Parallel CUDA Based Implementation of Gaussian Pyramid Image Reduction

Cristiane B. R. Ferreira*, Fabrízzio A. A. M. N. Soares*, Wellington S. Martins*,
Ronaldo M. Costa*, William D. Ferreira*, Thyago P. Carvalho*,
Rafael T. Parreira*, Gelson da Cruz Jr.†
*Instituto de Informática - INF, Alameda Palmeiras, Quadra D, Campus Samambaia – Goiânia – GO – Brazil
†Escola de Engenharia Elétrica, Mecânica e de Computação - EMC, Praça Universitária, Bloco A – Goiânia – GO – Brazil
Universidade Federal de Goiás
{cristiane,fabrizzio,wellington,ronaldocosta,william,thyagoperes,rafaeltomaz}@inf.ufg.br, gcruz@emc.ufg.br

***Abstract*—A digital image can be represented in several dimensions. In automatic identification process of buildings, people and others objects, resolution can determine efficiency and efficacy of recognition algorithms. Image dimension reduction is useful to minimize computational effort and avoiding adaptation of object detection algorithms. However, the image reduction process also requires high effort and can spend almost same amount of saved time and effort. This paper proposes a parallel implementation of the Gaussian Pyramid multiresolution approach for image reduction of any image and experimental implementation in CUDA. The implementation was performed in 3 different GPU's and compared with traditional approach in a regular personal computer. Experiments show significative time processing reduction which means high efficiency and potential for adoption of our parallel solution in a image processing chain.**

***Keywords*—Pyramid Gaussian, Parallel programming, Speedup**

I. Introdução

Uma imagem digital pode ser encontrada em diversas dimensões, sendo bastante comuns formatos entre 0,3 e 15 *Megapixels*. Este tipo de imagem pode ser facilmente apresentado em uma tela de computador usando algoritmos que reduzem as dimensões das imagens descartando-se linhas e colunas alternadamente de modo a se ter uma versão possível de ser exibida na área desejada. A redução de resolução também é útil quando deseja-se reduzir o esforço computacional ou evitar a adaptação de algoritmos para detecção de objetos, dentre outros. A abordagem de descarte de linhas e colunas, embora seja útil para visualização através do olho humano, pode não ser boa para fins de processamento, visto que muitos dados relevantes podem ser descartados, comprometendo os resultados dos algoritmos.

Atualmente, existe uma categoria de imagens chamadas Imagens Gigapixel que possuem volume de dados da ordem de 0,3 a 300 *Gigapixels*. Dessa forma, a redução da dimensão da imagem pode auxiliar em aplicações com tarefas de identificação de objetos, pessoas e edificações, visto que a quantidade de informações a serem trabalhadas é menor, possibilitando o estudo em vários níveis de resolução, a fim de verificar se informações necessárias às identificações mencionadas anteriormente, ainda estão presentes em outras resoluções da imagem.

Diversos trabalhos têm utilizado abordagem paralela para implementar algoritmos tradicionais de processamento de imagens. Em 1999, Yip *et al.* [8] propuseram um algoritmo paralelo para estimação de derivadas parciais de imagens 2D e 3D em arquiteturas de memória distribuída MIMD (*Mútiplas Instruções Múltiplos Dados*). Neste trabalho foi explorada a característica de separabilidade do filtro Gaussiano, sendo que o algoritmo foi implementado em diversas fases correspondentes ao filtro separado. Foi utilizado um sistema Paragon da Intel com mais de 100 processadores e foram obtidos *speedup's* significativos.

Em 2009, Kraus [5] desenvolveu uma abordagem paralela para *Pyramidal blurring* que consistia de operações de redução e sintetização de pixels utilizando a interpolação bilinear por filtragem cúbica B-spline. Os experimentos foram realizados apenas em imagens de 1.024x1.024 *pixels* com 16 bits RGBA em uma Geforce 7900 GTX. Os experimentos realizados apresentaram $0,85$ ms para o processamento de 1 nível.

Em [7], Olmedo *et al.* apresenta uma abordagem de processamento ponto a ponto de imagens digitais usando tanto uma implementação sequencial quanto outra com computação paralela, enfatizando, em particular, operações de mudanças de contraste, escurecimento, clareamento e limiarização em imagens, por exemplo, com o objetivo de reduzir o tempo de execução de tais operações na abordagem paralela. Na abordagem paralela apresentada do referido trabalho, foram usados 32×32 blocos (1024 blocos no total), cada um deles possuindo *largura* x *altura da imagem* / 1024 *threads*. As imagens utilizadas são de dimensões de 256×256, 512×512, 1.024×1.024, 1.800×1.400 e 4.000×3.000 *pixels*. Os experimentos mostraram que os filtros implementados em CUDA apresentaram melhores resultados quando comparados com funções do OpenCV e implementações em linguagem C de forma sequencial. Com exceção do filtro negativo com imagens de baixas resoluções, o OpenCV obteve melhores resultados. Porém, no caso de imagens de alta resolução, a performance da implementação paralela em CUDA foi melhor.

