# ESTUDO DE NOVAS ABORDAGENS PARA OTIMIZAÇÃO DO PADRÃO DE COMPUTAÇÃO IRREGULAR WAVEFRONT PROPAGATION PATTERN

Mathias Oliveira, George Teodoro (Orientador)

***Abstract*—Neste trabalho, propomos uma modelagem alternativa para o padrão de computação Irregular Wavefront Propagation Pattern (IWPP), a modelagem IWPP MP. Além disso, propomos dois refinamentos para essa modelagem: o IWPP MP DQ e o IWPP MP UQ. Por meio de uma validação empírica, a implementação do IWPP MP UQ se mostrou mais rápida e mais eficiente no uso de memória do que as implementações baseadas no IWPP e no algoritmo SR na tarefa de computar a operação de reconstrução morfológica.**

## I. Introdução

A operação reconstrução morfológica é muito utilizada no processamento de imagens. Particularmente, essa operação é utilizada no processamento de imagens biomédicas. Por exemplo, ela é aplicada à imagens de amostras de tecido obtidas por meio de microscópios digitais com o intuito de identificar núcleos de células.

Embora essa operação seja relevante no processamento de imagens, ela é computacionalmente custosa, ou seja, na prática, para imagens muito grandes, essa operação toma uma grande quantidade de tempo para ser computada. Além disso, vale ressaltar que, muitas vezes, é desejado realizar a operação de reconstrução morfológica 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 uma implementação eficiente dessa operação.

Uma possível abordagem para diminuir o tempo de execução dessa operação 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, nesse trabalho, objetivamos construir implementações eficientes e capazes de tirar proveito dos recursos oferecidos pelas GPGPUs, para computar a operação de 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 a operação de reconstrução morfológica de forma eficiente é modelar a computação dessa operação 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, esse vizinho 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, a operação de reconstrução morfológica, a que desejamos abordar neste trabalho, pode ser computada por meio de pequenas modificações desse padrão de computação. Ademais, é

perceptível que esse padrão de computação pode ser facilmente executado de forma paralela.

### B. Reconstrução Morfológica

Antes de prosseguir, iremos descrever brevemente a operação de reconstrução morfológica. Essa operação recebe como entrada duas imagens, uma chamada de *marker* e a outra chamada de *mask*. O marker atribui para cada pixel da imagem um valor de 0 a 255. O mask determina para cada pixel um limitante superior (um número de 0 a 255) para o valor que o marker dele pode ter. O objetivo da reconstrução morfológica é fazer com que cada pixel propague o seu valor de marker para os seus pixels vizinhos que possuem um valor de marker menor que o seu. Observe que essa propagação deve ser feita de forma que o mask de cada pixel continue sendo respeitado. Neste trabalho iremos assumir que essas imagens têm tamanho $n \times n$, ou seja, são imagens quadradas.

Teodoro[1] apresentou 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 $(1, 1)$). Uma forma de paralelizar esse tipo de varredura é descrita e explorada em mais detalhes por Karas[2].

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 marker é menor do que o valor do seu mask e do que o valor do marker 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 seu valor de marker alterado, esse vizinho é adicionado à fila de sementes. Esse procedimento está descrito de forma mais precisa no Algoritmo 2. É importante ressaltar que nesse algoritmo, $N^+(p)$ (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 $p$).

É importante ressaltar que, 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 marker. Esse tipo de conexidade considerado neste trabalho está ilustrado na Figura 1. Vale ressaltar que as implementações construídas neste trabalho para computar a operação de reconstrução morfológica são uma pequena modificação do algoritmo descrito acima.

**Algoritmo 2** Reconstrução Morfológica (IWPP)

**Input:** I (Imagem mask $n \times n$)
**Input:** J (Imagem marker $n \times n$)
$S \leftarrow \emptyset$ ▷ 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**


Figura 1: Conjunto de vizinhos (conectividade) de um pixel considerado nas implementações da operação de reconstrução morfológica.

## 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. [1]. Neste artigo, são apresentadas, dentre outras coisas, implementações da operação de reconstrução morfológica, 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 a modelagem utilizada neste artigo para computar a operação de reconstrução morfológica foi apresentada na Seção I-B por meio do Algoritmo 2.

Ademais, no artigo *"Efficient Computation of Morphological Greyscale Reconstruction"* de Pavel Karas [2], 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. Esse algoritmo consiste em propagar o valor do marker de um pixel para seus vizinhos por meio de varreduras paralelas 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 paralela à lateral esquerda da imagem. Além disso, o Algoritmo 3 descreve o processamento realizado durante essa varredura. É fácil generalizar essa figura e esse algoritmo para varreduras que se iniciem em outras laterais da imagem.

Em ambos os trabalhos foram obtidos speedups relevantes em relação à execução sequencial da operação de reconstrução

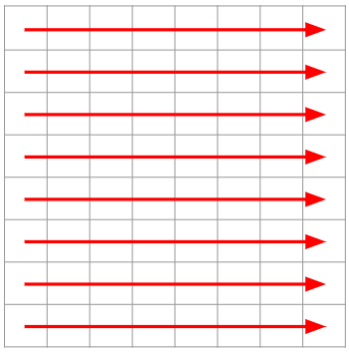


Figura 2: Sentido de propagação da informação em uma varredura paralela à lateral esquerda da imagem. Note que, cada linha pode ser processada por uma thread de forma independente das demais.

