c("c")) representam as dimensões da imagem: x e y para as coordenadas dos pixels, e c para os canais de cor (ex.: RGB). A função (Func transformada_dente_de_serra("transformada_dente_de_serra")) define a função Halide que aplicará a transformação dente de serra na imagem. A expressão de valor (Expr valor = cast<float>(input(x, y, c))) captura o valor do pixel em cada posição (x, y, c) e o converte para float para garantir precisão nas operações subsequentes. A expressão resultado (Expr resultado = select(...)) aplica a transformação dente de serra baseada no valor do pixel, criando uma expressão condicional que divide o intervalo de valores dos pixels em segmentos, aplicando diferentes fórmulas a cada segmento. O Paralelismo ocorre em (transformada_dente_de_serra.parallel(y).vectorize(x, 16)) a linha parallel(y) paraleliza o processamento ao longo da dimensão y, ou seja, diferentes linhas da imagem são processadas simultaneamente em diferentes núcleos da CPU. O método vectorize(x, 16) usa vetorização, que otimiza o processamento de blocos de 16 pixels ao longo da dimensão x, após esta etapa “realize” (na última linha) recebe um vetor com as dimensões da imagem de saída — largura, altura e o número de canais, é chamada e executa o pipeline para gerar a imagem de saída.

As versões tradicionais por não possuírem suporte para programação paralela utilizam técnicas diferentes para implementar a mesma operação de transformação radiométrica apresentada acima. Para tanto, em ambas as implementações tradicionais a abordagem baseia-se na iteração pelos pixels através de estruturas de repetições como “loops for” percorrendo cada pixel da imagem, processando

cada um individualmente. Podemos observar essa abordagem nos trechos de códigos apresentados nas Figuras 3 e 4, respectivamente nas linguagens Python e C.

```
def transformada_dente_de_serra(imagem):  
    altura, largura = imagem.shape  
    imagem_transformada = np.zeros_like(imagem, dtype=np.float32)  
  
    for i in range(altura):  
        for j in range(largura):  
            pixel = imagem[i][j]  
  
            if pixel < 63:  
                imagem_transformada[i][j] = int(255 * pixel / 62)  
  
            elif 63 <= pixel < 127:  
                imagem_transformada[i][j] = int(255 * (pixel - 63) / 63)  
  
            elif 127 <= pixel < 191:  
                imagem_transformada[i][j] = int(255 * (pixel - 127) / 63)  
  
            else:  
                imagem_transformada[i][j] = int(255 * (pixel - 191) / 64)  
  
    return np.clip(imagem_transformada, 0, 255).astype(np.uint8)
```

Figura 3 - Transformada Dente de Serra na linguagem Python.

```
void transformada_dente_de_serra(Mat imagem, Mat& imagem_transformada) {  
    int altura = imagem.rows;  
    int largura = imagem.cols;  
  
    imagem_transformada = Mat::zeros(altura, largura, CV_32F);  
  
    for (int i = 0; i < altura; i++) {  
        for (int j = 0; j < largura; j++) {  
            int pixel = imagem.at<uchar>(i, j);  
  
            if (pixel < 63) {  
                imagem_transformada.at<float>(i, j) = 255.0 * pixel / 62.0;  
            } else if (pixel >= 63 && pixel < 127) {  
                imagem_transformada.at<float>(i, j) = 255.0 * (pixel - 63) / 63.0;  
            } else if (pixel >= 127 && pixel < 191) {  
                imagem_transformada.at<float>(i, j) = 255.0 * (pixel - 127) / 63.0;  
            } else {  
                imagem_transformada.at<float>(i, j) = 255.0 * (pixel - 191) / 64.0;  
            }  
        }  
    }  
  
    imagem_transformada.convertTo(imagem_transformada, CV_8U, 1.0, 0.0);  
}
```

Figura 4 - Transformada Dente de Serra na linguagem C

Esse tipo de implementação pode ser mais custosa em relação ao tempo de processamento quando comparada com técnicas de programação paralela que divide um problema em várias partes menores que serão executadas simultaneamente visando melhor performance.