Elboher e Werman (2012) [3] desenvolveram um método paralelo para convoluir imagens com filtros Gaussianos a partir de operações de imagens integrais ao longo das colunas e linhas das imagens. Segundo os autores, foram obtidos resultados melhores que a convolução Gaussiana sequencial prevista no estado da arte, enquanto a acuidade de aproximação similar foi preservada.

Recentemente, em 2016, Li *et al.* [6] propuseram um

algoritmo paralelo para casamento (*matching*) de imagens RAW[1] de alta resolução. Foi utilizada uma arquitetura de processador de múltiplos núcleos para *matching* e imageamento. Os experimentos demonstraram que o método proposto pôde adquirir casamentos mais robutos na maioria dos casos e em menor tempo quando comparados a métodos tradicionais como *Binary robust independent elementary features* (BRIEF), *Oriented FAST and rotated BRIEF* (ORB), *Binary robust invariant scalable keypoints* (BRISK) e *Fast retina keypoint* (FREAK).

O presente trabalho propõe uma abordagem paralela para a Pirâmide Gaussiana, baseada em arquiteturas M-SIMD (GPU) com a finalidade de acelerar o processamento de imagens quaisquer como passo inicial para, no futuro, tal implementação ser ampliada e adaptada para o processamento de imagens Gigapixel. É importante ressaltar que a abordagem multirresolução da Pirâmide Gaussiana é implementada nas versões sequencial e paralela, que serão detalhadas posteriormente.

Este artigo encontra-se organizado da seguinte maneira: a Seção II descreve a abordagem multirresolução da Pirâmide Gaussiana. A Seção III apresenta os pseudocódigos relacionados às abordagens sequencial e paralela da Pirâmide Gaussiana, seguida da Seção IV que traz as implementações relativas às abordagens propostas, desenvolvidas para realizar os experimentos. Já a Seção V apresenta os experimentos realizados, a medida de desempenho utilizada e os resultados alcançados. Por fim, a Seção VI apresenta as conclusões obtidas a partir da análise dos resultados.

II. A Pirâmide Gaussiana

A abordagem multirresolução da Pirâmide Gaussiana tem como objetivo a redução da resolução de uma imagem para vários níveis. Por exemplo, se a imagem de origem tiver resolução 64×64 *pixels*, num primeiro nível de redução, a imagem de destino gerada terá resolução de 32×32 *pixels* no nível seguinte, caindo pela metade em cada nível subsequente.

A Pirâmide Gaussiana se inicia a partir de uma imagem de origem g_0 que contenha C colunas e L linhas de *pixels*. Cada *pixel* representa a intensidade de luz de um ponto da imagem correspondente a um inteiro I entre 0 e $K-1$[2]. Essa imagem se torna o nível inferior ou zero da Pirâmide Gaussiana. O nível 1 da Pirâmide contém a imagem g_1, que é uma versão reduzida ou filtrada passa-baixa de g_0. Cada valor do nível 1 é computado como uma média ponderada dos valores no nível zero dentro de uma janela 5×5. Cada valor do nível 2, representado por g_2, é obtido dos valores do nível 1 aplicando o mesmo padrão de pesos. Para melhor entendimento, uma representação gráfica do processo em uma dimensão é apresentado na Figura 1.

A função Gaussiana, conhecida também como distribuição normal, pode ser vista na Figura 2(a), a qual pode ser referenciada como "curva em forma de sino". Além disso, pode-se verificar a máscara Gaussiana usada nesse processo na Figura 2(b).

[1]Formato bruto, geralmente refere-se à formatos próximos às características do sensor.

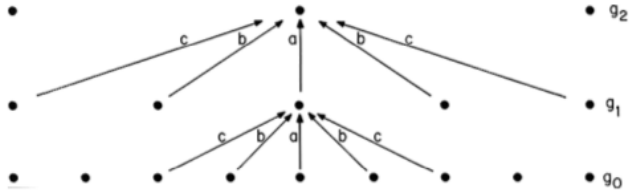
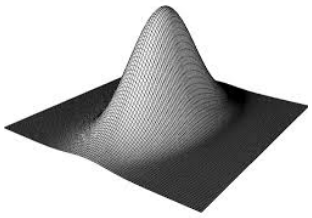


Figura 1. Representação gráfica do processo que gera uma Pirâmide Gaussiana em uma dimensão[2].



