Estudo de novas abordagens para otimização do padrão de computação Irregular Wavefront Propagation

Mathias Oliveira, George Teodoro (Orientador)

***Abstract*—Neste trabalho, foi proposta uma modificação para o padrão de computação Irregular Wavefront Propagation Pattern (IWPP), com a finalidade de acelerar o processo de computação das operações de reconstrução morfológica e transformada de distância euclidiana. Em seguida, a modificação proposta para o IWPP, chamada de IWPP MP, foi utilizada na implementação das operações previamente citadas. Vale ressaltar que, essas implementações foram construídas por meio do modelo de programação CUDA da Nvidia. Por meio de experimentos, observou-se que a abordagem proposta pelo IWPP MP permite reduzir significativamente o tempo de computação das operações de reconstrução morfológica e transformada de distância euclidiana para instâncias que apresentam padrões complexos de propagação de informação.**

I. Introdução

As operações de transformada de distância euclidiana e reconstrução morfológica são muito utilizadas no processamento de imagens. Particularmente, essas operações são utilizadas no processamento de imagens biomédicas. Por exemplo, essas duas operações são aplicadas a imagens de amostras de tecido obtidas por meio de microscópios digitais com o intuito de identificar núcleos de células nessas imagens.

Embora essas operações sejam relevantes no processamento de imagens, elas são operações computacionalmente custosas, ou seja, são operações demoradas de serem computadas. Além disso, vale ressaltar que, muitas vezes, é desejado realizar essas operações em imagens de alta resolução, por exemplo, imagens de resolução 4kx4k, 8kx8k, ou até imagens de resolução 100kx100k. Com base nessas duas observações, fica clara a necessidade de implementações eficientes para essas operações.

Uma possível abordagem para diminuir o tempo de execução dessas operações é utilizar o processamento paralelo intensivo proporcionado pelas *General Purpose GPUs* (GPGPUs). Por meio de uma GPGPU, é possível processar milhares de threads simultaneamente de forma eficiente. Dessa forma, o objetivo deste trabalho foi construir implementações que pudessem tirar proveito dos recursos oferecidos pelas GPGPUs, para computar as operações de transformada de distância euclidiana e reconstrução morfológica de forma rápida e correta.

A. Irregular wavefront propagation pattern (IWPP)

Embora as GPGPUs propiciem uma computação paralela de alto desempenho, é necessário que o algoritmo que elas irão executar seja capaz de tirar proveito dos recursos que elas proporcionam. Dessa forma, o primeiro passo para executar as as operações de transformada de distância euclidiana e reconstrução morfológica de forma eficiente é modelar a computação dessas operações de forma a torná-la o mais paralelizável possível. Uma vez que essa modelagem foi construída, é possível implementá-la para ser processada eficientemente pelas GPGPUs.

Uma das formas de realizar essa tarefa é por meio do padrão de computação Irregular wavefront propagation pattern (IWPP). O IWPP é um padrão de computação comum em algoritmos que visam tirar proveito de recursos de processamento paralelo intensivo, por exemplo, os que são oferecidos pelas GPGPUs. Neste trabalho iremos nos ater ao IWPP realizado em imagens de duas dimensões, mas é fácil perceber que esse padrão de computação pode ser facilmente generalizado para dimensões maiores.

O IWPP recebe como entrada uma imagem de duas dimensões. Nesta imagem, cada pixel possui um conjunto de atributos. O primeiro passo do IWPP é selecionar e adicionar a uma fila um conjunto de pixels que irá propagar suas informações para seus vizinhos, esse conjunto de pixels é chamado de sementes. Ao longo da execução, cada pixel p da fila de sementes é processado e caso algum vizinho dele seja atualizado, ele é adicionado à fila de sementes. Esse processo é repetido iterativamente até que uma configuração estável seja alcançada. Uma descrição mais precisa do IWPP pode ser encontrada no algoritmo 1.

Algoritmo 1 Irregular Wavefront Propagation Pattern (IWPP)

```
Input: I (Imagem m × n)
S ← Conjunto de Sementes de I        ▷ Fase de Inicialização
while S ≠ ∅ do                       ▷ Fase de Propagação
    Remova p de S
    V ← Vizinhos de p em I
    while V ≠ ∅ do
        Remova q de V
        if CondicaoPropagacao(I(p), I(q)) then
            I(q) ← Update(I(p))
            S ← S ∪ {q}
        end if
    end while
end while
```

Embora o IWPP seja um padrão de computação simples, as duas operações que desejamos abordar podem ser expressas por meio de pequenas modificações desse padrão

de computação. Ademais, é perceptível que esse padrão de computação é facilmente computado de forma paralela.

B. Transformada de distância euclidiana

Primeiramente, temos que definir o que é a operação de transformada de distância euclidiana. Essa operação recebe como entrada uma imagem binária, cada pixel tem valor 0 ou 1. Nessa imagem, cada pixel pertence a exatamente um dos seguintes grupos de pixel: pixels background (pixels de valor 0) ou pixels foreground (pixels de valor 1). O resultado da operação é uma matriz em que na posição de cada pixel está a menor distância dele para um pixel do tipo background. Note que, para pixels do tipo background, essa distância é 0.

Para computar a operação de transformada de distância euclidiana foi escolhido o algoritmo de Danielson [1]. É importante ressaltar que o algoritmo de Danielson não é um algoritmo exato, mas sim uma heurística para aproximar o resultado da operação de transformada de distância euclidiana.

Embora esse algoritmo não seja exato, existe um limitante teórico para o erro máximo que ele produz e várias aplicações que utilizam a operação de transformada de distância euclidiana são capazes de lidar com esse erro de aproximação.

O algoritmo de Danielson computa a operação de transformada de distância euclidiana por meio da construção de um diagrama de Voronoi aproximado. Para computar esse diagrama de Voronoi, o algoritmo trata os pixels como pontos do plano euclidiano e divide o plano euclidiano, equivalentemente os pixels da imagem, em regiões que estão mais próximas de cada pixel do tipo background. Uma vez que o diagrama de Voronoi foi construído, é calculada, para cada pixel, a distância do pixel para o seu pixel background mais próximo, indicado pelo diagrama de Voronoi previamente computado.

Em [2] é proposto uma forma de modelar o algoritmo de Danielson por meio do padrão de computação IWPP. Mais precisamente, esse trabalho propõe uma forma de computar o diagrama de Voronoi aproximado, utilizado pelo algoritmo de Danielson, por meio do padrão de computação IWPP.

