Método de Suavização de Imagem baseado num Modelo Variacional Paralelizado em Arquitetura CUDA

Carlos A. S. J. Gulo[1], Henrique F. de Arruda[1], Antonio C. Sementille[1], Alex F. de Araujo[2], João Manuel R. S. Tavares[2]

[1] Departamento de Ciências da Computação,
Universidade Estadual Paulista - UNESP, Bauru, Brasil.

[2] Instituto de Engenharia Mecânica e Gestão Industrial, Faculdade de Engenharia,
Universidade do Porto, Porto, Portugal.

Resumo

O aumento constante da velocidade de cálculo dos processadores tem sido uma grande aliada no desenvolvimento de áreas da ciência que exigem processamento de alto desempenho. Associado ao aumento dos recursos computacionais, tem-se presenciado um aumento no emprego de técnicas de computação paralela, no intuito de explorar ao máximo a capacidade de processamento das arquiteturas multiprocessador. No entanto, o custo financeiro para aquisição de *hardware* para computação paralela não é baixo, implicando assim aa busca de alternativas. A arquitetura GPGPU (*General Purpose Computing on Graphics Processing Unit*), torna-se uma opção de baixo custo sem comprometer o poder de processamento necessário. Neste trabalho, esta arquitetura é empregada na paralelização de um método de suavização de imagem baseado num modelo variacional, aplicado em sequências de imagens de ultra-sonografia. Os resultados obtidos são promissores, permitindo um ganho de tempo computacional considerável.

Palavras-chave: *General Purpose Computing on Graphics Processing Units* (GPGPU), Processamento de Imagem, Suavização, Ruído Multiplicativo.

Contato dos Autores:
sander@unemat.br
h.f.arruda@gmail.com
semente@fc.unesp.br
fa.alex@gmail.com
tavares@fe.up.pt

1. Introdução

Diversas áreas de pesquisa como computação científica, computação gráfica, realidade virtual, astronomia, geoprocessamento, processamento e análise de imagem, previsão meteorológica, dentre muitas outras, necessitam de alto poder de processamento. Uma maneira de lidar com esta necessidade cada vez mais crescente pode ser por meio de técnicas de paralelização, as quais utilizadas para reescrever as soluções existentes decompondo os algoritmos ou os dados, em partes menores para serem executadas em vários processadores simultaneamente. Desta maneira, é possível minimizar a carga de operações por núcleo de processamento e maximizar o *speedup*, ou ganho computacional, da solução [Page 2009].

Neste trabalho, realizou-se um estudo das técnicas de paralelização comumente adotadas em GPGPU, apresentando uma implementação paralelizada do método de suavização de imagens afetadas por ruído multiplicativo, baseado num modelo variacional, proposto por Jin e Yang [2011]. Os testes experimentais desta paralelização foram realizados utilizando o modelo de paralelização *Single Instruction, Multi-Thread* (SIMT) e a arquitetura *Compute Unified Device Architecture* (CUDA) [Kirk e Hwu 2010, Borgo e Brodlie 2009]. Os resultados preliminares apontaram um ganho computacional de aproximadamente 3.5 vezes, em relação à implementação sequencial do mesmo método.

Este artigo está organizado da seguinte forma: na seção seguinte, são apresentados os conceitos importantes empregados na paralelização em GPGPU, bem como a descrição do método de suavização adotado. A descrição da paralelização desenvolvida é apresentada na seção 3, seguida pelos resultados experimentais e as considerações finais.

2 Fundamentação Teórica

2.1 Computação Paralela e GPGPU

O processo de paralelização de um algoritmo inicia-se com a decomposição de um problema maior em subproblemas, isto é, tarefas. Em seguida as tarefas são distribuídas entre os processadores disponíveis, tomando os cuidados necessários para garantir, sempre que possível, a independência de dados e controlar a sincronização e comunicação entre as Unidades de Processamento (UP), buscando assim uma maior eficiência na paralelização das tarefas [Parhami 2002].

De acordo com Vadja [2011], a decomposição de algoritmos pode ser classificada em funcional, baseada em dados e baseada em tarefas. Na decomposição funcional é possível uma UP continuar realizando suas operações sem a necessidade de aguardar o resultado de processamento em outra UP. Ao utilizar esta técnica a implementação pode ser feita determinando as partes

do código a serem paralelizadas. A decomposição funcional pode ser subdivida em estática e dinâmica. Na paralelização de código, quando é definido um número específico de UPs, tem-se a decomposição funcional estática, sendo sua principal desvantagem a falta de escalabilidade [Kirk e Hwu 2010]. Por outro lado, na decomposição funcional dinâmica, a aplicação é independente da quantidade de processadores, permitindo escalabilidade e *speedup* ao sistema computacional [Vadja 2011].