1/256 x

1	4	6	4	1
4	16	24	16	4
6	24	36	24	6
4	16	24	16	4
1	4	6	4	1

(a) Função Gaussiana (b) Máscara Gaussiana 5x5

Figura 2. Representação gráfica e matricial da Função Gaussiana, respetivamente.

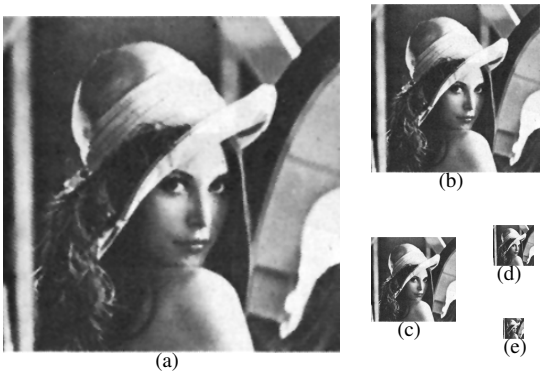


Figura 3. Aplicação da abordagem de Pirâmide Gaussiana em uma imagem bidimensional. a) 100%, b)50%, c)25%, d) 12.5% e e) 6.25%.

O *kernel* da Pirâmide Gaussiana é calculado conforme a Equação 1.

$$im_{l+1}(i,j,k) = \sum_{m=-2}^{2} \sum_{n=-2}^{2} gm(m,n) \cdot im_l(2i+m, 2j+n, k) \quad (1)$$

onde im_{l+1} e im_l são as imagens de destino e origem, respectivamente, gm é a máscara Gaussiana, i, j e k representam linha, coluna e canal de cor da imagem, respectivamente.

A aplicação da abordagem multirresolução de Pirâmide Gaussiana em uma imagem bidimensional pode ser percebida na Figura 3, repetida em 5 níveis de decomposição, a partir do nível 0 (Figura 3(a)), que representa a imagem original.

III. Pirâmide Gaussiana Sequencial e Paralela

A. Abordagem Sequencial

O PSEUDOCÓDIGO I apresenta a parte principal da abordagem da Pirâmide Gaussiana, considerando sua construção padrão, ou seja, sequencial, para uma imagem bidimensional.

PSEUDOCÓDIGO I. LISTAGEM DO PSEUDOCÓDIGO DA ABORDAGEM SEQUENCIAL

```
   function Reduce(im_l, gm): im_{l+1}
   {Reduz a dimensão da imagem im_l para o próximo nível da pirâmide}
   Entrada  im_l[1..linhas, 1..colulas, 1..canais]: Byte
            gm[−2..2, −2..2]: Real
   Saída    im_{l+1}[1..linhas/2, 1..colunas/2, 1..canais]: Byte
1  begin
2    for k ← 1 to canais do
3    begin
4      for i ← 1 to linhas/2 do
5      begin
6        for j ← 1 to colunas/2 do
7        begin
8          newPixel ← 0;
9          for m ← −2 to 2 do
10         begin
11           for n ← −2 to 2 do
12           begin
13             newPixel ← newPixel
14                       + im_l[2*i+m, 2*j+n, k]
15                       * gm(m,n);
16           end
17           im_{l+1}[i,j,k] ← newPixel;
18         end
19       end
20     end
21   end
22 end
```

No pseudocódigo apresentado, im_l e im_{l+1}representam a imagem de origem e de destino, respectivamente, e *gm* é a máscara Gaussiana, conforme representação da Figura 2(b).

As linhas 2 a 6 referem-se às estruturas de repetição para percorrer cada canal, linha e coluna da imagem, respectivamente. As linhas de 7 a 20 referem-se ao cálculo da convolução sobre um único *pixel* da imagem de destino. Na convolução, quando aplicada a *pixels* próximos da borda da imagem, não há *pixels* vizinhos suficientes para realizar seu cálculo. Uma das abordagens tradicionais é adotar *pixels* de valores nulos correspondentes a posições inexistentes na imagem. No pseudocódigo apresentado não é considerado o tratamento do valor dos *pixels* fora da região da imagem. Nas linhas 12 a 16 é calculado e acumulado o valor do novo *pixel* e, na linha 17, tal valor é armazenado na imagem de destino.

B. Abordagem Paralela

Na abordagem paralela apresentada neste trabalho, cada *pixel* da imagem de destino é calculado em paralelo. Entretanto, o cálculo do *pixel* gerado a partir da convolução da máscara com a janela da imagem original é realizado em uma estrutura de repetição não paralelizada. Assim, ocorrem diversas convoluções em paralelo, porém o cálculo da convolução não é paralelo. O PSEUDOCÓDIGO II apresenta a parte principal da abordagem paralela da Pirâmide Gaussiana.

