Aplicabilidade de GPUs de baixo custo na Otimização da Análise de Similaridade de Imagens

Felipe Prochazka
LaTIM - Laboratório de Telemedicina e Informática Médica
Universidade Federal de Alagoas - Campus Arapiraca
Arapiraca-AL, Brasil
Email: pro.ufal@gmail.com

Prof. Dr. Marcelo Costa Oliveira
LaTIM - Laboratório de Telemedicina e Informática Médica
Universidade Federal de Alagoas - Instituto de Computação
Maceió-AL, Brasil
Email: oliveiramc@ic.ufal.br

***Resumo*—Este trabalho evidencia a tecnologia *Compute Unified Device Architecture* (CUDA) desenvolvida pela Nvidia Corporation para lidar paralelamente com uma grande quantidade de informação com o auxílio de Unidades de Processamento Gráfico (GPUs) de baixo custo, utilizando a extensão PyCuda através de cenários de processamento em que *grids* computacionais e *cloud computing* eram geralmente as alternativas disponíveis para se lidar em tempo hábil com a Análise de Similaridade de Imagens. Visto que, esta é uma etapa fundamental da Recuperação de Imagens baseada em Conteúdo (CBIR) que considera as características ou padrões das imagens como parâmetros de busca. Em virtude disso, é obtida uma redução drástica no tempo de execução do algoritmo que analisa a base de dados contendo até 8192 vetores de características extraídos a partir do cálculo de 32 atributos de textura de cada imagem e provê suporte a identificação de imagens relevantes ao usuário.**

***Keywords*-CBIR; GPU; CUDA;**

***Abstract*—This work shows the technology Compute Unified Device Architecture (CUDA) developed by Nvidia Corporation to deal parallel with a large amount of information with the aid of low-cost Graphics Processing Units (GPUs), using the PyCuda extension through processing scenarios in which grid computing and cloud computing were generally available alternatives for dealing timely with the Similarities Analysis of Images. This is a fundamental stage on the Content-Based Image Retrieval (CBIR) that considers the characteristics or patterns of images as search parameters. As a result, obtaining at a drastic reduction in the runtime of the algorithm, that analyzes the database of up to 8192 feature vectors extracted from the calculation of 32 texture attributes from each image and provides support for the identification of relevant images to the user.**

***Keywords*-CBIR; GPU; CUDA;**

I. Introdução

A Recuperação de Imagem baseada em Conteúdo (CBIR - *Content-Based Image Retrieval*) é apresentada como um conjunto de técnicas de visão computacional que utiliza muitos recursos computacionais tanto de processamento quanto de armazenagem com o enfoque na busca de imagens digitais, evidenciando diversos estudos no campo de indexação, reconhecimento e análise de imagens ao longo das últimas décadas ([1], [2], [3]).

Dessa forma, a CBIR permitiu o desenvolvimento de programas mais modernos para a busca e indexação da crescente quantidade de imagens. Assim, uma imagem pode ser comparada com bases de imagens, verificando o grau de similaridade entre estas imagens digitais. Consequentemente, a análise de similaridade se tornou uma etapa fundamental para se encontrar imagens de uma mesma categoria ou similares às requisitadas pelo usuário com uma maior chance de serem relevantes, permitindo mais agilidade, segurança e precisão, como é visto na Figura 1.

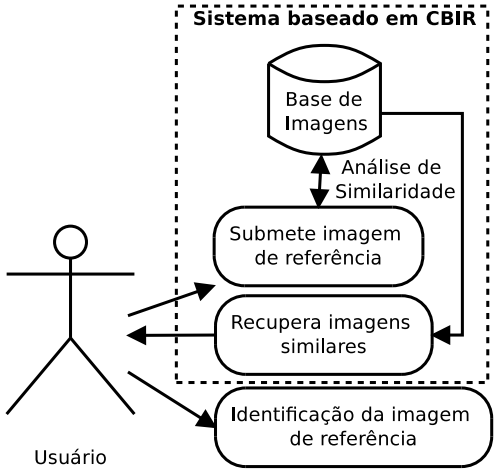

Figura 1. Exemplo básico de busca de imagem com CBIR

Um exemplo é o caso de sucesso do serviço *Google Images*. A partir do retorno dos resultados, os algoritmos do buscador tentam produzir uma contextualização da imagem fornecida, facilitando assim, a exibição das imagens por ordem de similaridade.