Considere a seguinte modelagem para a computação do diagrama de Voronoi aproximado. Uma matriz auxiliar é inicializada da seguinte forma: para pixels do tipo background atribuímos as coordenadas do pixel e para pixels do tipo foreground atribuímos a coordenada de um ponto do plano euclidiano cuja distância para qualquer pixel da imagem é maior do que a distância entre quaisquer dois pixels da imagem (chamamos esse pixel de pixel inf). Em seguida, os pixels background são colocados em uma fila (chamada de fila de sementes), e para cada pixel p dessa fila é informado para seus vizinhos na imagem qual é o pixel background mais próximo do pixel p. Com base nessa informação, o pixel vizinho decide se atualiza o seu pixel background mais próximo na matriz auxiliar. Caso o pixel vizinho atualize o seu pixel background mais próximo, ele é adicionado à fila de pixels sementes. O processo é repetido até que a fila se esvazie. Esse procedimento está descrito de forma mais precisa no algoritmo 2.

É importante ressaltar que o conjunto de vizinhos de um pixel interfere diretamente na qualidade da aproximação do diagrama de Voronoi construído. Neste trabalho, consideramos apenas o caso em que um pixel (i, j) possui como vizinhos apenas os pixels $(i-1, j)$, $(i+1, j)$, $(i, j-1)$, $(i, j+1)$, quando estes corresponderem à pixels da imagem. Esse padrão de vizinhos está ilustrado na figura 1.

Algoritmo 2 Transformada de Distância Euclidiana (IWPP)

Input: I (Imagem binária $m \times n$)
Output: M (Diagrama de Voronoi $m \times n$)
$S \leftarrow \emptyset$ ▷ Fase de Inicialização
$VR \leftarrow$ array nulo
for all p pixel de I
  $VR(p) \leftarrow (I(p) = 0)\ ?\ p : inf$
  if $I(p) = 0$ **then**
    $S \leftarrow S \cup \{p\}$
  end if
end for
while $S \neq \emptyset$ **do** ▷ Fase de Propagação
  Remova p de S
  $V \leftarrow$ Vizinhos de p em I
  while V $\neq \emptyset$ **do**
    Remova q de V
    if $Dist(q, VR(p)) < Dist(q, VR(q))$ **then**
      $VR(q) \leftarrow VR(p)$
      $S \leftarrow S \cup \{q\}$
    end if
  end while
end while
for all p pixel de I ▷ Cálculo de Distâncias
  $M(p) \leftarrow DIST(p, VR(p))$
end for

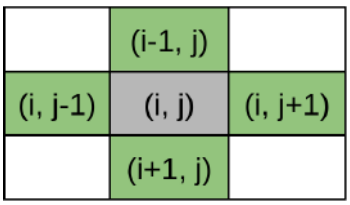


Figura 1: Conjunto de vizinhos (conectividade) de um pixel considerado nas implementações das operações de reconstrução morfológica e transformada de distância euclidiana.

Vale ressaltar que, a implementação construída neste trabalho para aproximar a computação da operação de transformada de distância euclidiana é uma pequena modificação do algoritmo descrito acima.

C. Reconstrução morfológica

Antes de prosseguir, precisamos definir a operação de reconstrução morfológica. Essa operação recebe como entrada duas imagens, uma chamada de marcador e a outra chamada de máscara. O marcador atribui para cada pixel da imagem um valor de 0 a 255. A máscara determina para cada pixel um limitante superior (um número de 0 a 255) para o valor que o marcador dele pode ter. O objetivo da reconstrução

morfológica é fazer com que cada pixel propague o valor do seu marcador para os seus pixels vizinhos que possuem um valor de marcador menor que o seu. Observe que essa propagação deve ser feita de forma que a máscara de cada pixel continue sendo respeitada. Assuma que essas imagens têm tamanho $n \times n$.

Em [2] é proposta uma forma de modelar a operação de reconstrução morfológica por meio do IWPP. Inicialmente são realizadas duas varreduras na imagem com o intuito de propagar o valor do marcador dos pixels para seus pixels vizinhos. A primeira varredura ocorre em ordem de *raster scan* (caminhando pelas linhas da imagem do pixel $(1, 1)$ até o pixel (n, n)) e a segunda varredura ocorre na ordem de *anti-raster scan* (caminhando pelas linhas da imagem do pixel (n, n) até o pixel $(1, 1)$). Esse tipo de varredura é descrita e explorada em mais detalhes em [3].

Em seguida, por meio de uma varredura na imagem são selecionados para uma fila (chamada de fila de sementes) os pixels p da imagem que possuem pelo menos um vizinho cujo valor do marcador é menor do que o valor da sua máscara e do que o valor do marcador do pixel p. Em seguida, cada pixel da fila é processado. Além disso, durante o processamento dos pixels da fila de sementes, se um vizinho de um pixel processado tem o valor do marcador alterado, esse vizinho é adicionado à fila de sementes. Esse procedimento está descrito de forma mais precisa no algoritmo 3. É importante ressaltar que no algoritmo 3, $N^+(p)$ (denota os vizinhos do pixel p que são alcançados pelo *raster scan* antes do pixel p) e $N^-(p)$ (denota os vizinhos do pixel p que são alcançados pelo *anti-raster scan* antes de p).

Algoritmo 3 Reconstrução Morfológica (IWPP)

Input: I (Imagem máscara $m \times n$)
Input: J (Imagem marcador $m \times n$)
$S \leftarrow \emptyset$ ▷ Fase de Inicialização
Durante um *raster scan* de I e J:
  Seja p o pixel atual de J na varredura
  $J(p) \leftarrow \min\{\max\{J(q) | q \in N^+(p) \cup \{p\}\}, I(p)\}$
Durante um *anti-raster scan* de I e J:
  Seja p o pixel atual de J na varredura
  $J(p) \leftarrow \min\{\max\{J(q) | q \in N^-(p) \cup \{p\}\}, I(p)\}$
  if $\exists q \in$ Vizinhos$(p) : J(q) < J(p)$ e $J(q) < I(q)$ **then**
    $S \leftarrow S \cup \{p\}$
  end if
while $S \neq \emptyset$ **do** ▷ Fase de Propagação
  Remova p de S
  $V \leftarrow$ Vizinhos de p em J
  while V $\neq \emptyset$ **do**
    Remova q de V
    if $J(q) < J(p)$ e $J(q) < I(q)$ **then**
      $J(q) \leftarrow \min\{J(p), I(q)\}$
      $S \leftarrow S \cup \{q\}$
    end if
  end while
end while

De maneira análoga ao descrito na seção I-B - Transformada de distância euclidiana, neste trabalho consideramos que um pixel (i, j) possui no máximo quatro vizinhos: os pixels $(i - 1, j)$, $(i + 1, j)$, $(i, j - 1)$, $(i, j + 1)$ sempre que esses corresponderem à pixels do marcador. Vale ressaltar que, a implementação construída neste trabalho para a computar a operação de reconstrução morfológica é uma pequena modificação do algoritmo descrito acima.