IV. IMPLEMENTAÇÕES EXPERIMENTAIS

A partir dos pseudocódigos apresentados na Seção III, as implementações a seguir foram desenvolvidas para realizar os experimentos. Para a versão sequencial, foi considerada uma máquina com processador Core i7-5500U de 2.4 GHz, 8 GB de memória RAM. Já para a versão paralela, foram feitos testes em 3 GPU's distintas, com as seguintes configurações:

- GPU NVIDIA GeForce GTX 830M com 2 multiprocessadores com 128 núcleos cada, 2 GB de memória RAM, computabilidade 5.0 e capacidade máxima de 1024 *threads* por bloco;
- GPU NVIDIA Tesla C2075 com 14 multiprocessadores com 32 núcleos cada, 6 GB de memória RAM, computabilidade 2.0 e capacidade máxima de 1024 *threads* por bloco;
- GPU NVIDIA GeForce Titan X com 24 multiprocessadores com 128 núcleos cada, 12 GB de memória RAM, computabilidade 5.2 e capacidade máxima de 1024 *threads* por bloco.

PSEUDOCÓDIGO II. LISTAGEM DO PSEUDOCÓDIGO DA ABORDAGEM PARALELA

```
   function ReduceP(im_l, gm): im_{l+1}
   {Reduz, em paralelo, a dimensão da imagem im_l para o próximo nível da pirâmide}
   Entrada  im_l[1..linhas, 1..colunas, 1..canais] de Byte
            gm[−2..2, −2..2] de Real
   Saída    im_{l+1}[1..linhas/2, 1..colunas/2, 1..canais] de Byte
1  begin
2    for all pixels in im_l(i,j,k),
3        1 ≤ i ≤ linhas/2, 1 ≤ j ≤ colunas/2,
4        1 ≤ k ≤ canais pardo
5    begin
6      newPixel ← 0;
7      for m ← −2 to 2 do
8      begin
9        for n ← −2 to 2 do
10       begin
11         newPixel ← newPixel
12                  + im_l[2*i+m, 2*j+n, k]
13                  * gm(m,n);
14       end
15       im_{l+1}[i,j,k] ← newPixel;
16     end
17   end
18 end
```

Para codificação e compilação foram utilizados Linux Ubuntu 14.04, compilador GNU C++ 4.8.4, OpenCV 2.4 e CUDA SDK 7.5. A fim de considerar os custos de todas as transferências de dados nos experimentos, foram reportados os tempos decorridos em uma máquina dedicada desconsiderando fatores externos, como alta carga causada por outros processos.

A. Implementação Sequencial

O CÓDIGO I mostra a parte principal da implementação da abordagem padrão, ou seja, sequencial da Pirâmide Gaussiana para uma imagem bidimensional.

A variável gm é do tipo *Matrix<float>*, desenvolvido pelos autores, e otimizado para o uso tanto em funções na CPU quanto na GPU. Além disso, *Matrix* é um tipo criado para armazenar matrizes multidimensionais em forma de um vetor unidimensional em que os elementos das linhas de uma matriz são armazenados sequencialmente. Assim, ele encapsula um vetor linear da imagem RGB no mesmo formato do OpenCV, a estrutura da imagem (linhas, colunas e canais) e funções do CUDA para realizar alocação e liberação de memória, bem como cópias de dados entre memória do sistema e da GPU.

Na função *imReduce*, os parâmetros *im1* e *im2* são a imagem de origem e de destino, respectivamente, e *gm* é a máscara Gaussiana. Todavia, o valor de cada *pixel* da máscara já se encontra multiplicado por $1/256$. Observa-se que as variáveis $im1$ e $im2$ são do tipo *Mat*, que é um tipo para armazenamento de imagens da biblioteca do *OpenCV*.

CÓDIGO I. LISTAGEM DA FUNÇÃO SEQUENCIAL

```
1  void imReduce(Mat im1, Mat im2, Matrix<float> gm) {
2
3    int l2 = im2.rows;
4    int c2 = im2.cols;
5    int z  = im2.layers;
6
7    for(int k = 0; k < z; k++){
8      for(int i = 0; i < l2; i++){
9        for(int j = 0; j < c2; j++){
10         float nPix = 0;
11         for (int m = -2;m <= 2; m++){
12           for (int n = -2;n <= 2; n++){
13             float oPix = pix(im1, 2*i+m, 2*j+n, k);
14             nPix = nPix + oPix * gm(m+2, n+2);
15           }
16         }
17         im2.at<cv::Vec3b>(i,j)[k] = (unsigned char)nPix;
18       }
19     }
20   }
21 }
```

As linhas 7 a 9 referem-se às estruturas de repetição para percorrer cada canal, linha e coluna da imagem, respectivamente e as linhas de 11 a 16 referem-se ao cálculo da convolução sobre um único *pixel* da imagem de destino.