Contudo, com uma produção intensa de imagens em diversas bases de dados, se tornou cada vez mais necessária a busca por soluções de alto desempenho, visto que, quanto maior a base de imagens para análise por meio da CBIR, maior a necessidade destes sistemas por processamento e eficiência [4].

Com a pretenção de amortizar o tempo de processamento das técnicas de CBIR, algumas propostas baseadas em *grids* computacionais foram implementadas por alguns autores ([5], [6], [7]). Todavia, trabalhos baseados em Unidades de Processamento Gráfico (GPUs - *Grafical Processing Units*) são uma boa oportunidade de se explorar um novo paradigma do paralelismo desde o lançamento da arquitetura CUDA

(*Compute Unified Device Architecture*) pela Nvidia Corporation em 2006 ([4], [8], [9]).

Alguns pesquisadores já utilizaram as vantagens oferecidas pelas GPUs, conseguindo incrementos de até 300 vezes no desempenho de suas aplicações [10]. Dessa forma, alguns projetos comerciais de CBIR com a aplicabilidade do CUDA visaram melhorar o desempenho através da paralelização de algoritmos de grande custo computacional, como o *Incogna Image* que é baseado na potência de GPUs Nvidia Tesla.

Assim, a programabilidade de GPUs é bastante promissora nesta área devido ao baixo custo de aquisição deste tipo de *hardware* e o alto poder de processamento paralelo, tornando possível fornecer a capacidade computacional esperada para se alcançar resultados promissores com sistemas baseados em CBIR, sendo assim, uma alternativa aos tradicionais *grids* computacionais ou ainda ao *cloud computing*.

Este trabalho apresenta a aplicabilidade das GPUs com o uso da arquitetura CUDA na otimização dos algoritmos de CBIR, em especial, na etapa de Análise de Similaridade de Imagens, propondo o Módulo Acelerador de Análise de Similaridade de Imagens via GPUs (MAASI-GPU), visando acelerar este processo por meio dos conceitos de paralelismo.

II. Trabalhos Correlatos

A. Sistemas comerciais baseados em CBIR

Na tabela I, é possível visualizar um levantamento sobre a relação entre cada sistema e a tecnologia de processamento adotada, além de exibir o número de imagens estimada na base de dados.

Tabela I
Sistemas comerciais baseados em CBIR

Nome	Processamento	Número de Imagens
Google Image	Não informa	Não informa
TinEye	Não informa	2 bilhões
Incogna Image	GPU com CUDA	100 milhões
vSearch	Não informa	10 milhões
IMMENSELAB	Não informa	10 milhões

B. Projetos de pesquisa baseados em CBIR

Alguns projetos foram desenvolvidos para explorar técnicas de CBIR, como o *Program for Indexing and Research Images by Affinity (PIRIA)* [11] e o *Anaktisi* [12] que combinam técnicas baseadas em cor e textura para recuperar imagens. Já o *GNU Image Finding Tool (GIFT)* [13] e o *BRISC* [14] são sistemas de código-aberto disponível. O *GIFT* é um dos mais notáveis programas de CBIR sob licença GNU, desenvolvido pela Universidade de Geneva. Enquanto o *BRISC* é voltado para imagens médicas de nódulos pulmonares a partir da base de imagens LIDC (*Lung Image Database Consortium*).

III. Extração de Atributos de Textura com a CBIR

A CBIR é fundamentada na extração de atributos de uma imagem com base em suas características por meio da análise dos *pixeis* que a formam. Desse modo, se tornou possível melhor identificar uma imagem a partir da sua composição, como, por exemplo, pelos atributos de cor, forma ou textura, demonstrando vantagens à identificação de imagens baseadas puramente em descritores textuais que formam a classificação tradicional de arquivos de imagens, passível de altas taxas de erros em suas rótulações textuais [15].

Sabe-se que, as texturas são padrões visuais complexos compostos por propriedades como brilho, densidade, granulação, rugosidade, dentre outros. Estas propriedades podem estar relacionadas com a cor da superfície ou com a aparência física do objeto. A representação da textura possui alguns métodos que merecem destaque: métodos estruturais (a identificação da textura é baseada na detecção de primitivas matemáticas), espectrais (o espectro de Fourier é utilizado para verificar a frequência da imagem) ou ainda estatísticos.