II. Referencial teórico

A principal referência para este trabalho é o artigo *"Efficient irregular wavefront propagation algorithms on hybrid CPU–GPU machines"* de Teodoro et al. [2]. Neste artigo, foram implementadas as operações de reconstrução morfológica e transformada de distância euclidiana, por meio do padrão de computação IWPP, de forma eficiente para sistemas com GPGPUs, com multi GPGPUs e com multi CPUs e GPGPUs. Observe que as modelagens utilizadas por esse artigo para as operações de reconstrução morfológica e transformada de distância euclidiana foram apresentadas na seção I - Introdução por meio dos algoritmos 2 e 3.

Ademais, no artigo *"Efficient Computation of Morphological Greyscale Reconstruction"* de Pavel Karas [3], é proposta uma outra forma de implementar a operação de reconstrução morfológica com a finalidade de executá-la em GPGPUs. Neste trabalho, a operação de reconstrução morfológica foi implementada por meio do algoritmo SR. Na implementação desse algoritmo apresentada em [3], o valor do marcador de um pixel é propagado para seus vizinhos por meio de varreduras por linhas paralelas às laterais da imagem até que a imagem alcance um estado estável, ou seja, um estado em que não ocorrem mais propagações. Na figura 2 está ilustrada uma varredura por linha que se inicia na lateral esquerda da imagem. Além disso, o algoritmo 4 explicita o tipo de processamento realizado durante essa varredura.

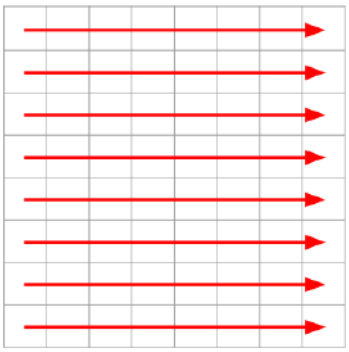


Figura 2: Sentido de propagação de informação de uma varredura por linha que se inicia na lateral esquerda da imagem.

Em ambos os trabalhos foram obtidos speedups relevantes em relação à execução sequencial das operações de transformada de distância euclidiana e reconstrução morfológica. Dessa forma, neste trabalho iremos propor uma modelagem alternativa para as operações de transformada de distância euclidiana e reconstrução morfológica com o objetivo diminuir o tempo de execução dessas operações. Vale ressaltar que, as implementações que serão apresentadas neste trabalho foram

Algoritmo 4 Varredura por linha (lateral esquerda)

```
Input: I (imagem n × n)
for i = 1 to n do
    for j = 1 to n − 1 do
        if CondicaoPropagacao(I(i, j), I(i, j + 1)) then
            I(i, j + 1) ← Update(I(i, j))
        end if
    end for
end for
```

desenhadas de forma a tirar proveito dos recursos providos pelas GPGPUs.

III. Implementação construída

Neste trabalho foram construídas: uma implementação para a operação de reconstrução morfológica e uma implementação de um algoritmo aproximativo para a operação de transformada de distância euclidiana por meio da API de C++ do modelo de programação CUDA [4]. Note que, CUDA é um modelo de programação para GPGPUs da Nvidia.

Antes de apresentar a implementação, é necessário introduzir um conceito importante no modelo de programação CUDA, o conceito de *shared memory*. De modo análogo a uma CPU, uma GPGPU possui uma hierarquia de memória. Na hierarquia de memória das GPGPUs existem vários tipos de memória, dentre eles a *memória global* e a *memória compartilhada (shared memory)*.

Por um lado, a memória global é o tipo mais abundante de memória que uma GPGPU possui. Esse tipo de memória é fisicamente armazenado em memória DRAM. Dessa forma, embora a memória global ofereça uma grande capacidade de armazenamento, a interação com essa memória é lenta e demanda vários ciclos de clock para ser executada.

Por outro lado, a memória compartilhada (shared memory) é um tipo de memória escasso nas GPGPUs. Esse tipo de memória é armazenada em registradores no chip da GPGPU. Além disso, as threads de um mesmo bloco de threads podem compartilhar o acesso a esse tipo de memória. Dessa forma, a interação com essa memória é rápida, mas a quantidade de informação que essa memória armazena é pequena.

Portanto, um dos desafios na implementação de um algoritmo em CUDA é gerenciar o acesso dos blocos de threads a esses tipos de memória de forma a tentar minimizar a latência envolvida nas operações de leitura e escrita aos dados que devem ser processados durante a computação que é realizada na GPGPU.

A implementação proposta neste trabalho é análoga à implementação apresentada no artigo [2] pois ambas se baseiam no padrão de computação IWPP. Uma das principais diferenças entre a implementação construída nesse trabalho e a implementação apresenta em [2] está na forma como a memória compartilhada é utilizada. Ao invés de usar a memória compartilhada como um buffer para a fila de pixels que devem ser processados, como foi feito em [2], foi decidido usar a memória compartilhada para armazenar uma sub imagem de tamanho 32x32 da imagem original (chamada de megapixel) e realizar a propagação da informação dos pixels desse megapixel de forma mais rápida e independente da imagem original. Em seguida, os resultados computados no megapixel são transferidos para a imagem original.

Algoritmo 5 IWPP com Megapixel (IWPP MP)

```
Input: I (Imagem n × n)
                                          ▷ Fase de Inicialização
filaGlobal← Todos os megapixels da imagem
Faça em paralelo:                         ▷ Fase de Propagação
__shared__ MP[34][34]
while filaGlobal ≠ ∅ do
    Remova p da filaGlobal
    MPAtualizados ← ∅
    leituraDoMegapixel(MP, I, p)
    processamentoDoMegapixel(MP)
    MPAtualizados ← escritaDoMegapixel(I, p, MP)
    filaGlobal ← filaGlobal ∪ MPAtualizados
end while
```

Embora a alteração na estrutura do padrão de computação IWPP seja simples, ela diminui a quantidade de operações atômicas realizadas durante a execução das duas operações de interesse, permite que a computação seja feita em uma memória mais rápida e permite que tanto as leituras quanto as escritas na memória global sejam agregadas (coalesced), de modo a diminuir a latência envolvida na interação com a memória global. A nova proposta de modelagem do IWPP está descrita de forma mais precisa no algoritmo 5. Ressaltamos que, cada um dos passos do algoritmo 5 será detalhado em seguida.

A. Fila global

Para armazenar os megapixels que devem ser processados foi utilizada uma fila global que é compartilhada por todos os blocos de threads que estão processando a imagem. Note que por meio dessa abordagem, problemas de desbalanceamento de carga são mitigados.