Nas linhas 13 e 14, o valor do novo *pixel* é calculado e na linha 17 é armazenado na imagem de destino.

B. Implementação paralela

Na implementação paralela, os dados das imagens são passados a partir do tipo *Matrix* e cada *thread* é responsável pelo cálculo paralelo de um *pixel* na imagem de destino. Porém, conforme já mencionado anteriormente, o cálculo do *pixel* gerado a partir da convolução da máscara com a janela da imagem original é realizado em uma estrutura de repetição não paralelizada. Ressalta-se que no tipo *Matrix*, para uso na GPU, os dados são alocados dinâmicamente com a função *cudaMalocPitch*, *copiados* com a função cudaMemcpy2d e acessados usando parâmetro de *Pitch* retornado pela função alocação. O uso destas funções proporciona um melhor alinhamento do acesso à memória e viabiliza a realização de *cache* de dados pela GPU que reduzem significativamente o tempo de acesso à memória global.

Considerando que cada GPU suporta uma quantidade máxima T de *threads* por bloco, e o algoritmo desenvolvido foi planejado para trabalhar com imagens que possuem 3 canais de cores (R,G,B), a dimensão do bloco foi calculada conforme a Equação 2, e o *grid* foi definido em duas dimensões, conforme Equações 3 e 4,

$$nt = \left\lfloor \sqrt{\frac{T}{nc}} \right\rfloor, \tag{2}$$

$$bx = \left\lceil \frac{C}{nt} \right\rceil, \tag{3}$$

$$by = \left\lceil \frac{L}{nt} \right\rceil, \tag{4}$$

onde nt é o número de *thread*, T (1.024) é o número máximo de *threads* por bloco, nc (3) é o número de canais da imagem, de modo que $nt \times nt \times 3 \leq T$, bx e by são os números de blocos nos eixos x e y, respetivamente e C e L são o número de colunas e linhas da imagem original, respetivamente.

Desta forma, a função *kernel* é invocada com $[bx \times by]$ blocos com $[nt \times nt \times 3]$ threads cada.

O CÓDIGO II apresenta a listagem da implementação paralela da Pirâmide Gaussiana.

CÓDIGO II. LISTAGEM DA FUNÇÃO PARALELA

```
1  __global__ void imReduceP(Matrix<unsigned char> im1,
2  Matrix<unsigned char> im2, Matrix<float> gm) {
3
4    int i = blockIdx.x * blockDim.x + threadIdx.x;
5    int j = blockIdx.y * blockDim.y + threadIdx.y;
6    int k = blockIdx.z * blockDim.z + threadIdx.z;
7
8    int l2 = im2.rows;
9    int c2 = im2.cols;
10   int z  = im2.layers;
11
12   if ((i >= 0) && (i < l2) &&
13       (j >= 0) && (j < c2) &&
14       (k >= 0) && (k <  z)){
15     float nPix = 0;
16     for (int m = -2; m <= 2; m++){
17       for (int n = -2; n <= 2; n++) {
18         float oPix = pix(im1, 2*i+m, 2*j+n, k);
19         nPix = nPix + oPix * gm(m+2, n+2);
20       }
21       im2(i,j,k) = (unsigned char)nPix;
22     }
23   }
24 }
```

Comparando-se a implementação sequencial (CÓDIGO I) com a implementação paralela (CÓDIGO II) é possível perceber que a implementação paralela não apresenta as estruturas de repetição para percorrer os canais de cores, linhas e colunas da imagem. Ao invés disso, as linhas 4 a 6 do código calculam as coordenadas do *pixel* a ser convoluído de acordo com as informações de bloco e *thread*. Como dito no início desta seção, as imagens de origem e destino são passadas para a GPU no formato $Matrix$.