Para verificar o desempenho dos algoritmos sequenciais e paralelos das três técnicas selecionadas (transformadas, filtragens e detecção de bordas) aplicamos um procedimento simples de medida do tempo de execução de cada operação sobre uma imagem.

No total foram realizadas 30 medidas de tempo de execução para cada algoritmo em imagens distintas. Essas execuções ocorreram em duas máquinas com configurações diferentes, que incluem a capacidade de processamento levando em consideração a quantidade de núcleos disponíveis em cada uma. Sendo a primeira máquina disponibilizada pelo Instituto de Computação (IComp/UFAM) e que possui maior quantidade de núcleos, e a segunda é a máquina do bolsista do projeto, que possui uma menor quantidade de núcleos.

Após finalizar as execuções de todos os algoritmos selecionados os dados coletados foram organizados em uma tabela de forma que foi possível identificar cada tempo de processamento que levou para que uma determinada transformação fosse aplicada em uma imagem, sendo esses valores todos em milissegundos. Diante dos trinta valores de tempo de execução por cada algoritmo foi realizado uma média aritmética para verificar a média geral após as 30 execuções. Essas médias encontradas serviram como parâmetro para verificar o ganho de desempenho entre as versões tradicionais e paralelas de uma forma que dividimos o valor da média da versão tradicional pelo valor da média da versão em paralelo para verificarmos qual foi o ganho estabelecido.

Diante de todos esses procedimentos estabelecidos chegamos a valores que fosse possível realizar uma comparação justa entre as versões tradicionais e paralelas dos algoritmos aplicados em processamento de imagens, levando em consideração a peculiaridade das abordagens estabelecidas ao longo do processo, sendo esses fatores e resultados discutidos na próxima sessão.

Como forma de resumir a metodologia apresentamos a seguir um passo-a-passo que seguimos para realização dos experimentos:

1. Montagens de base de imagens com 30 imagens de tamanhos variados.

-
- Seleccionamos um conjunto de 30 imagens de tamanhos variados para a realização dos experimentos. Essas imagens foram organizadas em seis grupos, categorizados de acordo com a quantidade de pixels. O primeiro grupo possui entre 1.800.000 a 1.900.000 pixels, o segundo grupo entre 2.000.000 a 4.000.000 pixels, o terceiro grupo entre 4.000.000 a 8.000.000 pixels, o quarto grupo entre 9.000.000 a 12.200.000 pixels, o quinto grupo entre 12.200.000 a 19.000.000 pixels e o sexto grupo entre 22.000.000 a 33.000.000 pixels.
 2. Execução das operações e medida de tempo de processamento para cada uma das 30 imagens.
 - Aplicamos as operações de transformações radiométricas, filtragens espaciais, e detecção de bordas em cada uma das 30 imagens. O tempo de processamento foi medido para cada imagem e operação, tanto nas versões tradicionais (Python e C) quanto na versão paralela (Halide).
 3. Cálculo da média e ganho de desempenho da versão paralela com a versão tradicional em cada operação.
 - Para cada operação e conjunto de imagens, calculamos a média dos tempos de processamento após 30 execuções. Com essas médias, determinamos o ganho de desempenho da versão paralela em comparação com as versões tradicionais, utilizando a fórmula:
Ganho de desempenho: Média do tempo Tradicional/Média do tempo Paralelo
 - Esse cálculo nos permitiu quantificar as melhorias obtidas com a paralelização em termos de tempo de processamento.

4. RESULTADO e DISCUSSÃO

Nesta seção, serão apresentados os resultados obtidos a partir dos experimentos conduzidos para avaliar o desempenho dos algoritmos de processamento de imagens implementados em Python, C e Halide.

Os testes foram realizados em duas configurações de hardware distintas. Primeiro em máquinas do Instituto de Computação (ICOMP) do modelo 183 - 13th Gen Intel(R) Core(TM) i7-13700, com clock máximo de 4100,0000 MHz, possuindo 16 núcleos por soquetes e 2 Thread(s) por núcleo, isso totaliza 32 Thread(s) lógicas disponíveis, levando em consideração que Thread(s) lógicas disponíveis é igual ao número de Núcleo(s) por soquet x número de Thread(s) por núcleo. Isso significa que uma



UFAM

UNIVERSIDADE FEDERAL DO AMAZONAS
PRÓ-REITORIA DE PESQUISA E PÓS-GRADUAÇÃO
DEPARTAMENTO DE PESQUISA

PROGRAMA INSTITUCIONAL DE BOLSAS DE INICIAÇÃO CIENTÍFICA - PIBIC
PROGRAMA DE APOIO À INICIAÇÃO CIENTÍFICA - PAIC

tarefa pode ser dividida em até 32 partes que podem ser executadas simultaneamente utilizando a paralelização. Seguindo esses parâmetros, apresentamos os resultados nas tabelas a seguir.