É importante ressaltar que a fila de megapixels fica armazenada na memória global. Optou-se por não utilizar um buffer para a fila de megapixels na memória compartilhada pois isso aumentaria a quantidade de recursos que cada bloco de threads demandaria para ser executado e, consequentemente, diminuiria a quantidade de blocos de threads que a GPGPU consegue executar simultaneamente.

Por fim, é importante ressaltar que foi implementado um contador de blocos ativos, ou seja, um contador de blocos que estão processando algum megapixel, com o intuito de evitar o problema de desbalanceamento de carga ao processar determinadas instâncias. Dessa forma, os blocos de threads só terminam a sua execução quando todos os blocos tentam ler da fila e verificam que a fila está vazia. Isso implica que, mesmo que a fila de megapixels esteja vazia, se existe algum bloco de threads processando um megapixel, todos os outros blocos de threads continuam executando até que de fato todos os megapixels da fila tenham sido processados.

Para exemplificar o possível problema mencionado acima, considere, no caso da reconstrução morfológica, uma imagem

de marcador em que apenas um pixel tem valor não nulo e a imagem de máscara é tal que todo pixel possui valor 255. A solução correta para a operação de reconstrução morfológica é atribuir esse valor não nulo para todos os pixels da imagem marcador. Observe que, para essa instância, assumindo que o megapixel que possui o pixel não nulo é o último da fila de megapixels, sem o contador de blocos ativos, vários blocos de threads terminam a sua execução de forma precoce após tentar ler da fila de megapixels e verificar que ela está vazia. Entretanto, eventualmente, todos os megapixels da imagem são inseridos na fila de megapixels, de modo a sobrecarregar os poucos blocos de threads que não tiveram a execução encerrada. É fácil ver que o contador de blocos ativos resolve esse problema.

O exemplo acima pode parecer artificial, mas pode ocorrer no processamento de várias instâncias mesmo que em escala menor. Por isso, esse problema foi tratado na implementação construída para as operações de transformada de distância euclidiana e reconstrução morfológica.

B. Leitura do megapixel

A primeira etapa para processar um registro lido da fila de megapixels é transferir o megapixel correspondente da memória global para a memória principal. Nessa etapa, o bloco de threads cópia da memória global para a memoria compartilhada uma sub imagem de 32x32 pixels (um megapixel) da imagem que está sendo processada. Em seguida, o bloco de threads cópia os pixels das laterais superior, inferior, esquerda e direita do megapixel que foi lido da fila global para a memória compartilhada. Caso alguma dessas laterais não exista, atribuímos um valor que faz com que essas laterais não interfiram na propagação que irá ocorrer na memória compartilhada. A figura 3 exemplifica a estrutura de um megapixel.

Observe que, embora o processo de leitura de um megapixel envolva muitas leituras, a leitura do bloco principal do megapixel e das laterais superior e inferior ocorrem em posições de memória consecutivas e alinhadas, portanto, são agregadas (coalesced) pelo compilador, o que diminui o custo de realizar essas operações.

Note que as quinas da sub imagem de 34x34 pixels (figura 3) não são lidas. Isso ocorre pois seriam leituras desnecessárias para a conectividade escolhida para as implementações de transformada de distância euclidiana e reconstrução morfológica. Relembre que estamos assumindo que cada pixel possui no máximo quatro vizinhos.

C. Processamento do megapixel

Uma vez que os dados estão na memória compartilhada é necessário propagar as informações dos pixels para seus vizinhos. Com o intuito de reduzir o número de operações atômicas e inspirado na implementação de Karas [3], foi decidido implementar a propagação de informação dentro do megapixel por meio de varreduras por linha (essa operação foi descrita na figura 2 e no algoritmo 4), até que o megapixel alcance um estado estável. Dessa forma, cada interação do processamento do megapixel envolve quatro varreduras por

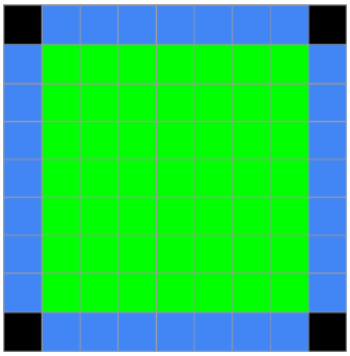


Figura 3: O bloco principal (megapixel de 32x32 pixels lido da fila global) está colorido de verde e as laterais desse megapixel estão coloridas de azul. Note que a lateral direita do megapixel (bloco principal) é a primeira coluna do megapixel (da sub imagem 32x32) à direita. É fácil ver que relações análogas podem ser estabelecidas entre os megapixels vizinhos do bloco principal e as laterais do bloco principal. Por fim, observe que as quinas coloridas de preto sempre são inicializadas com um valor que não afeta a propagação que ocorre no restante do megapixel. Note que, em um abuso de notação, também nos referimos à sub imagem inteira acima, a sub imagem 34x34 armazenada na memória compartilhada, como um megapixel.

linha do megapixel. Essas quatro varreduras por linha se iniciam na lateral superior, inferior, esquerda e direita do megapixel.

Vale ressaltar que foram realizados testes comparando o tipo de propagação escolhido para o megapixel com a propagação proposta pelo padrão de computação IWPP. Nestes testes, foi constatado que a propagação por meio de varreduras por linha, em geral, é mais rápida do que a propagação por onda proposta pelo IWPP. Isso ocorre pois a implementação baseada na propagação por ondas proposta pelo IWPP exige uma grande quantidade de operações atômicas e a manutenção de uma fila de pixels do megapixel que devem ser processados. Observe que o uso dessa fila geraria interações com a memória global e possivelmente demandaria mais recursos para cada bloco de threads ser executado, o que diminuiria a quantidade de blocos de threads que a GPGPU consegue executar simultaneamente.

D. Escrita na memória global

Uma vez que a propagação dentro do megapixel foi computada é necessário escrever os resultados obtidos na imagem original. Essa etapa tem que ser realizada de forma atômica, pois é possível que mais de um bloco de threads esteja processando o mesmo megapixel ou que dois blocos de threads estejam processando megapixels vizinhos. Observe que, também escrevemos na imagem original os resultados que a propagação do megapixel gera para os pixels das laterais do megapixel que foi processado. Portanto, a escrita dos resultados computados na imagem original tem que ser realizada de forma atômica para evitar condições de corrida. A forma de escrita na memória global utilizada na implementação constrída está ilustrada no algoritmo 6. Note que, por meio da forma de escrita utilizada no algoritmo 6, é possível que o

Algoritmo 6 Escrita do valor de um pixel na imagem original

```
Input: I (Imagem n × n)
Input: p (Pixel que será atualizado)
Input: novoValor (Valor que será escrito)
while true do
    valorAssumido ← I(p)
    if condAtualização(valorAssumido, novoValor) then
        valorAntigo ← atomicCAS(&I(p), valorAssumido, novoValor)
        if valorAntigo = valorAssumido then
            break                          ▷ novoValor escrito
        end if
    else
        break                              ▷ novoValor não escrito
    end if
end while
```

valor obtido para um pixel do megapixel não seja atualizado na imagem original.