Para a decomposição baseada em dados, a paralelização inicia-se com a distribuição de processos, ou *threads*, entre as UPs disponíveis para o processamento de conjuntos de dados diferentes e não dependentes [Gurd 1988].

A implementação de técnicas de decomposição de tarefas, deve levar em consideração o funcionamento básico de processos e *threads*, em nível de sistema operacional. Conceitualmente, os processos podem conter *threads*, caracterizando-se assim a dependência hierárquica entre eles. Os *threads* compartilham recursos como espaço de endereçamento, variáveis globais, arquivos, dentre outros. Por outro lado, os processos são completamente isolados, sendo considerados pelo sistema operacional como unidades de isolamento permitindo melhor gerenciamento de falhas [Tanenbaum 2010].

Aplicações paralelas, em geral, exigem interações na troca de informações, acesso aos mesmos dados, espera no término de processamento, dentre outras. O uso destas interações é facilitado por meio de mecanismos de sincronização como *locks*, semáforos, regiões críticas [Gebali 2011].

Durante a execução de uma aplicação paralela, uma UP pode precisar de resultados de outra. Esta dependência de dados exige que a aplicação aguarde o processamento das UPs envolvidas, realizando o sincronismo para concluir a tarefa. Em outra situação, partes da aplicação paralela podem realizar acessos frequentes a recursos do sistema, como *hardware* e acesso de memória compartilhada. Nestes casos, a decomposição exige a necessidade de sincronizar as diferentes tarefas paralelizadas [Page 2009 e Gebali 2011].

A comunicação entre as UPs, no contexto das GPGPUs, é fortemente acoplada. Dessa maneira adotou-se o modelo SMP (*Stream Programming Model)*, o qual consiste em um conjunto de processadores idênticos que podem executar tarefas em cooperação para resolver problemas utilizando o compartilhamento de memória [Hennessy 2007]. As UPs envolvidas na execução das tarefas comunicam-se para trocar informações acerca do processamento, além de realizar operações, como a atualização dos dados armazenados em memória.

As técnicas de paralelização mais comuns ao tipo de arquitetura GPGPU [NVIDIA 2010, Kirk e Hwu 2010], são: paralelismo de tarefas, o qual realiza a execução simultânea de múltiplas tarefas, que podem ser *threads* ou processos, em dados diferentes; paralelismo de dados, utilizado na presente pesquisa, onde múltiplos dados recebem a execução de uma tarefa; e o paralelismo de instruções, quando há a execução de múltiplas instruções por processador.

A aplicação destas técnicas de paralelização exige o suporte específico do *hardware*. A CPU (*Central Processing Unit*) é otimizada para desempenho de código sequencial, utilizando sofisticadas lógicas de controle para permitir que instruções de um único *thread* executem em paralelo A velocidade de acesso à memória não aumenta muito em razão da utilização do processador para propósito geral, satisfazendo dispositivos de entrada e saída, sistemas operacionais e aplicativos [Kirk e Hwu 2010, Sanders 2011]. Na GPGPU é priorizada a realização de uma enorme quantidade de cálculos de pontos flutuantes, disponibilizando pequenas unidades de memória *cache* para facilitar o controle de requisitos de largura de banda. Assim, múltiplos *threads* podem acessar os mesmos dados na memória gráfica, dispensando a necessidade de recuperação constante de dados na memória principal, uma vez que a velocidade de acesso à memória gráfica é, em geral, cerca de 10 vezes mais rápida em processadores gráficos [Kirk e Hwu 2010, Sanders 2011].

A partir destas considerações acerca das GPGPU, ressaltam-se as principais características sobre o modelo SPM, denominado pela NVIDIA como SIMT. Neste modelo, os dados são representados como um *stream*, ou seja, um fluxo de dados é classificado como um conjunto, e estruturado em um *array*. Este modelo descreve um *kernel* com um conjunto de instruções que processam um conjunto de dados de entrada. O *kernel* executa instruções paralelamente em todo o *stream*, utilizando-o como dado de entrada e saída [Kirk e Hwu 2010, NVIDIA 2010].