Halide x Python		
Transformadas Radiométricas		
Técnica	Média geral em milissegundos após 30 execuções	Ganho em desempenho
Expansão de contraste linear (Halide)	52.2667	Halide foi 200 vezes mais rápido que Python.
Expansão de contraste linear (Python)	10540.8364	
Logaritmo (Halide)	57.9667	Halide foi 131 vezes mais rápido que Python.
Logaritmo (Python)	7649.6159	
Dente de serra (Halide)	54.6000	Halide foi 423 vezes mais rápido que Python.
Dente de serra (Python)	23112.5866	
Filtragens Espaciais		
Técnica	Média geral em milissegundos após 30 execuções	Ganho em desempenho
Média (Halide)	108.4333	Halide foi 822 vezes mais rápido que Python.
Média (Python)	89158.4204	
Mediana (Halide)	167.2000	Halide foi 1178 vezes mais rápido que Python.
Mediana (Python)	197033.2473	
K vizinhos mais próximos (Halide)	256.9000	Halide foi 681 vezes mais rápido que Python.
K vizinhos mais próximos (Python)	175001.3283	
Detecção de bordas		
Técnica	Média geral em milissegundos após 30 execuções	Ganho em desempenho
Sobel (Halide)	260.1667	Halide foi 630 vezes mais rápido que Python.

Sobel (Python)	163969.2557	
Roberts (Halide)	234.8667	Halide foi 685 vezes mais rápido que Python.
Roberts (Python)	161012.9039	

Tabela 2 - Comparação entre as implementações em Python e Halide (máquina do IComp)

A Tabela 2 apresenta os tempos de execução das operações implementadas em Python e Halide na máquina do IComp. Os dados demonstram que a implementação em Halide foi significativamente mais rápida do que a em Python para todas as operações. Por exemplo, na operação de Expansão de Contraste Linear, o tempo médio de execução utilizando Halide foi de 52,27 milissegundos, enquanto o mesmo processamento em Python levou 10.540,84 milissegundos, resultando em um ganho de desempenho de aproximadamente 200 vezes. Esse padrão de superioridade de Halide se repetiu em outras operações, como o filtro da mediana, onde Halide foi 1178 vezes mais rápido que Python, e detecção de bordas utilizando o operador Roberts, com Halide 685 vezes mais rápido que Python.

Halide x C		
Transformadas Radiométricas		
Técnica	Média geral em milissegundos após 30 execuções	Ganho em desempenho
Expansão de contraste linear (Halide)	52.2667	Halide foi 2 vezes mais rápido que C.
Expansão de contraste linear (C)	108.4604	
Logaritmo (Halide)	57.9667	Halide foi 2 vezes mais rápido que C.
Logaritmo (C)	154.5248	
Dente de serra (Halide)	54.6000	Halide foi 2 vezes mais rápido que C.
Dente de serra (C)	121.7629	
Filtragens Espaciais		
Técnica	Média geral em milissegundos após 30 execuções	Ganho em desempenho
Média (Halide)	108.4333	Halide foi 10 vezes mais rápido que C.
Média (C)	1175.0984	