**Algoritmo 3** Varredura Paralela à Lateral Esquerda da Imagem Utilizada no Algoritmo SR.

**Input:** I (imagem mask $n \times n$
**Input:** J (imagem marker $n \times n$)
**for** $i = 1$ **to** $n$ **do**
  **for** $j = 1$ **to** $n - 1$ **do**
    **if** $J(i,j) > J(i,j+1)$ **then**
      $J(i,j+1) \leftarrow \min\{J(i,j), I(i,j+1)\}$
    **end if**
  **end for**
**end for**

morfológica. Dessa forma, neste trabalho, iremos propor uma modelagem alternativa para a operação de reconstrução morfológica com o objetivo de diminuir o tempo de execução dessa operação. Vale ressaltar que as implementações dessa nova modelagem que serão apresentadas neste trabalho foram desenhadas de forma a tirar proveito dos recursos de processamento paralelo providos pelas GPGPUs. Ademais, essas implementações foram feitas em C++ com o auxílio do framework CUDA (12.3.2).

## III. IWPP MP

Embora o algoritmo SR apresente speedups significativos quando comparado com implementações a nível de CPU [2], ele possui algumas ineficiências. O primeiro problema, o problema da granularidade, é que esse algoritmo sempre realiza quatro varreduras (uma iniciando de cada lateral da imagem) na imagem inteira, mesmo que apenas um pixel da imagem não tenha se estabilizado. Como consequência, o custo de cada iteração do algoritmo é alto e não diminui à medida que a imagem se aproxima de se tornar estável.

Outro problema do algoritmo SR é a localidade de memória. Assumindo que a imagem está armazenada na memória no padrão *row-major ordering*, o acesso à memória nas varreduras que se iniciam nas laterais superior e inferior da imagem não é otimizado. Isso ocorre pois o endereço de memória do pixel $(i,j)$ é $i * W + j$ ao passo que o endereço de memória do pixel $(i+1,j)$ é $(i+1) * W + j$, onde $W$ é a largura da imagem. Dessa forma, os acessos à memória realizados pelas threads de uma mesma warp não são *coalesced*, o que impacta a velocidade de acesso à memória do algoritmo.

Para resolver o problema da granularidade, o algoritmo IWPP usa uma fila de pixels que devem ser processados. Conforme descrito na Seção I-B, o IWPP realiza uma primeira passada na imagem para construir a fila de sementes. Em seguida, essa fila de sementes é dividida em sub-filas. Cada sub-fila dessas é atribuída a um thread block para ser processada de acordo com a fase de propagação do Algoritmo 2. Observe que esse processamento ocorre de forma independente das demais sub-filas.

É perceptível que por meio dessa estratégia, o algoritmo IWPP resolve o problema da granularidade. Entretanto, essa estratégia introduz novos problemas. O primeiro problema é que ao permitir que muitas threads processem a imagem de forma independente, muitos pixels são reprocessados várias vezes. Esse problema já foi apontado por Teodoro[1]. O segundo problema é que é necessária uma quantidade maior de memória para executar o IWPP. Isso ocorre pois é necessário armazenar as filas de megapixel. Note que, cada fila pode ter tamanho da ordem do tamanho da imagem. Conforme apontado por Teodoro[1], se o algoritmo ficar sem memória, o resultado da computação desse algoritmo é apenas uma solução parcial da operação de reconstrução morfológica. Note que, essa solução parcial pode ser reaproveitada como input do algoritmo IWPP, e esse procedimento pode ser repetido até que não ocorram *overflows* de memória. Entretanto, essa abordagem introduz vários *overheads* na computação da operação de reconstrução morfológica, o que aumenta o tempo de execução do algoritmo.

Um último problema do IWPP é que os problemas de acesso à memória presentes no algoritmo SR não são resolvidos. Note que a maioria dos pixels tem um vizinho na linha inferior e na linha superior e o acesso a esses pixels e, no IWPP, em geral, não é feito de forma sincronizada pelas threads de uma mesma warp. Além disso, com a abordagem do IWPP é possível que threads de uma mesma warp estejam processando pixels muito distantes na imagem. Dessa forma, o *coalecesing* de acessos à memória é negativamente impactado.

### A. Overview do IWPP MP

Dito tudo isto, nos apresentaremos o algoritmo IWPP MP. Para amenizar os problemas do algoritmo SR e do algoritmo IWPP apontados previamente, nós propomos agregar blocos de pixels em uma estrutura chamada de Megapixel (MP). Para ser mais preciso, nós iremos particionar a imagem em quadrados adjacentes de 32x32 pixels e executar o algoritmo IWPP na imagem particionada (essa imagem será chamada de grid de megapixels). Esse particionamento está ilustrado na Figura 3.

Uma vez que iremos executar o algoritmo IWPP no grid de megapixels, o algoritmo utilizará uma estrutura auxiliar para armazenar os megapixels que devem ser processados (a fila de megapixels). A primeira etapa do algoritmo consiste em construir uma fila com todos os megapixels da imagem. Uma vez que a fila inicial está pronta, nós executamos um kernel com vários thread blocks para processar os megapixels da fila. Esse procedimento está descrito no Algoritmo 4.

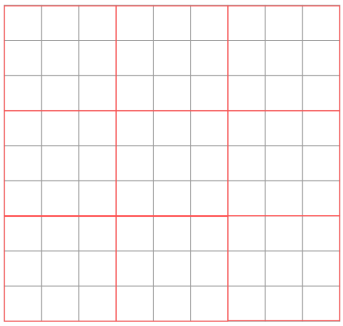


Figura 3: Exemplo de particionamento de uma imagem em megapixels. Nesse exemplo, uma imagem de 9x9 pixels foi particionada em megapixels de tamanho 3x3. Observe que, o grid de megapixels é composto apenas pelos megapixels, ou seja, é o grid de tamanho 3x3 indicado pelas bordas vermelhas.

Uma vez que um thread block obtém um megapixel da fila, ele copia os trechos das imagens marker e mask associados a esse megapixel para a shared memory da GPGPU. Em seguida, ele executa o algoritmo SR nos trechos das imagens que se encontram na shared memory. Uma vez que o megapixel está estabilizado, o resultado da computação realizada é escrito de forma atômica na imagem marker original que se encontra na memória global.