A. Métodos estatísticos

Nos métodos estatísticos, é realizada uma análise da imagem a partir da intensidade de cada *pixel*, visando assim, obter os descritores de suas características. Dessa forma, são utilizados métodos matemáticos para averiguar a distribuição dos *pixeis* no interior da imagem em relação a uma amostra de referência. A maioria destes métodos são utilizados em escala de cinza e podem tirar vantagem da estatística da primeira categoria (média, moda e a mediana) ou segunda categoria (variação, desvio-padrão ou ainda as matrizes de co-ocorrência). Assim, a estatística da primeira categoria ou ordem é baseada nos dados obtidos a partir das intensidades dos *pixeis* no histograma. Contudo, assim como, toda análise de histograma em escala de níveis de cinza, é passível de limitações devido a ausência de informações sobre a distribuição espacial dos mesmos.

B. Matriz de Co-Ocorrência (MCO)

Com a finalidade de contornar este problema nos métodos estatísticos, a estatística de segunda ordem utiliza a matriz de co-ocorrência, que evidencia informações sobre o posicionamento dos *pixeis*. Assim, este tipo de estrutura matemática representa a distribuição de probabilidade de ocorrência de pares de *pixeis* com determinado tipo de intensidade com níveis de cinza(*i* e *j*) e distância em uma direção (*d*) sob uma orientação (θ) entre os mesmos nas dimensões *x* e *y*.

É recomendável pela literatura [16] que, devido ao cálculo da matriz de co-ocorrência consumir bastante processamento, que o valor de *d* deve ser entre 1 e 2 *pixeis* e os valores utilizados de θ sejam $0\,^\circ$, $45\,^\circ$, $90\,^\circ$, $135\,^\circ$. Embora, se nota avanços em métodos de processamento paralelo como alternativa para lidar também com a demanda computacional deste tipo de problema [17].

C. Atributos de Textura

Há aproximidamente 20 funções estatísticas clássicas de segunda ordem utilizadas na extração de atributos de textura por meio de uma matriz de co-ocorrência para se poder realizar a análise de similaridade [16]. Contudo, algumas destas, já definem de maneira satisfatória uma imagem por esta técnica [6], onde *C(i,j)* são os elementos da matriz de co-ocorrência, μ_x e μ_y são as médias e σ_x e σ_y são os desvios-padrões.

1) **Energia**: homogeneidade dos *pixeis*, maiores valores significam intensidades mais uniformes na imagem.

$$\sum_{i,j} C(i,j)^2$$

2) **Entropia**: dispersão de ocorrências de níveis de cinza na imagem.

$$-\sum_{i,j} C(i,j) \log C(i,j)$$

3) **Momento da Diferença Inverso**: concentração das ocorrências de níveis de cinza na diagonal da matriz de co-ocorrência da imagem.

$$\sum_{i,j} \frac{1}{1+(i-j)^2} C(i,j)$$

4) **Matiz**: grau de simetria, valores altos quando a imagem não é simétrica.

$$\sum_{i,j} (i-j-\mu_x-\mu_y)^3 C(i,j)$$

5) **Contraste**: variação de intensidade dos *pixeis*, valores altos quando a diferença é grande entre os níveis de cinza da imagem.

$$\sum_{i,j} (i-j)^2 C(i,j)$$

6) **Proeminência**: grau de simetria, valores altos quando a imagem não é simétrica.

$$\sum_{i,j} (i-j-\mu_x-\mu_y)^4 C(i,j)$$

7) **Correlação**: grau de dependência linear da tonalidade de níveis de cinza da imagem.

$$-\sum_{i,j} \frac{(i-\mu_x)(j-\mu_y)}{\sqrt{\sigma_x \sigma_y}} C(i,j)$$

8) **Variância**: dispersão dos níveis de cinza em relação a intensidade média da imagem.

$$\sum_{i,j} (i-\mu)^2 C(i,j)$$

D. Criação dos vetores de características

Dessa forma, 8 atributos podem ser calculados para uma matriz de co-ocorrência em 4 orientações ângulares de θ (0 °, 45 °, 90 °, 135 °), extraindo assim, 32 atributos de textura para formar um vetor de características para identificar cada imagem.

IV. O CÁLCULO DE SIMILARIDADE

Entende-se que, a partir das técnicas de extração de atributos representativos de uma imagem ou de uma coleção delas, um sistema baseado em CBIR precisa realizar uma análise destes atributos para poder quantificar a similaridade entre uma imagem e as que se encontram numa base de imagens, quando for necessário realizar alguma comparação.