UFAM

UNIVERSIDADE FEDERAL DO AMAZONAS
PRÓ-REITORIA DE PESQUISA E PÓS-GRADUAÇÃO
DEPARTAMENTO DE PESQUISA

PROGRAMA INSTITUCIONAL DE BOLSAS DE INICIAÇÃO CIENTÍFICA - PIBIC
PROGRAMA DE APOIO À INICIAÇÃO CIENTÍFICA - PAIC

Mediana (Halide)	167.2000	Halide foi 122 vezes mais rápido que C.
Mediana (C)	20485.0092	
K vizinhos mais próximos (Halide)	256.9000	Halide foi 166 vezes mais rápido que C.
K vizinhos mais próximos (C)	42856.6879	
Detecção de bordas		
Técnica	Média geral em milissegundos após 30 execuções	Ganho em desempenho
Sobel (Halide)	260.1667	Halide foi 5 vezes mais rápido que C.
Sobel (C)	1367.5888	
Roberts (Halide)	234.8667	Halide foi 4 vezes mais rápido que C.
Roberts (C)	1048.4216	

Tabela 3 - Comparação entre as implementações em C e Halide (máquina do IComp)

Similarmente, a Tabela 3 compara os tempos de execução entre as implementações em C e Halide na mesma máquina. Embora o desempenho de Halide ainda seja superior, a diferença foi menos pronunciada do que na comparação com Python. Por exemplo, na operação de Expansão de Contraste Linear, Halide foi aproximadamente 2 vezes mais rápido que C, com tempos médios de 52,27 milissegundos contra 108,46 milissegundos. Esse resultado sugere que, embora C seja uma linguagem de alto desempenho, Halide pode otimizar ainda mais o processamento paralelo, observando que em operações complexas como filtragens espaciais a linguagem teve resultados significativos, como no caso do filtro da mediana sendo Halide 122 vezes mais rápido que C.

Em segundo, vem a máquina do desenvolvedor do modelo 142 - Intel(R) Core(TM) i5-7200U, com clock máximo de 2.50GHz, possuindo 2 núcleos por soquetes e 2 Thread(s) por núcleo, isso totaliza 4 Thread(s) lógicas disponíveis. Isso significa que uma tarefa pode ser dividida em até 4 partes que podem ser executadas simultaneamente utilizando a paralelização. Seguindo esses parâmetros foi obtido os seguintes resultados:



UFAM

UNIVERSIDADE FEDERAL DO AMAZONAS
PRÓ-REITORIA DE PESQUISA E PÓS-GRADUAÇÃO
DEPARTAMENTO DE PESQUISA

PROGRAMA INSTITUCIONAL DE BOLSAS DE INICIAÇÃO CIENTÍFICA - PIBIC
PROGRAMA DE APOIO À INICIAÇÃO CIENTÍFICA - PAIC

Halide x Python		
Transformadas Radiométricas		
Técnica	Média geral em milissegundos após 30 execuções	Ganho em desempenho
Expansão de contraste linear (Halide)	185.4667	Halide foi 290 vezes mais rápido que Python.
Expansão de contraste linear (Python)	54202.6059	
Logaritmo (Halide)	211.5000	Halide foi 190 vezes mais rápido que Python.
Logaritmo (Python)	40204.0658	
Dente de serra (Halide)	189.3667	Halide foi 629 vezes mais rápido que Python.
Dente de serra (Python)	119273.2424	
Filtragens Espaciais		
Técnica	Média geral em milissegundos após 30 execuções	Ganho em desempenho
Média (Halide)	442.1333	Halide foi 1019 vezes mais rápido que Python.
Média (Python)	450812.6250	
Mediana (Halide)	663.8667	Halide foi 1665 vezes mais rápido que Python.
Mediana (Python)	1105394.0205	
K vizinhos mais próximos (Halide)	1084.3000	Halide foi 847 vezes mais rápido que Python.
K vizinhos mais próximos (Python)	918534.6280	
Detecção de bordas		
Técnica	Média geral em milissegundos após 30 execuções	Ganho em desempenho
Sobel (Halide)	1230.2667	Halide foi 639 vezes mais rápido que Python.
Sobel (Python)	786337.9029	
Roberts (Halide)	1006.5000	Halide foi 810 vezes mais rápido que Python.
Roberts (Python)	816129.8633	

Tabela 4 - Comparação entre as implementações em Python e Halide (máquina do Desenvolvedor)