Um último detalhe sobre o IWPP MP é que para permitir que os valores dos pixels sejam transportados entre megapixels, o algoritmo na verdade trabalha com trechos de 34x34 pixels das imagens marker e mask. Isso significa que quando um thread block copia um megapixel para a shared memory, ele também copia os pixels adjacentes às laterais do megapixel. Nós nos referimos a esses pixels extras como a *ghost zone* do megapixel. Vale ressaltar que, empiricamente, verificamos que uma ghost zone de 1 pixel de largura é suficiente para que o IWPP MP seja correto e veloz. Dessa forma, nesse artigo sempre trabalharemos com ghost zones desse tamanho. A estrutura de um megapixel está ilustrada na Figura 4.

Como o IWPP MP executa o algoritmo SR na shared memory, ele também sofre do problema da granularidade. Entretanto, como o tamanho de um megapixel é pequeno e a execução do algoritmo SR ocorre na shared memory (uma memória muito rápida), o problema da granularidade é altamente mitigado no IWPP MP.

Ademais, como o IWPP MP agrega os pixels em blocos de pixels adjacentes, é fácil ver que o padrão de acesso à memória do IWPP MP é melhor que o do IWPP. Note que o processo de carregar um megapixel na shared memory pode ser facilmente *coalesced*. Isso ocorre pois o carregamento do megapixel pode ser feito linha a linha, onde cada thread lê um pixel da linha atual. Portanto, as únicas leituras que não podem ser *coalesced* são as leituras das laterais esquerda e direita da ghost zone. Note também que, embora as varreduras verticais do algoritmo SR não apresentem um bom padrão de acesso, no IWPP MP, essas varreduras ocorrem na shared memory que é uma memória muito mais rápida do que a memória global. Dessa forma, esse problema também é mitigado pelo IWPP MP.

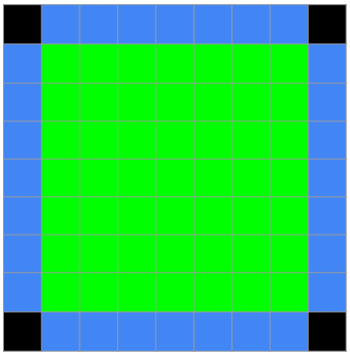


Figura 4: O bloco principal (megapixel de 32x32 pixels lido da fila global) está colorido de verde e as laterais desse megapixel estão coloridas de azul. Note que a lateral direita do megapixel (bloco principal) é a primeira coluna do megapixel (da sub imagem 32x32) à direita. É fácil ver que relações 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.

É importante notar que o IWPP MP também sofre com o problema de reprocessamento de megapixels. Entretanto, como, no IWPP MP, o IWPP é realizado no grid de megapixels, o volume de elementos processados repetidamente é significativamente menor do que o observado no IWPP. Além disso, o problema do IWPP utilizar muita memória para armazenar a fila de pixels também é mitigado. Isso ocorre pois a quantidade de memória necessária para armazenar a fila de megapixels é uma ordem de grandeza menor do que a necessária para armazenar a fila de pixels do IWPP. Note que, uma imagem quadrada cuja lateral tem tamanho $M$, tem $M^2$ pixels e $M^2/1024$ megapixels. Dessa forma, se são necessários GBs de memória para armazenar a imagem marker, serão necessários GBs de memória para armazenar a fila de pixels e serão necessários MBs de memória para armazenar a fila de megapixels.

### *B. Detalhes de implementação*

A implementação do IWPP MP utiliza uma fila global para armazenar os megapixels que devem ser processados. Dessa forma, para prevenir hazards de read-after-write e write-after-read, o acesso a essa fila é mediado por meio de uma mutex global. Ademais, cada thread block processa um megapixel da fila por vez. Essa abordagem foi escolhida pois possui uma implementação simples e resolve possíveis problemas de balanceamento de carga que podem surgir de um particionamento da fila de megapixels.

Junto a isto, os pixels das imagens marker e mask são armazenados na memória por meio do tipo char ao invés do tipo int, como é feito na implementação do IWPP. Dessa forma, a implementação do IWPP MP é capaz de processar

**Algoritmo 4** IWPP MP

```
Input: I (imagem mask n × n)
Input: J (imagem marker n × n)
__shared__ MP_marker[34][34]
__shared__ MP_mask[34][34]
Q ← all MPs of J
while Q is not empty do

▷ Queue Read
    P ← Q.pop()

▷ MP Fetch
    MP_marker ← copy_MP(marker, P)
    MP_mask ← copy_MP(mask, P)

▷ MP Processing
    SR_algorithm(MP_marker, MP_mask)

▷ MP Dump
    atomic_write_MP(J, P, MP_marker)

▷ Queue Write
    for P' neighbour of P do
        if P can propagate to P' then
            Q.push(P')
        end if
    end for
end while
```

imagens de resoluções maiores, pois a implementação utiliza exatamente um byte para armazenar o valor de um pixel.

Ademais, o IWPP MP executa por meio de thread blocks de 32 threads. Dessa forma, na etapa de executar o algoritmo SR na shared memory, cada thread do bloco pode fazer a varredura de uma linha (ou coluna) do megapixel de forma independente das demais linhas (ou colunas) do megapixel. O Algoritmo 4 descreve as etapas do IWPP MP executadas por cada thread block.

### *C. IWPP MP DQ*

Conforme descrito na Seção III-B, a implementação do IWPP MP utiliza uma fila global para computar a operação de reconstrução morfológica. Além disso, para controlar o acesso à fila de megapixels, é utilizada uma mutex global. Intuitivamente, esse design de fila é problemático, pois a mutex pode se tornar o bottleneck no processamento paralelo realizado pela GPGPU.