Ainda na escrita dos resultados, é verificado quais megapixels devem ser reprocessados, ou seja, adicionados à fila de megapixels. Utiliza-se o seguinte critério: se algum pixel do bloco principal for atualizado na imagem original, o megapixel deve ser reprocessado. Ademais, se algum pixel das laterais do bloco principal for atualizado na imagem original, o megapixel que comporta essa lateral deve ser processado, ou seja, ele deve ser adicionado à fila de megapixels. Portanto, os megapixels que tiveram alguma escrita que os modificou na imagem original são adicionados à fila de megapixels, para serem processados novamente.

Observe que um megapixel só é considerado estável quando um bloco de threads tenta processá-lo e verifica que nenhum pixel desse megapixel deve ser atualizado na imagem original.

E. Reconstrução morfológica

Para utilizar o IWPP MP, apresentado acima, para computar a operação de reconstrução morfológica foram realizadas as seguintes adaptações. Primeiro, é realizada uma varredura por linha iniciada em cada lateral da imagem para propagar o valor de marcador dos pixels. Note que, esse processo é análogo aos *raster* e *anti-raster scan* que ocorrem no início do no algoritmo 3. Em seguida, o processamento segue exatamente como descrito no algoritmo 5.

Além disso, algumas modificações são necessárias nas etapas do IWPP MP para que a implementação compute a operação de reconstrução morfológica. Na etapa de leitura do megapixel, temos que ler para a memória compartilhada tanto a sub imagem da máscara quanto a sub-imagem do marcador associadas ao megapixel que será processado. Junto a isto, na etapa de propagação dentro do megapixel, é necessário especificar o critério de propagação de informação. Note que a verificação realizada é a mesma que a apresentada no algoritmo 3. Ou seja, verificamos se o valor do marcador do pixel que está sendo processado é maior que o valor do marcador do seu vizinho e se o valor do marcador do vizinho é menor que o valor da sua máscara. Ademais, observe que, a forma de atualizar o valor do marcador do pixel vizinho é a mesma que foi apresentada no algoritmo 3.

Por fim, para garantir a consistência da computação é necessário modificar a condição de escrita dos resultados obtidos na fase de propagação do megapixel para a imagem original. Dessa forma, o valor do marcador de um pixel só é atualizado na imagem marcador original se o valor computado para ele durante a propagação do megapixel for maior do que o seu valor de marcador atual. Note que, a imagem máscara nunca é modificada. Dessa forma, é perceptível que essas modificações são suficientes para que a implementação do IWPP MP compute a operação de reconstrução morfológica.

F. Transformada de distância euclidiana

Observe que na computação da operação de transformada de distância euclidiana, o IWPP MP é empregado para computar o diagrama de Voronoi aproximado da imagem. Primeiro, inicializamos uma matriz auxiliar que armazenará o diagrama de Voronoi da mesma forma como foi proposto por [2] e descrito na seção I-B - Transformada de distância euclidiana. Em seguida, basta alterar o critério de propagação da informação dos pixels para os vizinhos e o critério de escrita dos resultados obtidos durante a propagação do megapixel na matriz original para que o IWPP MP compute o diagrama de Voronoi aproximado da imagem.

Observe que um pixel p só propaga sua informação de pixel background mais próximo para um vizinho se a distância do vizinho para o seu atual pixel background mais próximo for maior do que a sua distância para o pixel background mais próximo do pixel p. Observe que o essa verificação é a mesma que foi apresentada no algoritmo 2.

Ademais, na etapa de escrita dos resultados obtidos na imagem original, o pixel background mais próximo de um pixel p do megapixel só é atualizado na imagem original se a distância de p para o pixel background mais próximo obtido após a propagação do megapixel for menor do que a distância de p para o pixel background que está registrado na imagem original.

É perceptível que por meio dessas pequenas modificações, o IWPP MP consegue computar uma aproximação para o diagrama de Voronoi da imagem. Em seguida, basta computar a distância de cada pixel para o pixel background indicado pelo diagrama de Voronoi calculado. Esse processo ocorre da mesma forma como foi descrito no algoritmo 2.

IV. Metodologia

Para verificar como a nova modelagem do IWPP se compara com a modelagem implementada por Teodoro et al. [2], foram implementadas as modificações do IWPP MP para computar as operações de reconstrução morfológica e transformada de distância euclidiana em CUDA.

Além disso, as implementações baseadas no IWPP apresentado na seção I - Introdução para computar as operações de reconstrução morfológica e de transformada de distância euclidiana propostas em [2] foram extraídas da biblioteca nscale. O código dessa biblioteca foi gentilmente oferecido por um dos autores com a finalidade de servir de parâmetro

de comparação para a implementação que foi construída neste trabalho.

Dessa forma, foram extraídas da biblioteca nscale duas implementações da operação de reconstrução morfológica (uma é baseada no padrão de computação IWPP e a outra é baseada em varreduras por linha como proposto por Karas [3]) e uma implementação da operação de transformada de distância euclidiana (baseada no padrão de computação IWPP). Em seguida, foram realizados testes para verificar tanto a corretude quanto o tempo de execução de todas as implementações.

A. Corretude das implementações

O Matlab oferece implementações exatas para as operações de reconstrução morfológica e transformada de distância euclidiana. Dessa forma, para garantir a corretude das implementações testadas, para cada caso de teste selecionado, a saída de cada uma das implementações que estavam sendo testadas foi comparada com a resposta computada pelo Matlab.

No caso da reconstrução morfológica foram extraídas da biblioteca nscale as duas implementações que foram corretas em todos os casos de teste. Ademais, verificou-se que as implementações construídas nesse trabalho também se mostraram corretas em todos os casos de teste.

No caso da operação de transformada de distância euclidiana, verificou-se se o erro máximo das soluções aproximadas em relação à solução exata era tolerável. Ademais, foi verificado se os erros que ocorriam na computação do diagrama de Voronoi aproximado eram aceitáveis. Ou seja, foi verificado se eram erros oriundos do fato que o algoritmo de Danielson não fornece garantias de uma solução ótima ou se eram erros de implementação dos algoritmos.