Neste processo de busca por similaridade, é preciso realizar, uma comparação atributo a atributo de uma imagem com cada imagem do banco de dados. Visando assim, verificar a distância matemática utilizando uma métrica definida pelo usuário. Nesta comparação entre um vetor de características (formada por uma cadeia de atributos) e outro, valores menores indicam graus maiores de similaridade e, se a distância for zero em todas estas comparações entre uma imagem e outra, a imagem é idêntica a que está na base de imagens.

Há um certo número de distâncias matemáticas que podem ser utilizadas num sistema com CBIR [6]. Todavia, as funções de distância utilizadas neste trabalho foram: a Distância Euclidiana e a Distância Manhattan.

A. Distância Euclidiana (DE)

A Distância Euclidiana (DE) é frequentemente um método utilizado como padrão em muitos sistemas baseados em CBIR. A vantagem é a extensa literatura sobre a sua aplicabilidade, o baixo custo computacional e a sua relativa facilidade de implementação, embora, como desvantagem, esta métrica demonstre baixa precisão.

Dessa maneira, a equação calculada na comparação entre cada atributo do vetor de características de cada par de vetores envolvidos numa análise de similariedade é:

$$\mathrm{DE}(\mathbf{x},\mathbf{y}) = \sqrt{(x_1-x_2)^2+(y_1-y_2)^2}$$

Onde x_1 e y_1 são as coordenadas de um atributo pertencem a um vetor de características que se queira comparar com outro que contém x_2 e y_2.

B. Distância Manhattan (DM)

A Distância Manhattan (DM), também conhecida como "*city block*", consiste na soma das diferenças absolutas entre os vetores de características. A vantagem principal desta métrica é a performance, devido a simplicidade do cálculo.

Logo, o cálculo desta métrica, por definição, é realizado pela equação:

$$\mathrm{DM}(\mathbf{x},\mathbf{y}) = \|x_1-x_2\| + \|y_1-y_2\|$$

Onde x_1 e y_1 são as coordenadas de um atributo pertencem a um vetor de características que se queira comparar com outro que contém x_2 e y_2.

V. A ARQUITETURA CUDA COM A EXTENSÃO PYCUDA

A Arquitetura do Dispositivo Unificado de Computação (*Compute Unified Device Architecture* - CUDA) [1] é caracterizada por ser voltada para facilitar a programabilidade de placas

[1] http://developer.nvidia.com/nvidia-gpu-computing-documentation

gráficas GPUs com a linguagem de programação C/C++, além da capacidade de interagir com a CPU, formando assim, uma infraestrutura para extração do processamento da computação paralela e heterogênea, ou seja, é um *framework* baseado no paradigma da Computação de Propósito Geral para Unidades de Processamento Gráfico (*General-Purpose computation on Graphics Processing Units* - GPGPU).

A. Modelo de Programação

É importante definir que, o processamento de dados realizado pela arquitetura CUDA é baseado em linhas de execução paralelas ou *threads*. As *threads* ou *warps* (conjuntos de *threads* iniciadas juntas) são agrupados em blocos tridimensionais (*blocks*), blocos podem ser organizados numa grade bidimensional (*grids*) e muitas grades podem se organizar num núcleo (*kernel*). *Threads* de um mesmo bloco possuem memória compartilhada (*shared memory*) e podem com mais facilidade sincronizar suas execuções entre si, contudo *threads* de blocos diferentes não se comunicam, garantindo que cada bloco possa ser executado independentemente [18]. Assim, cada núcleo (*kernel*) executa uma função programada em CUDA.

Cada função programada em CUDA, ao ser executada por um *kernel*, precisa especificar a quantidade de recursos que será exigido da GPU, ou seja, o número de *threads* em cada bloco (*block*), o número de blocos (*blocks*) em cada *grid* e a quantidade de memória compartilhada destinada para a computação.

Com o objetivo de melhor abrangência e desenvolvimento da tecnologia CUDA, os desenvolvedores de software possuem a disposição extensões ou *wrappers* mantidos pela comunidade do mesmo que poderão auxiliar nesta tarefa utilizando a sua linguagem de programação mais familiar: CUDA para C, CUDA.NET para C#, o jCuda para Java e o PyCuda para Python, dentre outros.