De fato, um profiling na implementação do IWPP MP revelou que quando o IWPP MP é executado com a maior ocupação possível de thread blocks, ao processar todas as instâncias de teste de resolução 4kx4k, cerca de 99% dos ciclos de computação da GPGPU são gastos com gerenciamento de fila. Mais precisamente, esses ciclos são gastos nas seções Queue Read e Queue Write do Algoritmo 4. Observe que esse comportamento é extremamente indesejável, pois ele mostra que a implementação não está sendo capaz de utilizar os recursos de processamento paralelo oferecidos pela GPGPU. Nesse contexto, nós propomos uma nova forma de estruturar a fila de megapixels.

Ao invés de ter uma única fila global, o IWPP MP Double Queue (IWPP MP DQ) utiliza duas filas: uma fila só de leitura e uma fila só de escrita. A fila de leitura é consumida pelo kernel de reconstrução morfológica, enquanto a fila de escrita é utilizada para armazenar os megapixels que serão processados na próxima iteração do algoritmo. Ao término da iteração atual, é executado um kernel que troca os ponteiros dessas duas filas. Dessa forma, na próxima iteração do algoritmo, o papel das posições de memória apontadas pelas filas é invertido. Intuitivamente, esse processo se repete até que nenhum megapixel precise ser processado. Esse comportamento está descrito no Algoritmo 5.

**Algoritmo 5** IWPP MP DQ

```
Input: I (imagem mask n × n)
Input: J (imagem marker n × n)
read_Queue ← all MPs of J
write_Queue ← ∅
while read_Queue is not empty do
    morph_recon_kernel(I, J, read_Queue, write_Queue)
    swap_queues(read_Queue, write_Queue)
end while
```

Por meio desse novo design de fila, a operação de leitura se torna mais rápida, porque os megapixels que serão processados em uma iteração estão bem definidos antes do kernel de reconstrução morfológica ser executado. Isso implica que é possível dividir rapidamente a fila de leitura em sub-filas contíguas de tamanhos que diferem por no máximo 1 megapixel e atribuir cada sub-fila a um thread block distinto. Note que, como o tamanho de cada sub-fila difere por no máximo um megapixel, vários problemas de balanceamento de carga são mitigados. Ademais, é perceptível que o processo de ler da fila de leitura não precisa mais ser realizado de forma atômica. Junto a isso, como essas sub-filas são contíguas, as operações de leitura podem ser *coalesced*. Dessa forma, na versão IWPP MP DQ foi introduzido o processamento em batches de megapixels por meio de um buffer na shared memory.

Para reduzir os custos da operação de escrita na fila de escrita, nós introduzimos um buffer na shared memory para armazenar os megapixels que serão escritos na fila de escrita. Como o algoritmo utiliza uma fila só de escrita, para um thread block alocar uma quantidade de posições na fila, basta aumentar o contador de elementos da fila de forma atômica. Em seguida, o thread block pode escrever seus registros nas posições alocadas. Note que, como um thread block copia todo o buffer para posições contíguas de memória da fila de escrita, as operações de escrita realizadas pelas threads do thread block podem ser *coalesced*.

### *D. IWPP MP UQ*

Um problema do IWPP que persistiu no IWPP MP DQ é o reprocessamento de megapixels. Um profiling no conteúdo das

filas processadas pelo IWPP MP DQ apontou que, ao processar as instâncias de teste de tamanho 4kx4k, o conteúdo das filas processadas pelo algoritmo era extremamente redundante, ou seja, haviam megapixels que eram processados várias vezes dentro de uma mesma fila.

Dessa forma, foi proposta uma extensão para o IWPP MP DQ. O IWPP MP Unique Queues (IWPP MP UQ) estende o IWPP MP DQ com um processamento extra antes da operação de troca de filas. Nesse processamento extra, os megapixels repetidos são removidos da fila que será processada na próxima iteração. O comportamento do IWPP MP UQ está descrito no Algoritmo 6.

**Algoritmo 6** IWPP MP UQ

```
Input: I (imagem mask n × n)
Input: J (imagem marker n × n)
read_Queue ← all MPs of J
write_Queue ← ∅
while read_Queue is not empty do
    morph_recon_kernel(I, J, read_Queue, write_Queue)
    write_Queue ← extract_unique_MPs(write_Queue)
    swap_queues(read_Queue, write_Queue)
end while
```

Note que, a princípio, não é óbvio que o reprocessamento de megapixesl de fato é um problema. É possível que o reprocessamento computado em uma iteração permita que mais informação se propague pela imagem. Dessa forma, remover o reprocessamento de megapixels pode fazer com que sejam necessárias mais iterações do algoritmo para computar a operação de reconstrução morfológica. Entretanto, foi verificado empiricamente que, em imagens de altas resoluções, ou seja, imagens onde essa redundância é muito expressiva, o IWPP MP UQ computa mais rápido a operação de reconstrução morfológica do que o algoritmo IWPP MP DQ.

## IV. Metodologia

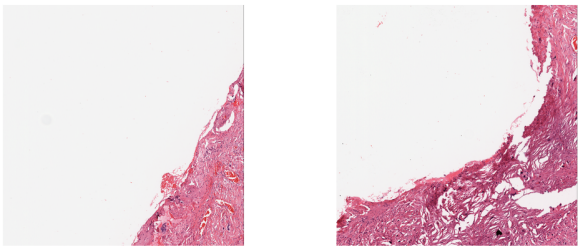
Foi realizada uma avaliação empírica para comparar as versões desenvolvidas do algoritmo IWPP MP com os algoritmos IWPP e SR. É importante ressaltar que as implementações da operação de reconstrução morfológica baseadas no IWPP e no algoritmo SR utilizadas nessa comparação foram as implementações apresentadas por Teodoro em [1]. Todos os experimentos apresentados neste artigo foram realizados em uma única máquina composta por uma CPU Ryzen 9 5950X com 64 GB de memória e uma GPGPU RTX 4090.