No modelo SIMT, a chamada de uma variedade de *kernels* é organizada como uma hierarquia de grupo de *threads*. O recurso de dividir *kernels* em blocos independentes, bem como o suporte eficiente a *threads* em GPGPU garante escalabilidade transparente e portável, permitindo um programa CUDA ser executado usando qualquer número de *cores*. Os *threads* são utilizados para paralelismo de granularidade fina e são agrupados formando blocos. Os blocos são utilizados para paralelismo de granularidade grossa e são agrupados formando uma grade o qual representa uma chamada de *kernel*. Esta hierarquia permite que cada *thread* dentro de um bloco, assim como, cada bloco dentro de uma grade, tenha um índice de identificação único [Hofmann 2010]. Neste modelo, os *threads* de um bloco podem ser organizados em três dimensões, para executar em um único multiprocessador na GPGPU. Também podem ser sincronizados e compartilhar dados com outros *threads* do mesmo bloco, por meio de memória compartilhada.

2.2 Método de Suavização baseado num Modelo Variacional

O uso de modelos variacionais tem sido adotados no desenvolvimento de métodos de suavização de imagens afetadas por ruído multiplicativo [Aubert e

Aujol 2008, Huang et al. 2009]. Jin e Yang [2011] propuseram um método promissor para remoção deste tipo de ruído em imagens usando o modelo variacional proposto por Rudin et al. [1992], e descrito como:

$$\min_u \left\{ J(u) + \lambda \int_{\Omega} (f-u)^2 \right\}, \quad (1)$$

onde Ω é um domínio fechado pertencendo a R^2, f é a imagem afetada pelo ruído, u é a imagem original na iteração atual, $J(u)$ é um termo regulador da equação, e λ é um parâmetro de peso.

O método proposto em [Jin e Yang 2011] foi projetado para remover ruído multiplicativo em imagens de ultra-sonografia. Para isso, os autores adotaram a função proposta por Krissian et al. [2005] para resolver o modelo variacional da Eq. (1) considerando:

$$E(u) = \int_{\Omega} \frac{(f-u)^2}{u}, \quad (2)$$

onde u é a imagem original, $f = u + \sqrt{ug}$ é a imagem afetada pelo ruído sendo g uma variável Gaussiana com média não nula, ou seja diferente de 0 (zero). Assim, o modelo variacional adotado por Jin e Yang [2011] pode ser descrito como:

$$\min_u \left\{ J(u) + \lambda \int_{\Omega} \left[\frac{(f-u)^2}{u} \right] \right\}, \quad (3)$$

onde $\lambda > 0$ é um parâmetro de peso.

Desta forma, o problema de remoção de ruído multiplicativo foi resolvido numericamente a partir do modelo dado pela Eq. (3) adotando:

$$\partial_t u = div\left(\frac{\nabla u}{|\nabla u|} \right) + \lambda \left(\frac{f^2}{u^2} - 1 \right), \quad (4)$$

onde ∇ é o operador gradiente e *div* o operador divergente.

Considerando $n = 1,2,...$ como sendo as iterações do modelo e o esquema de diferenças finitas [Rudin 1992], a Eq. (4) pode ser escrita em sua forma discreta como:

$$u_{i,j}^{n+1} = \Delta t \left[\frac{-D_x^+\left(u_{i-1,j}^n\right) + D_x^+\left(u_{i,j}^n\right)}{-\left|D_x\left(u_{i-1,j}^n\right)\right| + \left|D_x\left(u_{i,j}^n\right)\right|} + \frac{-D_y^+\left(u_{i,j-1}^n\right) + D_y^+\left(u_{i,j}^n\right)}{-\left|D_y\left(u_{i,j-1}^n\right)\right| + \left|D_y\left(u_{i,j}^n\right)\right|} \right] + \lambda^n \left(\frac{f^2}{\left(u_{i,j}^n\right)^2} - 1 \right) + u_{i,j}^n, \quad (5)$$

onde f é a imagem de entrada afetada pelo ruído.

Esta equação discreta é aplicada *n* vezes em todos os pontos da imagem, o que deixa o algoritmo de suavização com complexidade computacional *O(n.L.C)*, onde *L* e *C* são, respectivamente, o número de linhas e colunas da imagem de entrada. Nesta etapa do método, foi aplicada a paralelização em GPGPU para reduzir o tempo computacional associado a esta etapa do processamento, definido na implementação como *Kernel A*.