O PyCuda é uma extensão ou *wrapper* baseado na linguagem de programação Python para facilitar a programação CUDA [19]. Esta extensão foi desenvolvida por Andreas Klöckner da Universidade de Nova York (EUA) como uma proposta de abstrair do programador diversos detalhes de gerência de *threads* e de memória do CUDA, ou seja, nos detalhes de baixo nível da tecnologia inerentes a linguagem de programação C/C++.

Esta extensão visa produtividade da programação de alto nível do Python para a construção da aplicação e o encapsulamento do C/C++ para a escrita otimizada das funções (*kernels*) CUDA. Assim, enquanto o Python lida com as chamadas da CPU, permitindo o controle e a comunicação entre os recursos do sistema, o PyCuda, encapsulando o CUDA, lida com o paralelismo da GPU, permitindo assim, a definição de como os dados serão processados de maneira heterogênea [19]. Consequentemente, é evidente um modelo híbrido de programação que combina as virtudes de cada linguagem, tornando-se a escolha mais comum entre as extensões CUDA em desenvolvimento.

VI. Materiais e Métodos

Durante o desenvolvimento se verificou o tempo (em milíssegundos) de execução do algoritmo da Distância Euclidiana (DE) e da Distância Manhattan (DM) rodando na CPU e na GPU para cada plataforma de desenvolvimento.

A. Plataformas de Desenvolvimento

A máquina de teste 1 foi equipada com um Intel Core 2 Quad Q8200 2,33Ghz, uma GPU Nvidia Geforce 9400 GT 512Mb 128bits (2 SMs x 8 SPs = 16 *cuda cores* - Arquitetura G94 - 512 *threads* por *block*) e a máquina de teste 2 com um Intel Core i7 860 2.8Ghz, uma GPU Nvidia GTX 275 896Mb 448 bits (30 SMs x 8 SPs = 240 *cuda cores* - Arquitetura GT200b - 512 *threads* por *block*), ambas com 4Gb de RAM DDR3 e a mesma configuração de *software* para fins de comparação científica baseada em SO Linux Ubuntu 10.04 com o pacote CUDA 3.2 (SDK, *Toolkit* e o *Driver*), o Python 2.6, o NumPy 1.3 e a extensão (*wrapper*) PyCuda 0.94.

B. O Fator Speedup

Para mensurar o desempenho entre a solução sequencial e sua respectiva paralela, adotou-se a seguinte métrica [20]:

$$\mathrm{Speedup_p} = \frac{T_1}{T_p}$$

T_1 é o tempo de processamento do melhor algoritmo sequencial e T_p é o tempo de processamento de um algoritmo paralelo com p processadores.

C. Cenários de Processamento das Imagens

A métrica *speedup* foi mensurada usando a biblioteca *time* da linguagem de programação Python para cada cenário de processamento, baseada na quantidade de vetores de características das imagens (2, 16, 32, 64, 128, 256, 512, 1024, 2048, 4096, 8192) com 32 atributos de textura cada.

D. CUDA GPU Occupancy Calculator

O CUDA GPU *Occupancy Calculator* é um simulador dinâmico idealizado pela Nvidia com a capacidade de gerar gráficos comparativos para ajudar aos programadores da arquitetura CUDA no planejamento de funções otimizadas para ocupar ao máximo o uso de recursos da GPU nos mais diversos cenários de processamento.

VII. MAASI-GPU

Com a finalidade de aumentar o desempenho na busca de imagens similares, o Módulo Acelerador de Análise de Similaridade de Imagens via GPUs (MAASI-GPU) emprega os príncipios de Recuperação baseada em contéudo (CBIR) e a arquitetura CUDA com auxílio da extensão PyCuda na comparação paralela entre um vetor de característica de referência e uma base de dados (matriz de características) baseados em 32 atributos de textura, como pode ser visto na Figura 2.

Na Figura 3, se visualiza um fluxograma do MAASI-GPU e sua interação com sistemas baseados em CBIR, além de