Muitos dos experimentos apresentados neste artigo consistiram de coletar o tempo de execução de diferentes implementações da operação de reconstrução morfológica ao processar diferentes instâncias de teste. Dessa forma, é importante destacar que os tempos coletados se referem apenas ao tempo de computação da operação de reconstrução morfológica. Dessa forma, tempos de abertura, escrita de arquivos e transferência do tipo host-device não foram coletados.

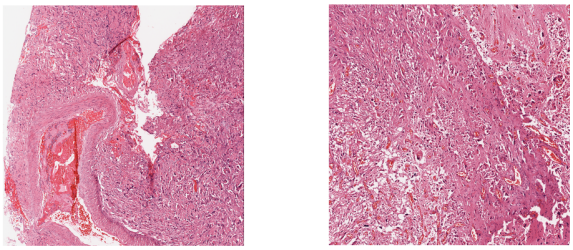
Ademais, foi considerado como o tempo de execução da implementação $I$ para resolver a instância de teste $T$ a mediana dos tempos de cinco execuções da implementação $I$ processando a instância $T$. Além disso, vale ressaltar que, após cada execução das implementações testadas, a corretude da solução produzida foi verificada. Dessa forma, nos experimentos realizados, todos os algoritmos retornaram a mesma solução (o mesmo valor para cada pixel) ao processar uma mesma instância de teste.

### A. Instâncias de teste

As instâncias de teste utilizadas nos nossos experimentos foram imagens de amostra de tecido. Como imagens de amostra de tecido podem variar na sua estrutura, nós avaliamos quatro casos em que se varia a proporção da imagem que é coberta pela amostra de tecido. Mais precisamente, utilizamos quatro percentuais de cobertura da imagem: 25%, 50%, 75%, e 100%. Esses percentuais de cobertura estão ilustrados na Figura 5. É importante ressaltar que essas imagens possuem resolução 4kx4k.



(a) 25% Coverage (b) 50% Coverage



(c) 75% Coverage (d) 100% Coverage

Figura 5: Imagens de amostra de tecido utilizadas para extrair os inputs (imagem marker e mask) da operação de reconstrução morfológica. Vale ressaltar que cada uma dessas imagens tem resolução 4kx4k.

A partir dessas imagens de amostra de tecido, foi realizado um pré-processamento para extrair de cada uma dessas imagens uma imagem marker e uma imagem mask. Observe que essas imagens marker e mask que são utilizadas como entrada do algoritmo de reconstrução morfológica. Ademais, também foram geradas instâncias de teste sintéticas de resoluções maiores. Essas imagens sintéticas consistem na concatenação de uma instância de teste com ela mesma na forma de um grid quadrado. Esse processo está ilustrado na Figura 6. Dessa forma, foi possível verificar o comportamento das implementações ao processar imagens de resolução 4kx4k, 8kx8k, 12kx12k, . . . , 28kx28k, 32kx32k, 64kx64k e 100kx100k.

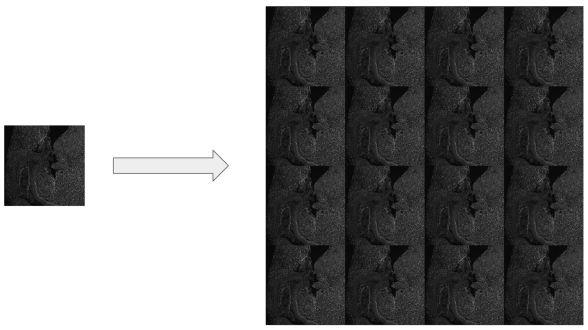


Figura 6: Exemplo de instância de teste sintética. Neste caso, o mask da instância de teste de 75% foi concatenado com ele mesmo na forma de um grid 4x4. Note que, como a instância de teste de 75% tem resolução de 4kx4k, essa instância de teste sintética tem resolução de 16kx16k.

Note que, o intuitivo seria que a resolução da última imagem fosse de 128kx128k. Entretanto, para armazenar tal imagem na GPGPU seria necessário pelo menos 37 GB de memória. Como nossa GPGPU possui apenas 24 GB de armazenamento, optamos por encerrar os testes na resolução de 100kx100k.

## V. Análise e Resultados

### A. Comparação entre Algoritmos

Para comparar o desempenho das implementações dos algoritmos SR, IWPP e IWPP MP na tarefa de computar a operação de reconstrução morfológica, foram realizados experimentos para medir o tempo de execução dessas implementações ao processar o marker e o mask extraídos das imagens de amostra de tecido. Os resultados apresentados nesta seção foram realizados conforme descrito na Seção IV.

Mais precisamente, as implementações avaliadas foram: a implementação a nível de GPGPU do IWPP MP UQ, uma implementação a nível de CPU (multicore) do IWPP (IWPP (CPU)), e implementações a nível de GPGPU dos algoritmos SR e IWPP. É importante ressaltar que quase todas essas implementações possuem parâmetros que influenciam no seu tempo de execução. Dessa forma, foi realizada uma análise com o objetivo de encontrar o melhor conjunto de parâmetros para cada uma das implementações avaliadas. Note que todos os tempos de execução apresentados se referem exclusivamente ao tempo de execução das implementações com os melhores parâmetros verificados.

Os tempos de execução coletados estão ilustrados na Figura 7. Claramente, por meio desse gráfico, a implementação IWPP MP UQ se mostrou mais rápida do que as implementações dos algoritmos SR e IWPP CPU. Mais precisamente, a implementação IWPP MP UQ apresenta um speedup de pelo menos 8.1x em relação à implementação IWPP CPU e um speedup de pelo menos 16.7x em relação à implementação SR em todas as instâncias de teste baseadas em amostras de tecido. Vale ressaltar que essas instâncias de teste possuem resolução 4kx4k.