UFAM

UNIVERSIDADE FEDERAL DO AMAZONAS
PRÓ-REITORIA DE PESQUISA E PÓS-GRADUAÇÃO
DEPARTAMENTO DE PESQUISA

PROGRAMA INSTITUCIONAL DE BOLSAS DE INICIAÇÃO CIENTÍFICA - PIBIC
PROGRAMA DE APOIO À INICIAÇÃO CIENTÍFICA - PAIC

A Tabela 4 apresenta os resultados obtidos na máquina do desenvolvedor, que possui menos núcleos de processamento. Nesta configuração, o ganho de desempenho de Halide sobre Python foi ainda mais acentuado. Na operação de Filtro da Mediana, Halide foi 1665 vezes mais rápido que Python e na operação de Detecção de Bordas utilizando o operador Roberts, Halide foi 810 vezes mais rápido que Python, isso comprova a eficácia da ferramenta quando operada sobre técnicas complexas aplicadas em imagens. Este resultado destaca a eficácia de Halide em otimizar o uso de múltiplos núcleos, mesmo em hardware com menos capacidade de paralelização.

Halide x C		
Transformadas Radiométricas		
Técnica	Média geral em milissegundos após 30 execuções	Ganho em desempenho
Expansão de contraste linear (Halide)	185.4667	Halide foi 1 vezes mais rápido que C.
Expansão de contraste linear (C)	272.8832	
Logaritmo (Halide)	211.5000	Halide foi 1 vezes mais rápido que C.
Logaritmo (C)	421.8791	
Dente de serra (Halide)	189.3667	Halide foi 1 vezes mais rápido que C.
Dente de serra (C)	260.5805	
Filtragens Espaciais		
Técnica	Média geral em milissegundos após 30 execuções	Ganho em desempenho
Média (Halide)	442.1333	Halide foi 11 vezes mais rápido que C.
Média (C)	5138.1128	
Mediana (Halide)	663.8667	Halide foi 147 vezes mais rápido que C.
Mediana (C)	98179.7097	
K vizinhos mais próximos (Halide)	1084.3000	Halide foi 208 vezes mais rápido que C.
K vizinhos mais próximos (C)	225874.5325	

Detecção de bordas		
Técnica	Média geral em milissegundos após 30 execuções	Ganho em desempenho
Sobel (Halide)	1230.2667	Halide foi 4 vezes mais rápido que C.
Sobel (C)	5574.5200	
Roberts (Halide)	1006.5000	Halide foi 4 vezes mais rápido que C.
Roberts (C)	4750.9302	

Tabela 5 - Comparação entre as implementações em C e Halide (máquina do Desenvolvedor)

Por fim, a Tabela 5 mostra a comparação entre Halide e C na máquina do desenvolvedor. Embora os ganhos de Halide ainda sejam evidentes, eles são menos expressivos do que em hardware com maior capacidade de paralelismo. Por exemplo, na operação de Expansão de Contraste Linear, Halide apresentou um tempo médio de 185,47 milissegundos, enquanto C teve um tempo de 272,88 milissegundos, resultando em um ganho de desempenho de aproximadamente 1,47 vezes. Isso reforça a ideia de que o desempenho de Halide é altamente dependente da quantidade de núcleos disponíveis, o que deve ser considerado ao escolher a infraestrutura para processamento de imagens.

Os resultados demonstraram que a linguagem Halide oferece um ganho de desempenho substancial em relação às implementações tradicionais em Python e C, especialmente em máquinas com maior capacidade de processamento paralelo. Os maiores ganhos de desempenho da programação paralela em Halide foram observados nas operações de **filtragem espacial** (especialmente **Mediana** e **K Vizinhos Mais Próximos**) e na **detecção de bordas (Roberts e Sobel)**. Esses ganhos são evidentes tanto na comparação com Python quanto com C, mostrando que a paralelização traz benefícios claros para algoritmos que envolvem operações complexas e que podem ser facilmente distribuídas entre múltiplos núcleos de processamento. Para ficar mais evidente o ganho de desempenho podemos analisar os dados graficamente em uma escala logarítmica:

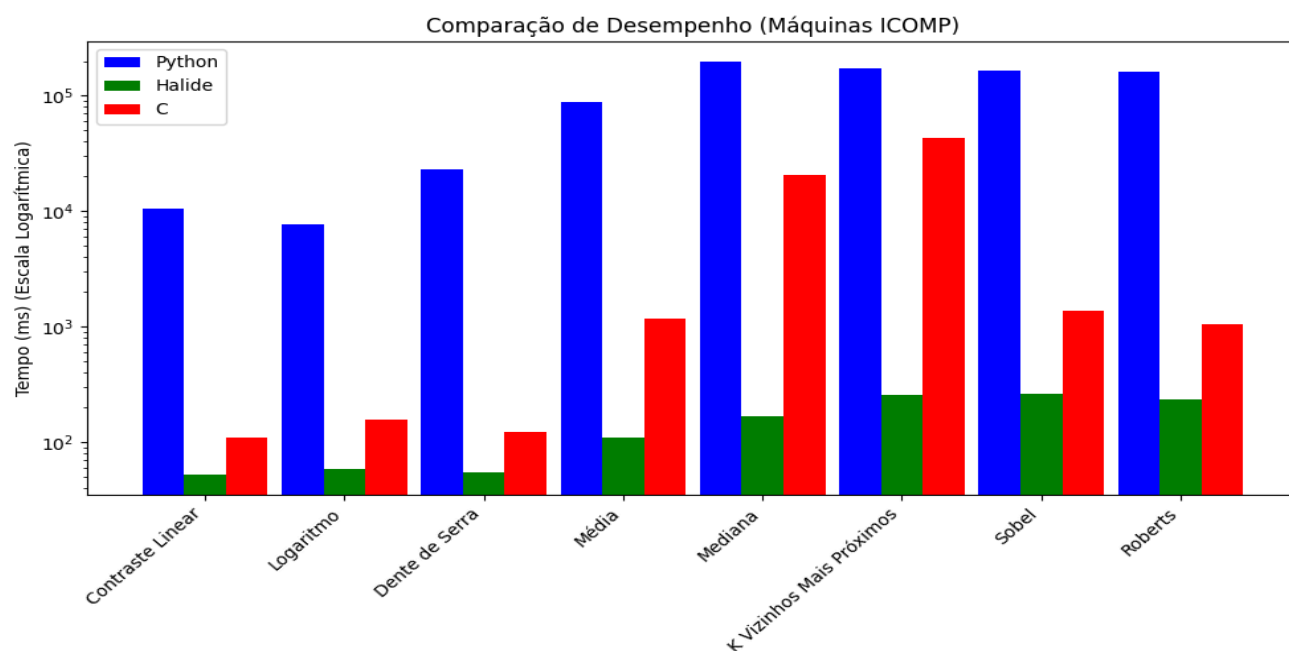


Figura 5 - Comparação de desempenho das operações sobre imagens (máquina do IComp)