B. Tempo de execução

Para verificar o tempo de execução das implementações em CUDA foi utilizado o objeto high_resolution_clock da biblioteca chrono da STL presente no C++. Essa decisão foi tomada pois pode ser necessário disparar várias vezes os kernels que realizam a computação das operações de reconstrução morfológica e transformada de distância euclidiana. Dessa forma, foi priorizado medir o tempo que um usuário teria que esperar para a operação ser computada e não apenas o tempo de uso da GPGPU.

Dessa forma, o tempo de execução coletado é o tempo entre o host chamar o procedimento que computa a reconstrução morfológica, ou a transformada de distância euclidiana, e retomar o controle do fluxo de execução do programa. Para coletar o tempo de execução dos algoritmos implementados no Matlab foi utilizado o comando tic;toc;.

Por fim, para cada instância de teste selecionada, cada uma das implementações foi executada 5 vezes e escolheu-se como tempo de execução, da instância pela implementação avaliada, a mediana desses 5 tempos de execução.

C. Hardware utilizado

Para executar as implementações do IWPP MP e as implementações extraídas da biblioteca nscale foi utilizado um computador com um processador i7-7500u com 8GB de memória RAM e uma GPGPU NVIDIA GeForce 940 MX (2GB). Vale ressaltar que, essa GPGPU possui arquitetura Maxwell e compute capability 5.0.

Para executar as implementações do Matlab, foi utilizado o Matlab online que executa em 8 cores e 8 threads de um processador Xeon(R) Platinum 8375C. Vale ressaltar que, a implementação do Matlab não possui acesso a uma GPGPU dedicada.

D. Casos de teste

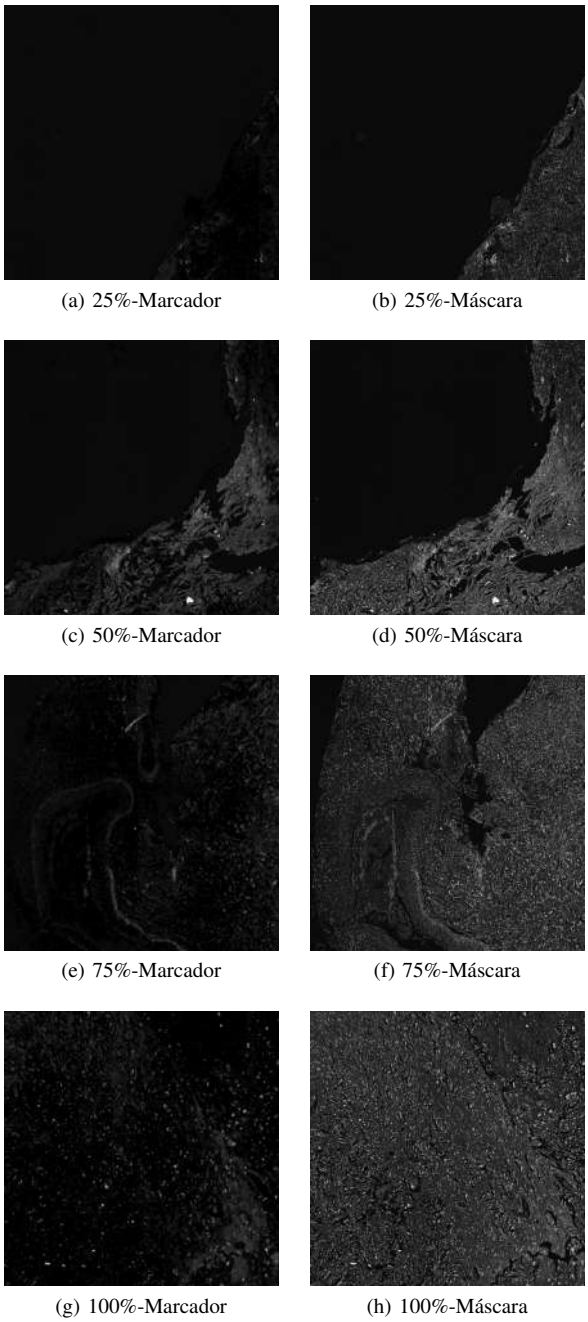
Nesta seção iremos descrever brevemente os casos de teste selecionados para medir o tempo de execução das implementações construídas. As imagens usadas como caso de teste podem ser encontradas no link: Casos de Teste.

Foram selecionados 8 casos de teste para verificar a corretude e o tempo de execução das implementações da operação de reconstrução morfológica. Desses 8 casos de teste, 4 são imagens 4kx4k de amostras de tecido tingidas com corante. Essas imagens foram pré-processadas para extrair as imagens marcador e máscara utilizadas pela operação de reconstrução morfológica. (figura 4). Essas imagens diferem entre si pois em cada uma delas o tecido cobre uma quantidade diferente da imagem, e o restante da imagem é um fundo branco. Dessa forma, nos casos de teste, temos uma imagem em que o tecido cobre 25%, 50%, 75% e 100% da imagem. Os outros 4 casos de teste foram gerados aleatoriamente. Dois desses testes são duas imagens de 4kx4k e os outros dois são duas imagens imagens de 8kx8k.

É importante ressaltar que as imagens aleatórias não possuem padrões que complicam a computação da operação de reconstrução morfológica. Essencialmente queremos ressaltar que as imagens de amostras de tecido possuem padrões curvos e complicados que tornam mais custoso o processo de computação da reconstrução morfológica, portanto, são as instâncias de teste mais difíceis.

Para comparar as implementações da operação de transformada de distância euclidiana foram selecionadas 12 instâncias de teste. Dessas 12 instâncias de teste 4 são imagens geradas aleatoriamente. Duas dessas instâncias aleatórias são imagens de 4kx4k e as outras duas são imagens de 8kx8k. As outras instâncias de teste são mais estruturadas:

- 16 circles: duas imagens, uma 4kx4k e uma 8kx8k, em que os pixels background estão dispostos em 16 círculos cujos centros são equidistantes na imagem.
- Big circle: duas imagens, uma 4kx4k e uma 8kx8k, em que os pixels background estão dispostos sobre um círculo cujo centro é o pixel central da lateral inferior da imagem e cujo raio é metade da largura da imagem.
- Squares on corners: uma imagem 4kx4k em que os pixels background estão concentrados em 4 quadrados que se encontram nos cantos da imagem.
- Diagonal: uma imagem 4kx4k em que os pixels background estão dispostos sobre a diagonal principal da imagem.
- X: uma imagem 4kx4k em que os pixels background estão dispostos sobre a diagonal e a anti diagonal da imagem.



(a) 25%-Marcador (b) 25%-Máscara

(c) 50%-Marcador (d) 50%-Máscara

(e) 75%-Marcador (f) 75%-Máscara

(g) 100%-Marcador (h) 100%-Máscara

Figura 4: Imagens de amostra de tecido usadas como casos de teste da operação de reconstrução morfológica.