C. Medidas de Desempenho

Os experimentos medem os tempos médios e totais de processamento obtidos com as Equações (5) e (7), o desvio padrão e o ganho de desempenho (*speedup*) obtidos com as Equações (6) e (8)[1], [4].

$$\overline{T}(P_T)_i = \overline{T}(CG)_i + \overline{T}(P)_i + \overline{T}(GC)_i, \tag{5}$$

$$Speedup_i = \frac{\overline{T}(1)_i}{\overline{T}(P_T)_i}, \tag{6}$$

$$T(P)_{Total} = \overline{T}(CG)_1 + \sum_{i=1}^{7} \overline{T}(P)_i + \overline{T}(GC)_7, \tag{7}$$

$$SpeedupT = \frac{\sum_{i=1}^{7} \overline{T}(1)_i}{\overline{T}(P)_{Total}}, \tag{8}$$

onde, $\overline{T}(P_T)_i$ é o tempo médio total para redução em paralelo para o nível i, $\overline{T}(CG)_i$ e $\overline{T}(GC)_i$ são os tempos médios de cópia da CPU-GPU e GPU-CPU, respectivamente, $\overline{T}(P)_i$ é o tempo de processamento paralelo do nível i. $Speedup_i$ é o ganho de desempenho de processamento e $\overline{T}(1)_i$

TABELA I. RESULTADOS OBTIDOS NOS EXPERIMENTOS

Dimensões da Imagem	Número de *Pixels*	Dimensões do *Grid*		Tempo de execução (ms) sequencial		Tempo de execução paralelo (ms) GTX 830M		C2075		Titan X	
		bx	by	μ	σ	μ	σ	μ	σ	μ	σ
$16.384 \times 8.192 \times 3$	402.653.184	512	256	56.879, 94	163, 14	500, 01	5, 43	401, 82	6, 04	370, 79	6, 01
$8.192 \times 4.096 \times 3$	100.663.296	256	128	14.727, 26	459, 05	125, 47	2, 87	99, 95	2, 35	92, 25	2, 37
$4.096 \times 2.048 \times 3$	25.165.824	128	64	3.590, 10	14, 79	32, 58	1, 17	25, 66	0, 48	23, 60	0, 48
$2.048 \times 1.024 \times 3$	6.291.456	64	32	901, 19	6, 41	8, 94	0, 51	7, 51	0, 67	6, 81	0, 48
$1.024 \times 512 \times 3$	1.572.864	32	16	226, 49	3, 04	2, 70	0, 16	2, 31	0, 16	2, 18	0, 16
$512 \times 256 \times 3$	393.216	16	8	57, 43	2, 76	0, 68	0, 26	0, 60	0, 26	0, 57	0, 26
$256 \times 128 \times 3$	98.304	8	4	16, 03	0, 39	0, 24	0, 08	0, 23	0, 08	0, 23	0, 08

é o tempo médio de processamento sequencial. $T(P)_{Total}$ é o tempo médio total de processamento paralelo para os 7 níveis consecutivamente e $SpeedupT$ é o ganho de desempenho total para os 7 níveis consecutivamente.

A seção seguinte apresenta os resultados obtidos a partir dos experimentos realizados.

V. RESULTADOS

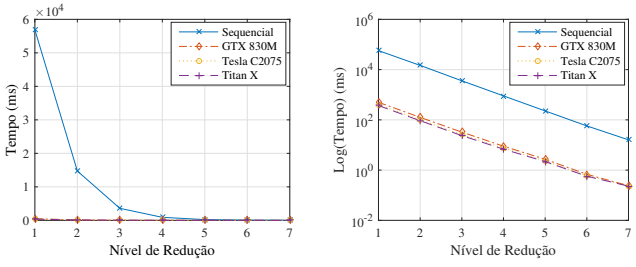
Os experimentos realizados consideram o uso de imagem com 16.384×8.192 *pixels* em resoluções diferentes e com três canais de cores (RGB). Após cada experimento é gerada uma imagem que possui metade da resolução anterior e que é utilizada como imagem inicial para demais experimentos. Como os tempos de processamento reduzem significativamente para imagens pequenas, os experimentos foram realizados apenas em 7 níveis de resolução. Ressalta-se que cada experimento é realizado por 10 repetições e seus valores são usados para a obtenção do valor médio de tempo de processamento (μ) e desvio padrão (σ).

A Tabela I apresenta a dimensão e o número de *pixels* das imagens, o número de blocos de *threads* utilizados (considerando que as GPU's utilizadas suportam apenas 1.024 *threads* por bloco), os tempos médios de execução (Equação (5)) e do desvio padrão apresentados nas versões sequencial e paralelas. Ressalta-se que os tempos médios de execução paralela consideram também os tempos de cópia da imagem entre CPU-GPU e GPU-CPU.

As Figuras 4(a) e 4(b) mostra um comparativo dos tempos médios de execução alcançados com as implementações sequencial e paralelas. No eixo x das figuras é representado o nível de redução que foi realizado sobre a imagem e no eixo y é apresentado o tempo em escala linear e logarítmica para os gráficos 4(a) e 4(b), respectivamente.