Por fim, o parâmetro λ é calculado automaticamente a cada nova iteração *n*, por meio da Eq. (6) e também foi paralelizado, definido na implementação como *Kernel B*:

$$\lambda^n = \frac{1}{\sigma^2} \left(\sum_{i,j} \left[Aux_{i,j}^n \right] \right) \quad (6)$$

onde σ^2 é a variância da imagem de entrada e

$$Aux_{i,j}^n = \left(D_x^- \left(\frac{D_x^+\left(u_{i,j}^n\right)}{\left|D_x\left(u_{i,j}^n\right)\right|} \right) + D_y^- \left(\frac{D_y^+\left(u_{i,j}^n\right)}{\left|D_y\left(u_{i,j}^n\right)\right|} \right) \right) \frac{\left(u_{i,j}^n - f\right) u_{i,j}^n}{u_{i,j}^n + f}.$$

3. Paralelização do Método

A implementação do algoritmo paralelizado em CUDA requer, inicialmente, o ajuste dos *kernels* de acordo com as especificações da arquitetura e o problema a ser decomposto.

Nesta pesquisa o problema paralelizado foi o método de suavização descrito na seção 2.2. O algoritmo é formado por dois pares laços "*for*" aninhados, sendo um deles utilizado para percorrer cada elemento da matriz bidimensional, utilizada para representar a imagem de entrada, e outro utilizado para recalcular os valores de cada elemento da matriz. Estes laços foram paralelizados em dois *kernels, A* e *B*, respectivamente.

O processamento de imagens, comumente, envolve a necessidade de cálculos em grandes quantidades de dados. Desta maneira, a primeira estratégia adotada foi alocar espaço na memória da placa de vídeo para a realização dos cálculos necessários.

As *threads* do *kernel A* realizam os cálculos da Eq. (5), em cada elemento da matriz de maneira independente. O cálculo da variável λ, descrito na Eq. (6), é paralelizado no *kernel B*, utilizando um vetor auxiliar para armazenar valores de cada iteração, ou seja, cada *thread* calcula o valor de uma iteração. Os *kernels* serão processados em imagens com dimensão reduzida (320×240 *pixels*), desta maneira os blocos não podem ser muito grandes para não subutilizar os CUDA *cores*. A quantidade de linhas da matriz de blocos foi determinado como sendo o número total de linhas da imagem de entrada dividido por 16 *threads* por bloco, e a quantidade de colunas como sendo o número total de colunas da imagem dividido por 16 *threads* por bloco.

4. Resultados e Discussão

O método de suavização descrito na seção 2.2, foi implementado no formato sequencial e paralelizado em CUDA. Os experimentos foram realizados em um computador equipado com processador Intel i7 860 de 2.8 GHz, com 8GB de memória RAM (DDR3 667MHz), sistema operacional Windows 7 versão 64 bits, placa gráfica GeForce GTX 450 (Nvidia) com 196 CUDA cores e 1GB de memória. O software utilizado para desenvolvimento foi o Microsoft Visual Studio 2010. A versão CUDA, instalada para os experimentos foi a 4.0. Os testes utilizaram sequências de imagem de

ultra-som, com dimensões de 320×240 *píxels* de resolução.

Os tempos médios calculados a partir de 50 execuções do algoritmo, para suavização de 255 quadros (*frames*) de um vídeo de ultra-som, foram aproximadamente 140 segundos para a versão sequencial e 39 segundos para a versão paralela. Isto signifca uma redução de tempo de processamento de aproximadamente 3.5 vezes para o algoritmo paralelo. O algoritmo de suavização foi executado usando 120 iterações e $\Delta t = 0.01$. O resultado da suavização num quadro retirado aleatoriamente do vídeo original é apresentado na Figura 1.

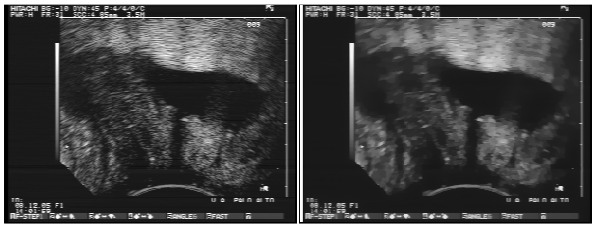

Figura 1: Resultado da suavização adotada neste trabalho. Imagem original à esquerda e suavizada à direita**.**

4. Conclusões

Neste artigo foi apresentada uma abordagem paralelizada, em GPGPU e utilizando a arquitetura CUDA, de um método de suavização de imagens baseado em um método variacional, aplicado a imagens de ultra-sonografia. A contribuição inicial é a utilização de um método de suavização eficiente, disponível para arquiteturas massivamente paralelas, oferecendo uma alternativa de alto desempenho em aplicações médicas.

Como trabalhos futuros pretende-se expandir os testes comparativos envolvendo outros modelos de paralelização.