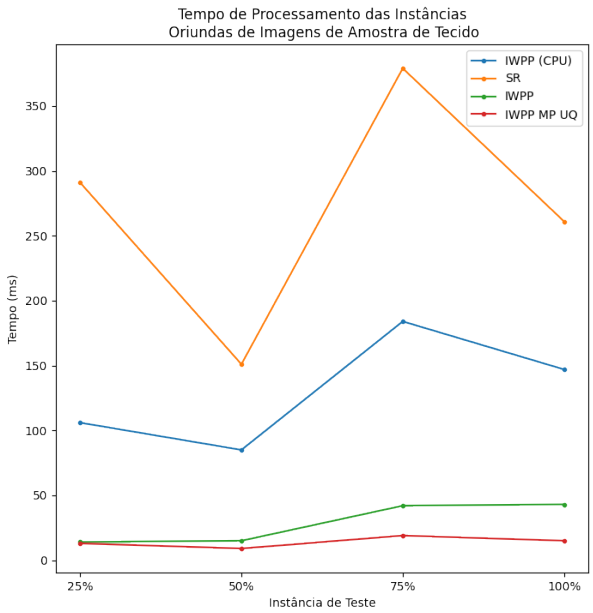



Figura 7: Gráfico que mostra o tempo de execução mediano das implementações IWPP (CPU), SR, IWPP, IWPP MP UQ ao processar as instâncias de teste obtidas por meio de imagens de amostra de tecido. Vale ressaltar que essas imagens possuem resolução 4kx4k. A implementação com os menores tempos de execução foi a IWPP MP UQ, embora para a imagem de 25%, a implementação IWPP apresentou um tempo de execução análogo ao da implementação IWPP MP UQ.

Entretanto, a implementação IWPP apresenta tempos de execução semelhantes aos do IWPP MP UQ quando consideramos as instâncias de teste de 25% e 50%. Mais precisamente, o IWPP MP UQ apresentou um speedup de 1.07x e 1.66x ao processar as instâncias de teste de 25% e 50%, respectivamente, em relação à implementação IWPP. Por outro lado, ao processar as instâncias de teste de 75% e 100%, o IWPP MP UQ apresentou speedups de 2.21x e 2.86x, respectivamente, em relação à implementação IWPP.

Acreditamos que os fatores que justificam esses resultados observados são exatamente os fatores que motivaram o IWPP MP UQ. Isto é, uma melhor interação com a memória, ou seja, um uso mais local e *coalesced* da memória global, o uso da shared memory, uma memória mais rápida, para executar o algoritmo SR e mitigação do problema de granularidade presente no algoritmo SR.

### B. Teste de Escalabilidade

Um outro experimento de interesse deste trabalho é verificar o comportamento das implementações da operação de reconstrução morfológica baseadas nos algoritmos SR, IWPP e IWPP MP, ao processar imagens de resoluções maiores. Dessa forma, foram realizados experimentos para coletar os tempos de execução dessas implementações ao processar imagens sintéticas de amostra de tecido de resoluções que variam de 4kx4k até 100kx100k.

Mais precisamente, as implementações avaliadas foram: a implementação a nível de GPGPU do IWPP MP UQ, a implementação a nível de GPGPU do IWPP MP DQ, uma implementação a nível de CPU (multicore) do IWPP (IWPP (CPU)) e implementações a nível de GPGPU dos algoritmos SR e IWPP. Note que esses experimentos foram realizados conforme descrito na Seção IV.

Com base nos resultados da Seção V-A e em outras análises realizadas ao longo deste trabalho, foi constatado que a instância de teste de amostra de tecido que gera a maior carga de trabalho para as implementações é a instância de 75%. Dessa forma, os experimentos de escalabilidade foram realizados com imagens sintéticas que consistem na concatenação da instância de teste de 75% com ela mesma no formato de um grid quadrado. Em outras palavras, o marker (mask) sintético utilizado nos experimentos é composto pela concatenação do marker (mask) da instância de teste de 75% com ele mesmo na forma de um grid quadrado. Essa operação está ilustrada na Figura 6.

É importante ressaltar que as implementações baseadas no algoritmo IWPP, por motivos de implementação, não são capazes de processar imagens que possuem resolução maior que 32kx32k. Ademais, embora a implementação baseada no algoritmo SR fosse capaz de processar imagens de resolução maior que 32kx32k, o comportamento desse algoritmo foi verificado apenas para as instâncias sintéticas de tamanho até 32kx32k, pois esses testes já foram suficientes para ilustrar o padrão de escalabilidade da implementação desse algoritmo. Dito isto, os resultados obtidos estão dispostos nas Figuras 8 e 9.

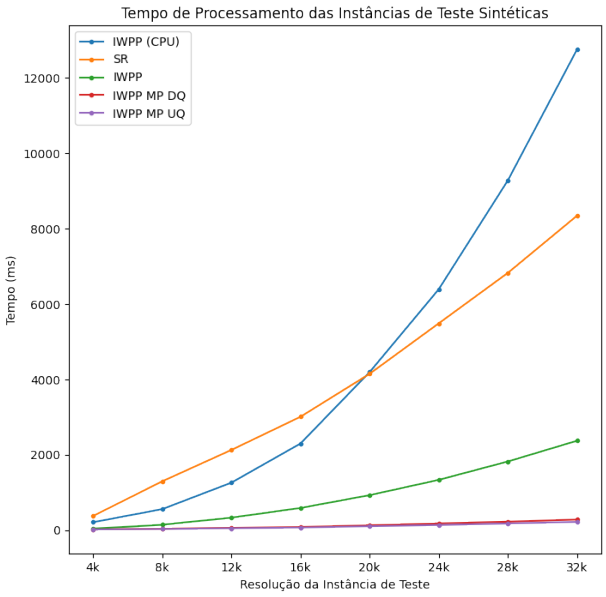



Figura 8: Gráfico que mostra a variação do tempo de execução das implementações IWPP (CPU), SR, IWPP, IWPP MP DQ e IWPP MP UQ à medida que a resolução da instância de teste aumenta. É perceptível que as implementações que apresentam a melhor escalabilidade são as implementações IWPP MP UQ e IWPP MP DQ.