- Central circle: uma imagem 4kx4k em que os pixels background estão dispostos sobre um círculo centrado no centro da imagem de raio aproximadamente 1024.

Novamente, é importante ressaltar que as imagens aleatórias não possuem padrões curvos que podem dificultar o processamento da imagem. Essencialmente, no processamento realizado pelo IWPP nas imagens aleatórias, em geral, um pixel background propaga sua informação para poucos vizinhos que estão próximos dele, o que torna o processo de propagação mais simples.

V. Resultados

Antes de analisar os resultados obtidos é importante esclarecer a configuração de execução das implementações. Para executar as implementações extraídas da biblioteca nscale foram utilizadas as configurações de execução de kernel (forma de bloco de threads, forma do grid de blocos de threads) que foram encontradas na biblioteca nscale, ou seja, essas informações não foram alteradas durante a extração dos algoritmos da biblioteca.

Para executar as implementações construídas baseadas no IWPP MP, foram utilizados blocos de threads de 32 threads. A escolha por blocos de threads menores foi tomada após a realização de alguns experimentos. Essencialmente, verificou-se que ao usar blocos de threads menores era possível aumentar significativamente a quantidade de blocos que a GPGPU consegue executar simultaneamente. Dessa forma, verificou-se que ao utilizar blocos de threads menores o desempenho foi melhor do que ao utilizar blocos de threads com mais threads.

A. *Reconstrução morfológica*

Os resultados obtidos estão sumarizados na tabela I.

Instância	nscale IWPP	nscale SR	IWPP MP	Matlab (CPU)
25%	288	3823	187	522
50%	389	1984	235	611
75%	1618	4978	582	1812
100%	1567	3436	596	1621
R1-4k	89	237	189	399
R2-4k	87	238	190	393
R3-8k	348	923	761	1657
R4-8k	348	1041	762	1662

Tabela I: Tempos de execução (em ms) das implementações da operação de reconstrução morfológica. As instâncias cujo nome iniciam com R, foram geradas aleatoriamente.

Por meio dos resultados apresentados na tabela I, é perceptível que para as imagens de amostra de tecido selecionadas, a implementação construída nesse trabalho, IWPP MP, é mais rápida do que ambas as implementações da biblioteca nscale. É importante observar que, para as instâncias mais complexas (75% e 100%), a implementação IWPP MP executa até 2.7x mais rápido do que a implementação de IWPP da biblioteca nscale. Além disso, é perceptível que o speedup é ainda maior quando comparamos a implementação IWPP MP com a implementação SR da biblioteca nscale.

Entretanto, para as instâncias aleatórias o resultado observado é o oposto. A implementação de IWPP MP é ligeiramente mais rápida do que a implementação SR da biblioteca nscale, mas pode demorar cerca de 2.2x mais tempo do que a implementação IWPP da biblioteca nscale.

Em resumo, os resultados observados sugerem que para imagens mais complexas a implementação IWPP MP é mais veloz do que a implementação IWPP da biblioteca nscale. Entretanto, para imagens com padrões de propagação mais simples, a implementação baseada no IWPP da biblioteca nscale é a mais indicada. Observe que, para poder afirmar essas conclusões é necessário realizar mais testes, principalmente, com outras imagens de amostra de tecido.

B. Transformada de distância euclidiana

Os resultados obtidos estão sumarizados nas tabelas II e III.

Instância	nscale IWPP	IWPP MP	Matlab (CPU)
16 circles	2150	1493	174
Big circle	3841	1560	160
Central circle	5397	1666	172
Squares on corners	3700	567	173
Diagonal	1268	2745	186
X	1291	2690	202
R1-4k	552	669	225
R2-4k	289	602	231
R3-8k	2333	2678	901
R4-8k	1204	2405	950
Big circle-8k	25965	7778	850
16 circles-8k	15527	6116	813

Tabela II: Tempos de execução (em ms) das implementações da operação de transformada de distância euclidiana. As instâncias cujo nome iniciam com R, foram geradas aleatoriamente.

Instância	nscale IWPP	IWPP MP	Matlab (CPU)
16 circles	0.58579	0.58579	-
Big circle	0.58579	0.58579	-
Central circle	0.58579	0.58579	-
Squares on corners	0.0006	0.0006	-
Diagonal	0.58579	0.58579	-
X	0.58579	0.58579	-
R1-4k	0.58579	0.34320	-
R2-4k	0.58579	0.58579	-
R3-8k	0.58579	0.58579	-
R4-8k	0.58579	0.58579	-
Big circle-8k	0.58579	0.58579	-
16 circles-8k	0.58579	0.58579	-

Tabela III: Erro máximo entre a solução da implementação e a solução exata para a operação de transformada de distância euclidiana. Relembramos que as implementações IWPP são heurísticas aproximativas e que a implementação do Matlab é exata.

Primeiro, é importante observar na tabela III que as implementações avaliadas geram soluções que se comportam de forma parecida, pois o erro máximo que ambas as implementações testadas geram em relação à solução exata, em geral, é o mesmo.

Por meio da tabela II, é perceptível que o desempenho da implementação de IWPP da biblioteca nscale é melhor para imagens aleatórias. Para duas das instâncias aleatórias selecionadas (R2 e R4), o tempo de execução da implementação IWPP MP pode ser 2x o tempo de execução da implementação IWPP nscale. Também é possível observar o mesmo comportamento para as instâncias de teste “X” e “Diagonal”.

Entretanto, para imagens com padrões de propagação mais complexos, particularmente imagens com curvas, a situação se mostra completamente diferente. Para as demais imagens de 4kx4k, o tempo de execução da implementação IWPP MP pode ser até 6 vezes mais rápido do que o tempo de execução da implementação IWPP nscale. Para as demais imagens de 8kx8k, o tempo de execução da implementação IWPP MP pode ser até 3 vezes mais rápido do que o tempo de execução da implementação IWPP nscale.

Por fim, é interessante observar na tabela II a coluna de tempo de execução da implementação presente no Matlab. Note que, em todos os casos de teste, a implementação do Matlab (que é baseada em outro algoritmo) é muito mais rápida do que as implementações baseadas no padrão de computação IWPP. Como a implementação do Matlab executa apenas a nível de CPU, é esperado que o algoritmo utilizado pela implementação do Matlab apresente tempos de execução ainda menores quando executado em GPGPUs. Portanto, os resultados obtidos sugerem que para computar de forma rápida a operação de transformada de distância euclidiana, a abordagem do padrão de computação IWPP não é a melhor possível.

VI. Análise de resultados e trabalhos futuros

Na seção V - Resultados foi verificado que para processar instâncias geradas aleatoriamente as implementações construídas neste trabalho (IWPP MP) apresentam um desempenho pior do que o desempenho das implementações correspondentes da biblioteca nscale (IWPP nscale).