É possível perceber tanto na Tabela I quanto na Figura 4 que, nos primeiros níveis de redução, onde o volume de *pixels* é mais expressivo, o tempo médio de processamento da abordagem sequencial mostra-se significativamente superior aos tempos médios da abordagem paralela. Como os tempos das GPU's são significativamente mais baixos que o tempo sequencial, a Figura 4(b) permite visualizar apenas a ordem de grandeza linear do tempo sequencial em relação ao tempo nas GPU's. Assim, a Figura 4(b), que apresenta o tempo em escala logarítmica, permite visualização da grandeza logarítmica dos tempos de cada abordagem.

A Tabela II mostra os *speedup's* obtidos (Equação (6)), considerando o tempo médio de processamento sequencial em relação aos tempos médios de cada solução paralela. Como pode ser percebido os *speedup's* para as 3 GPU's apresentam



(a) Escala linear (b) Escala logarítmica

Figura 4. Comparação do tempo médio de execução entre as implementações sequencial e paralelas.

ganhos significativos em todos os experimentos e demonstram forte correlação ao volume de *pixels*.

TABELA II. GANHO DE DESEMPENHO PARA REDUÇÃO DE UM NÍVEL

Dimensões da Imagem	Número de *Pixels*	Speedup $GTX830M$	$C2075$	$TitanX$
$16.384 \times 8.192 \times 3$	402.653.184	113, 76	141, 55	153, 40
$8.192 \times 4.096 \times 3$	100.663.296	117, 38	147, 34	159, 65
$4.096 \times 2.048 \times 3$	25.165.824	110, 18	139, 90	152, 13
$2.048 \times 1.024 \times 3$	6.291.456	100, 75	119, 93	132, 27
$1.024 \times 512 \times 3$	1.572.864	83, 79	98, 23	104, 00
$512 \times 256 \times 3$	393.216	85, 01	96, 36	100, 88
$256 \times 128 \times 3$	98.304	65, 77	69, 15	70, 51

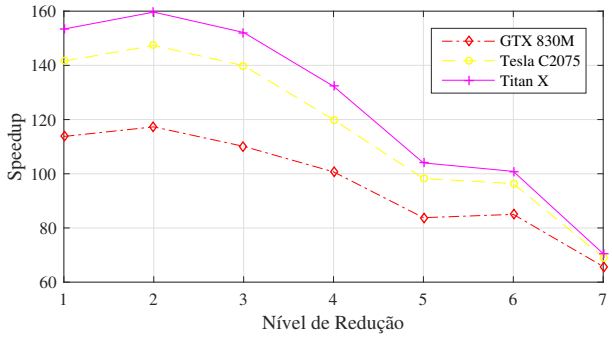


Figura 5. Comparação dos *speedup's* entre as GPU's.

Na Figura 5 é apresentado o comparativo dos *speedup's* alcançados com as implementações paralelas. No eixo x é representado o nível de redução realizado sobre a imagem a cada experimento. Assim, é possível perceber que com a redução do volume de dados, ocorre uma queda no Speedup de cada experimento. Ressalta-se que o Speedup representado possui ocilasções discretas que estão associadas ao tempo da abordagem sequencial que apresentou algumas reduções mais significativas de acordo com a geometria da imagem. Provavelmente este fenômeno está associado a algum alinhamento

de memória que proporcionou acesso mais rápido aos dados durante o processamento.

A Tabela III mostra os tempos gastos na cópia de dados da CPU para a GPU, bem como da GPU para a CPU nas versões paralelas implementadas.

TABELA III. TEMPO MÉDIO (μ) DE CÓPIA DE DADOS

Dimensões da Imagem	Número de *Pixels*	Tempo Médio (ms) CPU-GPU	GPU-CPU
$16.384 \times 8.192 \times 3$	402.653.184	$272,19$	$67,09$
$8.192 \times 4.096 \times 3$	100.663.296	$68,01$	$16,46$
$4.096 \times 2.048 \times 3$	25.165.824	$17,08$	$4,62$
$2.048 \times 1.024 \times 3$	6.291.456	$4,77$	$1,54$
$1.024 \times 512 \times 3$	1.572.864	$1,54$	$0,49$
$512 \times 256 \times 3$	393.216	$0,37$	$0,14$
$256 \times 128 \times 3$	98.304	$0,11$	$0,08$

O tempo de cópia CPU-GPU representa a cópia da imagem na sua dimensão original e o tempo GPU-CPU representa a cópia da imagem na sua dimensão reduzida. Ressalta-se que, a cada transformação, a imagem tem sua dimensão reduzida para $1/4$ do número original de *pixels*. Como pode ser visto na Tabela III, os tempos médios de cópia CPU-GPU e GPU-CPU apresentam uma relação proporcional de aproximadamente $1/4$ entre si. Os tempos de cópia apresentados são expressivos no caso da imagem com dimensões maiores, enquanto se mostram quase insignificantes nos experimentos com imagens de dimensões menores.