Agradecimentos

Este trabalho foi parcialmente desenvolvido no escopo do projeto com a referência PTDC/EEA-CRO/103320/2008 financiado pela Fundação para a Ciência e a Tecnologia (FCT) de Portugal. O autor Alex F. de Araujo agradece à FCT por sua bolsa de Doutoramento com referência SFRH/BD/61983/2009. O primeiro autor agradece a Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES) do Brasil pelo financiamento de bolsa de Mestrado e a Universidade do Estado de Mato Grosso (UNEMAT), do Brasil, pelo apoio concedido.

Referências

AUBERT, G.; AUJOL, J.-F., 2008. A VARIATIONAL APPROACH TO REMOVING MULTIPLICATIVE NOISE., SIAM JOURNAL OF APPLIED MATHEMATICS 68 (4) 925–946.

BORGO, R.; BRODLIE, K., 2009. STATE OF THE ART REPORT ON GPUS. SCHOOL OF COMPUTING, VISUALIZATION AND VIRTUAL REALITY RESEARCH GROUP.

GEBALI, F., 2011. ALGORITHMS AND PARALLEL COMPUTING. JOHN WILEY & SONS.

GURD, J. A., 1988. TAXONOMY OF PARALLEL COMPUTER ARCHITECTURES. IN: INTERNATIONAL SPECIALIST SEMINAR ON THE DESIGN AND APPLICATION OF PARALLEL DIGITAL PROCESSORS.

HENNESSY, J. L.; PATTERSON, D. A., 2007. COMPUTER ARCHITECTURE A QUANTITATIVE APPROACH. 4TH. ED. ELSEVIER.

HOFMANN, M.; BINNA, T., 2010. MASSIVE PARALLEL IMAGE PROCESSING. RELATÓRIO TÉCNICO, DEPARTMENT OF COMPUTER SCIENCE UNIVERSITY OF APPLIED SCIENCE RAPPERSWIL.

JIN, Z.; YANG, X., 2011. A VARIATIONAL MODEL TO REMOVE THE MULTIPLICATIVE NOISE IN ULTRASOUND IMAGES, JOURNAL OF MATHEMATICAL IMAGING AND VISION 39 (1) 62–74.

KIRK, D.; HWU, W.-M., 2010. PROGRAMMING MASSIVELY PARALLEL PROCESSORS: A HANDS-ON APPROACH. ELSEVIER.

KRISSIAN, K., KIKINIS, C.-F., WESTIN, K., 2005. SPECKLE-CONSTRAINED FILTERING OF ULTRASOUND IMAGES, IN: PROCEEDINGS OF THE 2005 IEEE COMPUTER SOCIETY CONFERENCE ON COMPUTER VISION AND PATTERN RECOGNITION (CVPR'05) - VOLUME 2 - VOLUME 02, CVPR '05, IEEE COMPUTER SOCIETY.

NVIDIA., 2010. GPU TUTORIAL: BUILD ENVIRONMENT, DEBUGGING/PROFILING, FERMI, OPTIMIZATION/CUDA 3.1 AND FERMI ADVICE.

PAGE, D., 2009. A PRACTICAL INTRODUCTION TO COMPUTER ARCHITECTURE. SPRINGER.

PARHAMI, B., 2002. INTRODUCTION TO PARALLEL PROCESSING: ALGORITHMS AND ARCHITECTURES. KLUWER ACADEMIC PUBLISHERS. (PLENUN SERIES IN COMPUTER SCIENCE, ISBN: 0-306-45970-1).

RUDIN, S.; OSHER, E. FATEMI, 1992. NONLINEAR TOTAL VARIATION BASED NOISE REMOVAL ALGORITHMS, JOURNAL OF PHYSICA D 60 (1-4) 259–268.

SANDERS, J.; KANDROT, E., 2011. CUDA BY EXAMPLE: AN INTRODUCTION TO GENERAL-PURPOSE GPU PROGRAMMING. ADDISON-WESLEY.

TANENBAUM, A. S., 2010. SISTEMAS OPERACIONAIS MODERONS. PEARSON EDUCATION DO BRASIL.

VADJA, A., 2011. PROGRAMMING MANY-CORE CHIPS. [S.L.]: SPRINGER, 2011. 241 P.

Y.-M. HUANG; NG, M. K., WEN, Y.-W., 2009. A NEW TOTAL VARIATION METHOD FOR MULTIPLICATIVE NOISE REMOVAL, SIAM JOURNAL ON IMAGING SCIENCES 2 (1) 20–40.