É possível que o desempenho da implementação IWPP nscale seja melhor em instâncias aleatórias na operação de transformada de distância euclidiana pois, nessas imagens, a onda originada por um pixel do tipo background se propaga para poucos pixels vizinhos. Isso ocorre, pois existem muitos pixels backgrounds espalhados na imagem. Dessa forma, é possível que a granularidade de propagação utilizada pela implementação IWPP nscale seja mais adequada para processar essas imagens do que a granularidade de propagação utilizada pela implementação IWPP MP.

Note que o mesmo fenômeno pode justificar os resultados observados para operação de reconstrução morfológica. Como as imagens aleatórias não possuem padrões que complicam a propagação dos pixels, após o processamento inicial de *raster* e *anti raster scan* (ou varreduras por linha), a imagem de marcador já está muito próxima do resultado final da operação de reconstrução morfológica. Dessa forma, é possível que a granularidade de propagação com qual a implementação IWPP nscale trabalha é mais eficiente para fazer os ajustes finais do que a granularidade de propagação utilizada pela implementação IWPP MP.

Para exemplificar o que foi dito no parágrafo anterior, após as varreduras por linhas iniciadas em cada lateral da imagem, as instâncias de teste da operação de reconstrução morfológica R1, R2, R3 e R4, já possuem respectivamente 91.64%, 91.63%, 91.63%, 91.62% dos valores corretos na imagem marcador. Ademais, os pixels da imagem marcador que ainda não receberam o valor correto da operação de reconstrução morfológica se encontram em ilhas de poucos pixels rodeadas por muitos pixels que já possuem o valor correto na imagem marcador.

Outro fator que pode justificar o pior desempenho observado das implementações IWPP MP em relação à implementação IWPP nscale correspondente em alguns casos de teste é a forma como a leitura e a escrita na fila de megapixels ocorre. Uma vez que todos os blocos de threads compartilham a mesma fila de megapixels é necessário utilizar uma primitiva de sincronização para evitar condições de corrida. Por exemplo, para evitar que um bloco de threads leia uma posição de

memória antes de ela terminar de ser escrita por outro bloco de threads, ou que um bloco de threads escreva em uma posição de memória enquanto um outro bloco de threads está lendo essa mesma posição de memória. Essas condições de corrida podem gerar comportamentos indefinidos e afetar a corretude do resultado computado.

Nas implementações do IWPP MP é usada apenas uma mutex para evitar as condições de corrida previamente citadas. Entretanto, uma forma de otimizar a interação dos blocos de threads com a fila de megapixels é implementando o padrão de sincronização leitores/escritores. Por meio da implementação desse padrão de sincronização, seria possível acelerar a interação dos blocos de threads com a fila de megapixels sem comprometer a corretude da implementação, pois os problemas citados anteriormente ocorrem apenas quando ocorrem simultaneamente operações de leitura e escrita. Entretanto, esses problemas não ocorrem se várias operações do mesmo tipo acontecerem simultaneamente.

Outra melhoria que pode ser feita para diminuir o tempo de execução das implementações IWPP MP é a implementação de uma seleção mais criteriosa para decidir se um megapixel deve ser adicionado ou não à fila de megapixels inicial. Nas implementações construídas do IWPP MP, a fila de megapixels inicial contém todos os megapixels da imagem independente do conteúdo desses megapixels. Entretanto, por meio de uma análise mais criteriosa, seria possível diminuir a quantidade de megapixels que são adicionados à fila de megapixels inicial e, consequentemente, a quantidade de trabalho que é realizada pelas implementações IWPP MP.

Por fim, seria interessante adicionar suporte para outras formas de conectividade para a operação de reconstrução morfológica, por exemplo, quando consideramos que um pixel (i, j) tem no máximo oito vizinhos: $(i-1, j)$, $(i+1, j)$, $(i, j-1)$, $(i, j+1)$, $(i-1, j-1)$, $(i-1, j+1)$, $(i+1, j-1)$, $(i+1, j+1)$, quando esses corresponderem à pixels da imagem.

Para a operação de transformada de distância euclidiana, seria interessante estudar o algoritmo utilizado pelo Matlab e tentar implementá-lo por meio do modelo de programação CUDA para verificar se por meio desse algoritmo é possível diminuir ainda mais o tempo de execução dessa operação.

VII. Código

O código das implementações construídas neste trabalho pode ser encontrado no repositório https://github.com/oliveira-mathias/IWPP-MP.

VIII. Conclusão

Por meio do que foi apresentado neste trabalho é perceptível que a introdução do uso de megapixels no padrão de computação IWPP é capaz de diminuir significativamente o tempo de execução das operações de reconstrução morfológica e transformada de distância euclidiana para imagens complexas.

Além disso, os resultados obtidos pela implementação IWPP MP da operação de reconstrução morfológica sugerem que vale a pena investir tempo e polir a implementação construída com o intuito de diminuir ainda mais o tempo de execução dessa operação.

Por outro lado, os resultados associados às implementações da operação de transformada de distância euclidiana, sugerem que a melhor abordagem para executar de forma rápida essa operação é o uso de um novo algoritmo, possivelmente, o algoritmo utilizado pelo Matlab.

Agradecimentos

Gostaria de agradecer ao meu orientador George Teodoro por gentilmente fornecer as imagens de amostra de tecido utilizadas como instâncias de teste das implementações da operação de reconstrução morfológica e o código da biblioteca nscale.

Referências

[1] P.-E. Danielsson, "Euclidean distance mapping," *Computer Graphics and Image Processing*, vol. 14, pp. 227–248, 1980. [Online]. Available: https://api.semanticscholar.org/CorpusID:2459704

[2] G. Teodoro, T. Pan, T. M. Kurc, J. Kong, L. A. Cooper, and J. H. Saltz, "Efficient irregular wavefront propagation algorithms on hybrid cpu–gpu machines," *Parallel Computing*, vol. 39, no. 4, pp. 189–211, 2013. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167819113000343

[3] P. Karas, "Efficient Computation of Morphological Greyscale Reconstruction," in *Sixth Doctoral Workshop on Mathematical and Engineering Methods in Computer Science (MEMICS'10) – Selected Papers*, ser. Open Access Series in Informatics (OASIcs), L. Matyska, M. Kozubek, T. Vojnar, P. Zemcik, and D. Antos, Eds., vol. 16. Dagstuhl, Germany: Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2011, pp. 54–61. [Online]. Available: https://drops.dagstuhl.de/entities/document/10.4230/OASIcs.MEMICS.2010.54

[4] "Cuda c++ programming guide," Jan 2024. [Online]. Available: https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html