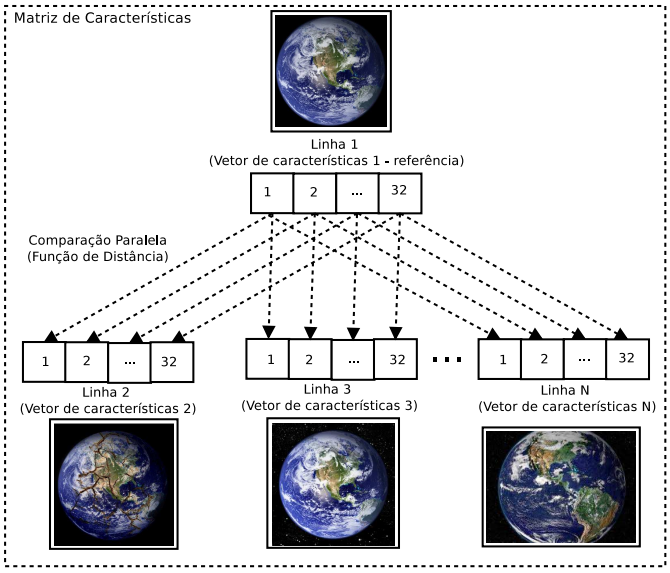

Figura 2. Comparação paralela entre vetores de características

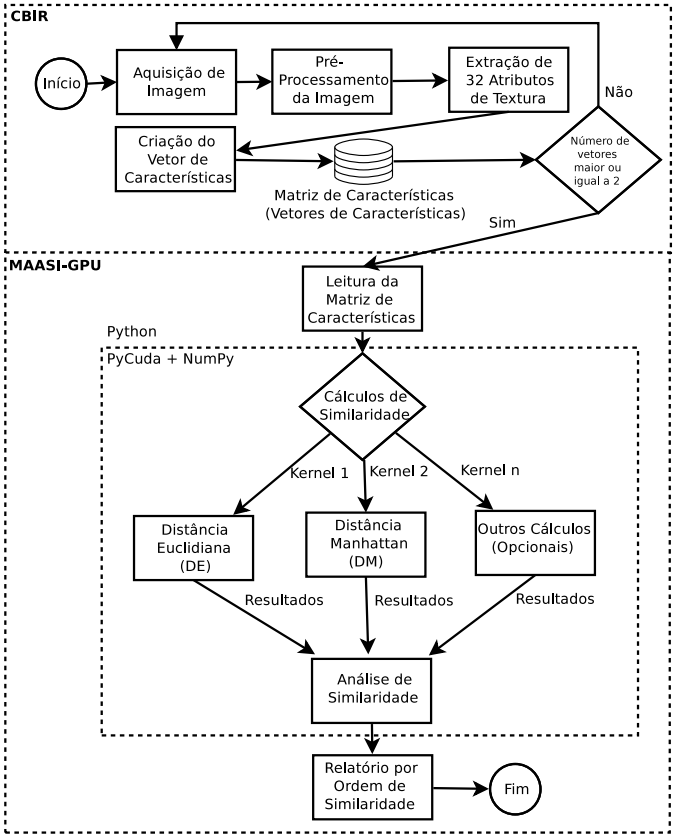

Figura 3. Fluxograma do MAASI-GPU

evidenciar os contextos de execução tanto da extensão PyCuda quanto da linguagem Python associados ao NumPy.

Desse modo, o MAASI-GPU foi fundamentado nas seguintes etapas:

1) A integração com sistemas baseados em CBIR é baseada em vetores de características de 32 atributos de textura (gerados conforme III-D) que alimentam a base de dados do módulo.
2) A matriz de características recebe o vetor de característica de referência e o posiciona na primeira linha em relação aos outros vetores de características já armazenados, facilitando a paralelização do algoritmo.
3) Após a leitura da matriz de características, os vetores de características são alocados para serem processados simultaneamente em cada *kernel* CUDA.
4) Na etapa de cálculo de similaridade descrita em IV-A e em IV-B, as funções de similaridade (DE e DM) são desenvolvidas utilizando a extensão PyCuda, resultando numa maior facilidade para a manutenção do módulo. Assim, os números de *blocks* e de *grids* são determinados dinamicamente de acordo com o número de vetores de características (conforme VI-C) que estão registrados na matriz de características (base de dados) e segue um tamanho restrito em 484 *threads* por *block* para obter otimização e não ultrapassar os limites da arquitetura CUDA nas máquinas de teste.
5) Relatórios por ordem de similaridade são gerados para auxiliar na análise e identificação das imagens. Por padrão do módulo, as primeiras 20 ocorrências de similaridade são exibidas para realizar o processamento num tempo adequado para o uso cotidiano dos usuários.

VIII. Resultados e Discussões

O desenvolvimento dos algoritmos paralelos e otimizados da Distância Euclidiana (DE) e da Distância Manhattan (DM) em PyCuda envolveu grande complexidade e requisitou mudanças na maneira como se desenvolve a programação.