Por meio da Figura 8 é perceptível que as implementações que apresentam o melhor padrão de escalabilidade são as implementações baseadas no IWPP MP. Ademais, é perceptível que quanto maior a resolução da instância de teste, maior é o speedup observado da implementação IWPP MP UQ em relação às outras implementações. Esse comportamento está ilustrado na Figura 9. Mais precisamente, para imagens de resoluções maiores, os speedups observados da implementação IWPP MP UQ em relação às implementações IWPP CPU, IWPP e SR é de aproximadamente 57x, 37x e 10x, respectivamente.

Novamente, acreditamos que os fatores que justificam o melhor desempenho da implementação do IWPP MP UQ em relação às implementações IWPP CPU, IWPP e SR são os fatores que motivaram o IWPP MP UQ. Ou seja, uma melhor interação com a memória, ou seja, um uso mais local e *coalesced* da memória global, o uso da shared memory para executar o algoritmo SR e a mitigação do problema de granularidade presente no algoritmo SR.

Ademais, a Figura 10 compara os tempos de execução das duas implementações do IWPP MP testadas. Conforme discutido na Seção III-D a diferença entre essas duas versões é que o IWPP MP UQ remove os megapixels repetidos das filas que serão processadas e, consequentemente, esse algoritmo processa uma quantidade menor de megapixels do que o IWPP MP DQ.

Por meio da Figura 10 fica claro que, para imagens de resoluções menores, o tempo de processamento da implementação do IWPP MP UQ é análogo ao tempo de processamento da implementação do IWPP MP DQ. Entretanto, ainda nesta figura, é perceptível que, para imagens de resoluções maiores, a etapa de remoção de megapixels redundantes da fila de megapixels é responsável por uma redução significativa no tempo de execução do IWPP MP UQ em relação ao IWPP MP DQ. A Figura 11 mostra o speedup que o IWPP MP UQ apresenta em relação ao IWPP MP DQ. Essa figura reforça o comportamento discutido e informa que, para a imagem de maior resolução testada, a imagem de 100kx100k, o IWPP MP UQ apresenta um speedup de 1.42x em relação ao IWPP MP DQ.

Nós acreditamos que esse comportamento é explicado devido à capacidade da RTX 4090 de realizar um processamento paralelo massivo. Para confirmar essa hipótese, nós executamos os dois algoritmos cinco vezes e coletamos a quantidade de megapixels que cada algoritmo processou. Na Tabela I está apresentada a mediana dos cinco tamanhos de filas coletados ao processar as 4 instâncias de teste oriundas de imagens de amostra de tecido (4kx4k) e a instância de teste sintética de resolução 100kx100k.

Por meio da Tabela I é perceptível que o IWPP MP UQ consegue computar a operação de reconstrução morfológica processando uma quantidade significativamente menor de megapixels. Note que o IWPP MP UQ pode processar de 41% a 54% menos megapixels do que o IWPP MP DQ. Entretanto, na imagem de 75%, a imagem onde essa diferença é mais expressiva, o IWPP MP UQ processa cerca de 146 mil

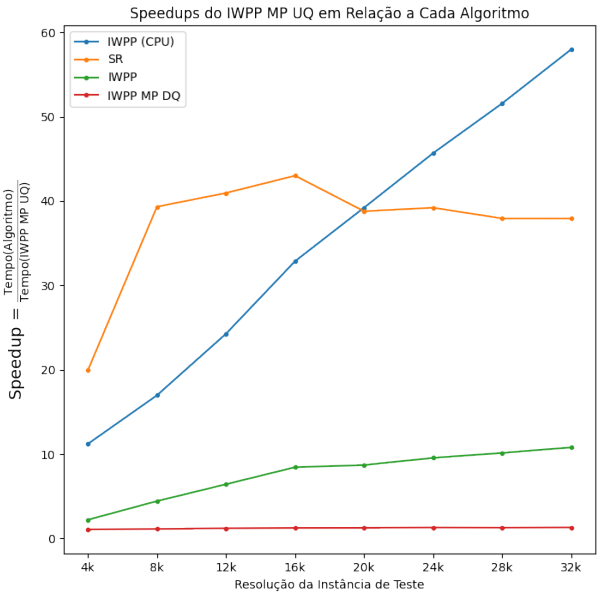



Figura 9: Gráfico que mostra a variação do speedup da implementação IWPP MP UQ em relação às implementações IWPP (CPU), SR, IWPP, IWPP MP DQ à medida que a resolução da instância de teste aumenta. É perceptível que existe uma tendência do speedup observado aumentar à medida que o tamanho da instância de teste aumenta.

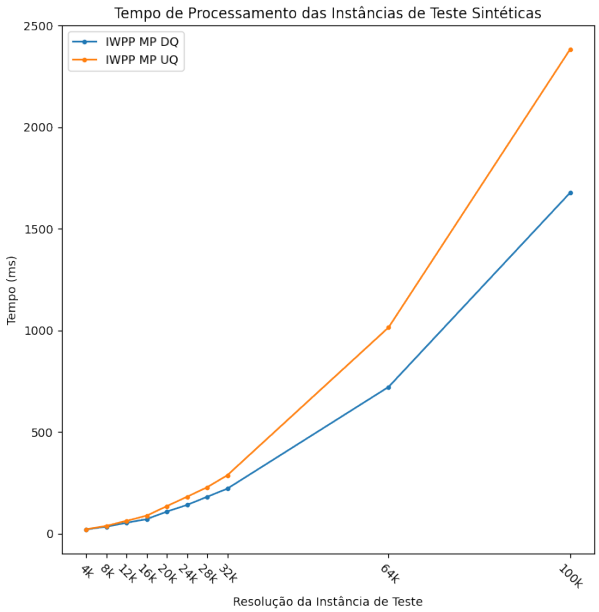



Figura 10: Gráfico que mostra o tempo de execução em (ms) das implementações ao processar as instâncias de teste sintéticas. É perceptível que para instâncias de teste de resoluções maiores, a implementação IWPP MP UQ apresenta um desempenho melhor do que a implementação IWPP MP DQ. Ademais, é possível observar que para imagens de resoluções menores, o tempo de execução dessas duas implementações é análogo.