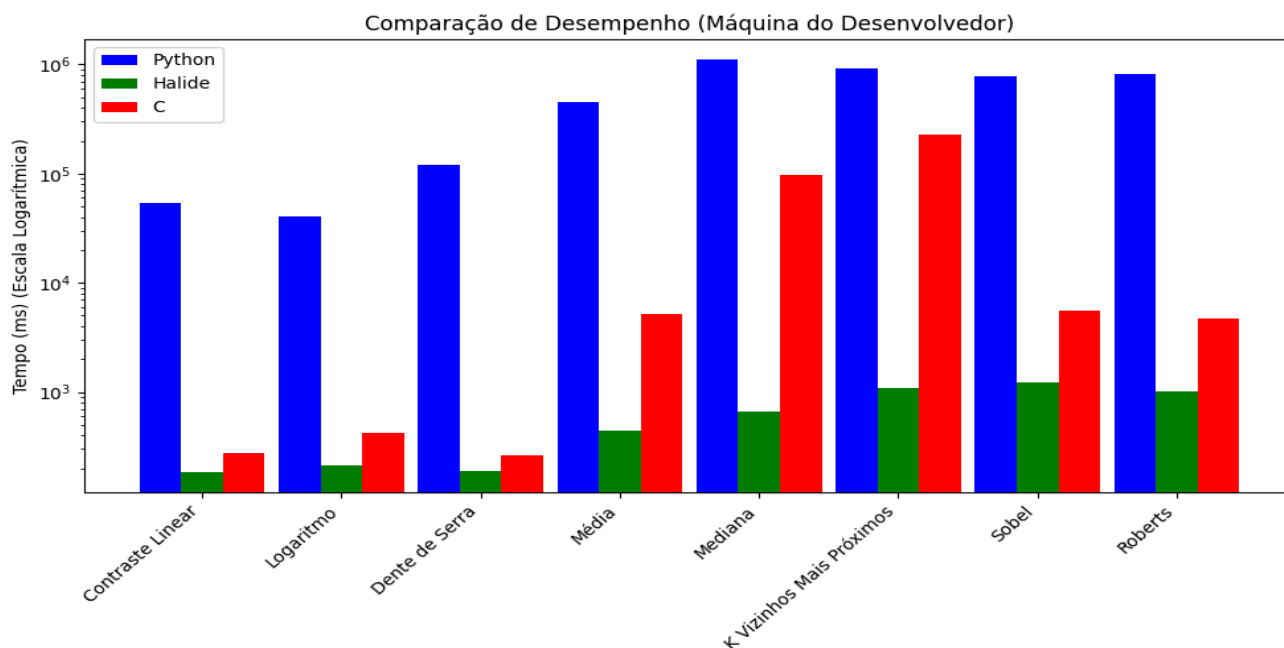


Figura 6 - Comparação de desempenho das operações sobre imagens (máquina do bolsista)

Esses resultados confirmam que a programação paralela, quando aplicada corretamente, pode reduzir significativamente os tempos de processamento, especialmente em tarefas que lidam com grandes volumes de dados ou operações intensivas em processamento, como no caso do processamento de



UFAM

UNIVERSIDADE FEDERAL DO AMAZONAS
PRÓ-REITORIA DE PESQUISA E PÓS-GRADUAÇÃO
DEPARTAMENTO DE PESQUISA

PROGRAMA INSTITUCIONAL DE BOLSAS DE INICIAÇÃO CIENTÍFICA - PIBIC
PROGRAMA DE APOIO À INICIAÇÃO CIENTÍFICA - PAIC

imagens. Os ganhos de desempenho são consistentes tanto nas máquinas do ICOMP quanto na máquina do desenvolvedor. Isso demonstra que a programação paralela em Halide proporciona benefícios significativos em operações que envolvem processamento intensivo e que podem ser paralelizadas eficientemente, independentemente da máquina utilizada.

Os resultados obtidos confirmam a hipótese de que a linguagem Halide, com sua capacidade de gerar código otimizado para execução paralela, oferece um desempenho superior para algoritmos de processamento de imagens, especialmente em ambientes com múltiplos núcleos de CPU. A maior diferença observada entre Halide e Python ressalta a ineficiência de Python em tarefas computacionalmente intensivas, mesmo quando são utilizadas bibliotecas como OpenCV.

Por outro lado, a comparação entre Halide e C revela que, embora o C ainda seja uma escolha sólida para implementação de algoritmos de alto desempenho, Halide pode superar essas implementações devido à sua otimização automática e suporte intrínseco à paralelização. Isso é particularmente vantajoso em cenários onde o tempo de desenvolvimento e a facilidade de manutenção são considerações críticas.

Os resultados também indicam que a performance das implementações pode variar dependendo do hardware utilizado. Por exemplo, a máquina do desenvolvedor, com menor capacidade de paralelismo, mostrou ganhos menos expressivos em comparação com as máquinas do ICOMP. Isso sugere que, para maximizar os benefícios de Halide, é essencial utilizar hardware adequado que suporte alta paralelização, ou seja, máquinas com maior quantidade de núcleos disponíveis.

5. CONCLUSÃO

Este projeto teve como objetivo investigar e comparar o desempenho entre algoritmos tradicionais e paralelos no processamento de imagens. O projeto deu um foco especial na utilização da linguagem Halide para implementação das versões paralelas dos algoritmos. Através de uma série de experimentos conduzidos em diferentes configurações de hardware, foi possível observar os ganhos significativos proporcionados pela programação paralela, especialmente em tarefas intensivas como filtragens espaciais e detecção de bordas.