Os resultados não sofreram influência em relação a falta de precisão do ponto flutuante em GPUs nas últimas casas decimais na Máquina de Teste 2 equipada com a Nvidia Geforce GTX 275, pois pode efetuar o Cálculo do Ponto Flutuante de Dupla Precisão. No caso das imagens de nichos específicos, como das imagens médicas, a precisão é um parâmetro essencial para a confiabilidade dos resultados.

Além disso, a taxa de ocupação do uso de *threads* por *block* está baseada numa divisão de recursos das GPUs proporcional a quantidade de vetores de características, sendo cerca de 94,53% nas máquinas de teste, o que conduz a um impacto no desempenho entre CPUs e GPus.

A. Resultados da Máquina de Teste 1

A Figura 4 demonstra uma comparação de desempenho obtido com os algoritmos de Distância Euclidiana (DE) e Distância Manhattan (DM) no MAASI-GPU e no MAASI-CPU na Máquina de Teste 1. O *speedup* foi 209x para DE e 236x para DM.

B. Resultados da Máquina de Teste 2

A Figura 5 evidencia uma comparação de desempenho obtido com os algoritmos de Distância Euclidiana (DE) e Distância Manhattan (DM) no MAASI-GPU e no MAASI-CPU na Máquina de Teste 2. O *speedup* foi 399x para DE e 409x para DM.

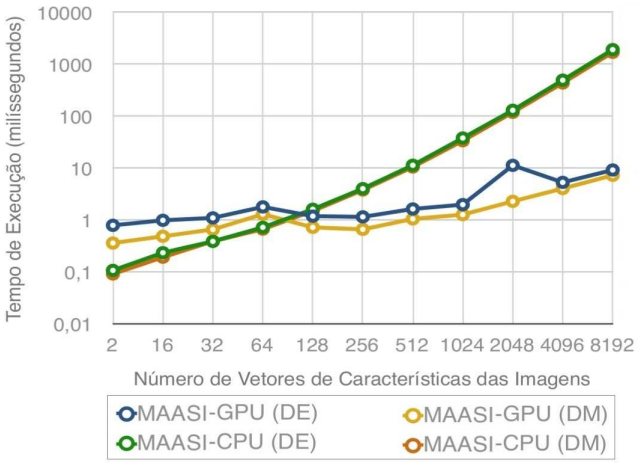

Figura 4. Desempenho da Máquina de Teste 1 equipada com uma Geforce 9400 GT

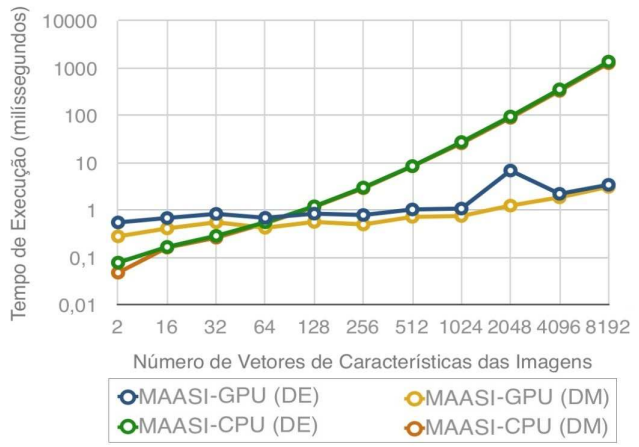

Figura 5. Desempenho da Máquina de Teste 2 equipada com uma Geforce GTX 275

IX. Conclusão

Os avanços na tecnologia de programabilidade de GPUs ressaltam a importância do estudo e do desenvolvimento de aplicações que analisem adequadamente a arquitetura do *hardware* gráfico e executem as suas rotinas com otimização e precisão na Análise de Similaridade de imagens.

A técnica de Recuperação de Imagens baseada em Conteúdo (CBIR) e em atributos de textura associada a computação paralela usando GPUs habilitadas com o CUDA (e da extensão PyCuda) são viáveis e ressaltam os benefícios do uso deste tipo de solução.

Desse modo, este trabalho foi desenvolvido a partir de um bom embasamento teórico com a finalidade de compreender o paradigma do paralelismo e extrair o máximo de desempenho da arquitetura CUDA através do entendimento da hierarquização de *threads* e da correta divisão dos recursos da GPU, mas também, reafirmando o uso do PyCuda como uma boa forma de se obter produtividade numa área em constante expansão para novas pesquisas e oportunidades de desenvolvimento de soluções de alto desempenho voltadas para a crescente demanda da sociedade.