| Test Instance | IWPP MP DQ | IWPP MP UQ |
|---|---|---|
| 25%-4k | 92453 | 54140 |
| 50%-4k | 111193 | 58495 |
| 75%-4k | 268794 | 126456 |
| 100%-4k | 254235 | 115606 |
| 75%-100k | 123893815 | 58629756 |

Tabela I: Profiling da quantidade de megapixels processados pelas implementações dos algoritmos IWPP MP DQ e IWPP MP UQ. Aqui está exposta a mediana do número de megapixels processados em cinco execuções de cada implementação.

megapixels a menos que o IWPP MP DQ. Nós acreditamos que esse número não é grande o suficiente para observarmos uma diferença significativa no tempo de execução desses dois algoritmos.

Por outro lado, ao processar a instância 75%-100k, o IWPP MP UQ processa cerca de 52% de megapixels a menos que o IWPP MP DQ. Além disso, essa diferença corresponde a cerca de 65 milhões de megapixels. Dessa forma, mesmo a capacidade massiva de processamento paralelo da GPGPU RTX 4090 não é suficiente para ocultar no tempo de execução o problema do reprocessamento de megapixels presente no algoritmo IWPP MP DQ.

## VI. Código

Os códigos das implementações desenvolvidas para computar a operação de reconstrução morfológica por meio dos algoritmos IWPP MP, IWPP MP DQ e IWPP MP UQ podem ser encontrados em https://github.com/oliveira-mathias/IWPP-MP.

## VII. Conclusão

Neste trabalho apresentamos uma nova modelagem para o padrão de computação Irregular Wavefront Propagation Pattern (IWPP), a modelagem IWPP MP. Embora essa modelagem seja genérica e permita a computação de várias operações, neste trabalho nós nos restringimos a estudar formas de utilizar essa nova modelagem para computar de forma eficiente a operação de reconstrução morfológica.

Por meio dos resultados apresentados na Seção V-A, a implementação do IWPP MP se mostrou mais eficiente do que a implementação baseada no IWPP ao processar todas as instâncias de teste oriundas de amostras de tecido. Os resultados mais notórios foram speedups de 2.21x e 2.86x observados ao processar as instâncias de 75% e 100%, respectivamente.

Na Seção V-B, argumentamos que a implementação IWPP MP UQ é a que apresenta melhor escalabilidade ao processar imagens de resoluções maiores. Para imagens de resoluções maiores, por exemplo, a instância sintética de 32kx32k, o IWPP MP UQ apresentou um speedup de 10.7x em relação à implementação baseada no IWPP.

Além disso, ainda nesta seção foi possível confirmar que, de fato, o processamento redundante realizado pelo IWPP MP

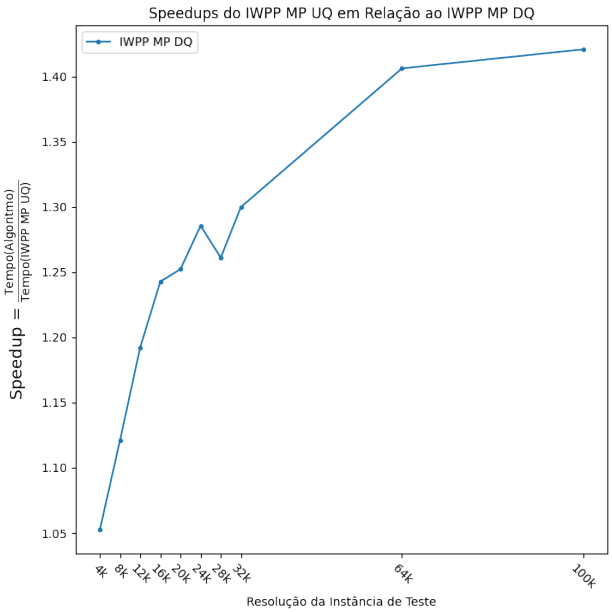



Figura 11: Gráfico que mostra a variação do speedup da implementação IWPP MP UQ em relação à implementação IWPP MP DQ à medida que a resolução da instância de teste aumenta. É perceptível que existe uma tendência do speeup observado aumentar à medida que o tamanho da instância de teste aumenta. Note também que os speedups são relativamente pequenos para imagens de resoluções menores.

DQ é um problema que aumenta o tempo de execução. Para ser mais específico, para imagens de resolução maiores, por exemplo, a instância sintética de 100kx100k, o IWPP MP UQ processa cerca de 52% menos megapixels do que o IWPP MP DQ e, simultaneamente, apresenta um speedup de 1.42x.

Dessa forma, o IWPP MP UQ se apresenta como uma alternativa viável e mais eficiente ao IWPP, uma vez que a implementação deste algoritmo apresentou speedups significativos em relação às implementações baseadas no IWPP e no algoritmo SR e, em geral, apresenta um uso de memória menor do que as implementações baseadas no IWPP.

## Agradecimentos

Esse trabalho não seria possível sem o código das implementações da operação de reconstrução morfológica baseadas nos algoritmos IWPP e SR, que foi gentilmente disponibilizado pelo meu orientador George Teodoro.

## Referências

[1] 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

[2] 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

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