TABELA IV. TEMPO TOTAL DE PROCESSAMENTO E GANHO DE DESEMPENHO PARA REDUÇÃO DE 7 NÍVEIS CONSECUTIVOS

Tipo	CPU/GPU	Tempo (ms)	Speedup
Sequencial	Core i7	$76.398,44$	
	GTX 830M	$488,42$	$156,42$
Paralelo	C2075	$355,88$	$214,67$
	Titan X	$314,22$	$243,14$

A Tabela IV apresenta o tempo total para redução da imagem em 7 níveis (Equação (7)) e ganho de desempenho (Equação (8)). Observa-se que os tempos das execuções paralelas são menores que a execução do primeiro nível de redução, apresentado na Tabela I. Isto deve-se ao fato de que, embora a imagem original precise ser copiada para GPU e este tempo seja grande, ao final da redução dos sete níveis, apenas a imagem final é copiada de volta para a CPU e esse tempo de cópia é insignificante (0,08 *ms*). Assim, o processamento de vários níveis em sequência proporciona uma expressiva economia de tempo de processamento, pois evitam-se sucessivas cópias entre CPU e GPU.

VI. CONCLUSÕES

Este trabalho apresenta uma abordagem de programação paralela com a Pirâmide Gaussiana para a redução de dimensão de imagens e realiza uma comparação com a mesma abordagem sequencial. A proposta tem como finalidade apresentar uma alternativa para manipulação de imagens de altíssima resolução conhecidas como Gigapixel em um tempo acessível. A abordagem foi desenvolvida utilizando a tecnologia NVIDIA Cuda e testada em 3 GPU's, sendo uma para uso em computadores de mesa básicos e outras duas para computação científica de alto desempenho.

Os resultados da comparação mostram que o tempo para redução de resolução em GPU's com alto poder de processamento é expressivamente reduzido frente às CPU's modernas. Para uma imagem RGB de aproximadamente $0,125$ Gigapixels (16.384×8.192), o tempo de processamento sofre uma redução de até 153 vezes, já incluídos seu tempo de cópia para a memória da GPU e retorno para a memória da CPU. A mesma imagem, para reduzir 7 níveis de resolução consecutivamente, apresenta uma queda no seu tempo total de processamento pois, o tempo de retorno da imagem reduzida torna-se insignificante, e consequentemente aumenta o ganho de desempenho para até 243 vezes.

Desta forma, diante do reduzido tempo de processamento, a abordagem apresentada demonstra um desempenho elevado e caracteriza-se como uma possibilidade de uso em estágios preliminares de diversos algoritmos clássicos, bem como os mais modernos para filtragem, segmentação, dentre outros em imagens Gigapixel.

VII. AGRADECIMENTOS

Este trabalho foi realizado com apoio financeiro dos editais 5, 6 e 8/2012 da Fundação de Apoio à Pesquisa do Estado de Goiás (FAPEG).

REFERÊNCIAS

[1] G. M. Amdahl, "Validity of the single processor approach to achieving large scale computing capabilities," in *Proceedings of the April 18-20, 1967, spring joint computer conference*. ACM, 1967, pp. 483–485.

[2] P. J. Burt and E. H. Adelson, "The Laplacian Pyramid as a Compact Image Code," *Transactions on Communications, IEEE*, vol. 9, no. 4, pp. 532–540, april 1983.

[3] E. Elboher and M. Werman, "Efficient and accurate Gaussian image filtering using running sums," in *2012 12th International Conference on Intelligent Systems Design and Applications (ISDA)*. IEEE, 2012, pp. 897–902.

[4] J. L. Gustafson, "Reevaluating Amdahl's law," *Communications of the ACM*, vol. 31, no. 5, pp. 532–533, 1988.

[5] M. Kraus, "Quasi-Convolution Pyramidal Blurring," *Journal of Virtual Reality and Broadcasting*, vol. 6, no. 6, 2009.

[6] Z. Li, J. Yang, J. Zhao, P. Han, and Z. Chai, "PIMR: Parallel and Integrated Matching for Raw Data," *Sensors*, vol. 16, no. 1, p. 54, 2016.

[7] E. Olmedo, J. d. l. Calleja, A. Benitez, and M. A. Medina, "Point to point processing of digital images using parallel computing," *International Journal of Computer Science, IJCSI*, vol. 9, no. 3, pp. 1–10, may 2012.

[8] H.-M. Yip, I. Ahmad, and T.-C. Pong, "An efficient parallel algorithm for computing the gaussian convolution of multi-dimensional image data," *The Journal of Supercomputing*, vol. 14, no. 3, pp. 233–255, 1999.