Como trabalho futuro acreditamos fortemente que a ferramenta desenvolvida possa ser utilizada na otimização das técnicas de Recuperação de Imagens Médicas baseadas em Conteúdo de grandes centros de saúde, visto que, o volume de imagens médicas produzido diariamente é crescente e constitui em um dos maiores desafios da área de TI em saúde.

Referências

[1] M. Flickner, H. Sawhney, W. Niblack, J. Ashley, Q. Huang, B. Dom, M.Gorkani, J. Hafner, D. Lee, D. Petkovic, D. Steele, and P. Yanker, "Query by image and video content: The qbic system," 1995.

[2] P. M. A. Marques, "Diagnóstico auxiliado por computador na radiologia," 2001.

[3] H. Müller, N. Michoux, D. Bandon, and A. Geissbugler, "A review of content-based image retrieval systems in medical applications - clinical benefits and future directions," 2004.

[4] I. Scholl, T. Aach, T. M. Deserno, and T. Kuhlen, "Challenges of medical image processing," *Computer Science - Research and Development*, 2011.

[5] V. Breton, C. Blanchet, L. Maigne, and J. Montagnat, "Grid technology for biomedical applications," *Lecture Notes in Computed Science*, 2005.

[6] M. C. Oliveira, W. Cirne, and P. M. de Azevedo Marques, "Towards applying content-based image retrieval in the clinical routine," *Future Generation Computer Systems*, 2007.

[7] T.-W. Chiang, T. Tsai, and M.-J. Hsiao, "A hierarchical grid-based indexing method for content-based image retrieval," *IIH-MSP '07 Proceedings of the Third International Conference on International Information Hiding and Multimedia Signal Processing*, 2007.

[8] S. Srikanthan, A. Kumar, and V. Krishnan, "Accelarating the euclidean distance matrix computation using gpus," *3rd International Conference on Electronics Computer Technology (ICECT)*, 2011.

[9] K. Yadav, A. Srivastava, A. Mittal, and M. Ansari, "Parallel implementation of shape based image retrieval approach on cuda in compressed domain," *IJCA Special Issue on Novel Aspects of Digital Imaging Applications*, 2011.

[10] Q. Fang and D. A. Boas, "Monte carlo simulation of photon migration in 3d turbid media accelerated by graphics processing units," *Optics Express 17*, 2009.

[11] "Piria: a general tool for indexing, search, and retrieval of multimedia content."

[12] K. Zagoris, S. A. Chatzichristofis, N. Papamarkos, and Y. S. Boutalis, "img(anaktisi): A web content based image retrieval system," *SISAP '09 Proceedings of the 2009 Second International Workshop on Similarity Search and Applications*, 2012.

[13] M. Rummukainen, J. Laaksonen, and M. Koskela, "An efficiency comparison of two content-based image retrieval systems, gift and picsom," *CIVR'03 Proceedings of the 2nd international conference on Image and video retrieval*, 2003.

[14] M. O. Lam, T. Disney, D. S. Raicu, J. Furst, and D. S. Channin, "Brisc - an open source pulmonary nodule image retrieval framework," *J. Digital Imaging.*, 2007.

[15] M. C. Oliveira, P. M. A. Marques, and W. C. C. Filho, "Grades computacionais na otimização da recuperação de imagens médicas baseada em conteúdo," *Radiologia Brasileira*, 2007.

[16] R. M. Haralick, K. Shanmugam, and I. Dinstein, "Textural features for image classification," *IEEE Transactions on Systems on Systems, Man, and Cybernetics*, 1973.

[17] F. I. Alam and R. U. Faruqui, "Optimized calculations of haralick texture features," *European Journal of Scientific Research*, 2011.

[18] M. Garland, S. L. Grand, J. Nickolls, A. Anderson, J. Hardwick, S. Morton, E. Phillips, Y. Zhang, and V. Volkov, "Parallel computing experiences with cuda," *Micro IEEE*, 2008.

[19] A. Kloeckner, N. Pinto, Y. Lee, B. Catanzaro, P. Ivanov, and A. Fasih, "Pycuda: Gpu run-time code generation for high-performance computing." Brown University, Tech. Rep., 2009.

[20] P. Pacheco, *Parallel Programming With MPI*. Moragen Kaufmann, 1996, vol. 1.