Os resultados obtidos validaram a hipótese inicial de que a programação paralela oferece um desempenho superior em relação às abordagens sequenciais tradicionais. As análises mostraram que a



UFAM

UNIVERSIDADE FEDERAL DO AMAZONAS
PRÓ-REITORIA DE PESQUISA E PÓS-GRADUAÇÃO
DEPARTAMENTO DE PESQUISA

PROGRAMA INSTITUCIONAL DE BOLSAS DE INICIAÇÃO CIENTÍFICA - PIBIC
PROGRAMA DE APOIO À INICIAÇÃO CIENTÍFICA - PAIC

programação paralela não apenas reduziu os tempos de processamento, mas também simplificou a implementação, permitindo que os desenvolvedores aproveitem ao máximo os recursos de hardware disponíveis sem a necessidade de uma otimização manual exaustiva.

Entretanto, o estudo também revelou algumas limitações. Primeiramente, o ganho de desempenho de Halide é fortemente dependente da arquitetura do hardware utilizado. Em máquinas com maior capacidade de paralelismo, os benefícios são mais pronunciados, enquanto que em máquinas com menos núcleos de processamento, os ganhos são menos expressivos. Além disso, a curva de aprendizado associada à utilização de Halide pode ser um desafio para desenvolvedores que não estão familiarizados com técnicas de programação paralela, mas ainda assim a linguagem oferece maior facilidade no momento da codificação uma vez que o desenvolvedor não necessita de um conhecimento tão profundo de hardware, sendo essa parte trabalhada pelo compilador Halide automaticamente.

Apesar dessas limitações, este trabalho contribui significativamente para o campo do processamento de imagens, oferecendo insights valiosos sobre a aplicação de técnicas paralelas em operações complexas. A adoção de Halide mostrou-se uma escolha vantajosa em contextos onde a eficiência e a velocidade de processamento são cruciais, destacando a importância de selecionar ferramentas de desenvolvimento que estejam alinhadas com as necessidades de desempenho do projeto.

Para pesquisas futuras, sugere-se explorar a aplicação de Halide em arquiteturas heterogêneas, como sistemas que utilizam tanto CPUs quanto GPUs, para avaliar se os ganhos observados podem ser ampliados. Além disso, uma investigação mais aprofundada sobre o impacto da otimização automática de Halide em diferentes tipos de operações de processamento de imagens pode oferecer novas oportunidades para melhorar ainda mais a eficiência em aplicações de larga escala.



UFAM

UNIVERSIDADE FEDERAL DO AMAZONAS
PRÓ-REITORIA DE PESQUISA E PÓS-GRADUAÇÃO
DEPARTAMENTO DE PESQUISA
PROGRAMA INSTITUCIONAL DE BOLSAS DE INICIAÇÃO CIENTÍFICA - PIBIC
PROGRAMA DE APOIO À INICIAÇÃO CIENTÍFICA - PAIC

REFERÊNCIAS

- [1] Mullapudi, R., Adams, A., Sharlet, D., Ragan-Kelley, J., & Fatahalian, K. (2016). Automatically Scheduling Halide Image Processing Pipelines. *ACM Transactions on Graphics (TOG)*, 35(4), Article 83, 11 pages. DOI: [10.1145/2897824.2925952](https://doi.org/10.1145/2897824.2925952).
- [2] Peter Pacheco, Matthew Malensek. An Introduction to Parallel Programming. Morgan Kaufmann, 2nd edition 2020. ISBN 978-0128046050.
- [3] Rafael C. Gonzalez, Richard E. Woods. Digital Image Processing. Pearson, 4th edition 2018. ISBN 978-1-292-22304-9.
- [4] Robert Robey, Yuliana Zamora. Parallel and High Performance Computing. Manning 2021. ISBN 978-1617296468.

Acesso ao repositório do GitHub com todos os códigos desenvolvidos:

Link:

<https://github.com/SebasNeto/Processamento-de-Imagens---Programacao-Paralela.git>

Acesso a todos os dados coletados na pesquisa - inclui o tempo de processamento de todos os algoritmos de acordo com a técnica (tradicional ou paralela) aplicada:

Link:

<https://drive.google.com/drive/folders/1Qsy1OPffPhSiCEgaYnYJ6thMOBx3zjm1?usp=sharing>