Multiscale AM-FM Decompositions with GPU acceleration for Diabetic Retinopathy Screening

Cesar Carranza*,†, Victor Murray*, Marios Pattichis*, and E. Simon Barriga‡
*Electrical and Computer Engineering Department
University of New Mexico, Albuquerque, New Mexico 87131
Emails: acarran@unm.edu, vmurray@ieee.org, pattichis@ece.unm.edu.
†Sección Electricidad y Electrónica, Pontificia Universidad Católica del Perú, Lima-32, Perú
‡VisionQuest Biomedical, Albuquerque, New Mexico 87106. Email: sbarriga@visionquest-bio.com

***Abstract*—A Computer Aided Diagnosis system based on multiscale amplitude-modulation frequency-modulation (AM-FM) methods has been recently developed for discriminating between normal and pathological retinal images. The original Matlab implementation of this system required large amounts of computational time and memory resources that would not permit real-time patient consultation. In this manuscript, we present a new implementation of the multiscale AM-FM decomposition, converted from MATLAB code into C/CUDA (Compute Unified Device Architecture) code, in order to take advantage of the graphics processing units (GPU) to significantly reduce the running time and memory resources.**

To perform the AM-FM decomposition at high speed, the original image is read into the main memory (host side, in the personal computer) and transferred to the global memory in the CUDA card (device side, the GPU), where the AM-FM decomposition is computed at thirteen different frequency scales. The results are transferred back to the host and saved in the local hard drive. The intensive mathematical operations are parallelized to achieve maximum utilization of the CUDA card.

This new implementation runs 20 times faster than an optimized, parallel implementation in MATLAB running on an personal computer with an Intel Xeon Processor W3520, and the memory requirement is reduced from 4GB to below 0.9GB for images of 2252×1996 pixels or smaller. The required resources for the new implementation can be found in entry-level CUDA cards that are currently available.

***Index Terms*—AM-FM, diabetic retinopathy, GPU, parallel processing.**

I. Introduction

According to the National Institutes of Health (NIH), diabetes is the leading cause of new cases of blindness among adults from 20 to 74 years old [1]. In the United States alone, in 2005-2008, 4.2 million people (28.5%) with diabetes ages 40 years or older had diabetic retinopathy, and of these, 4.4% had advanced diabetic retinopathy (DR) that could lead to severe vision loss. It is estimated that over 10 million diabetics do not receive the recommended annual eye examinations, significantly increasing their risk of blindness.

A system for automatic DR screening has been presented by Agurto et al. in [2], [3]. One of the requirements of an efficient screening system is that it produces real-time results that can be communicated to the patient while the patient is still in the doctor's office. The complexity of the original system implementation makes this a daunting task. This paper presents an efficient implementation using a Compute Unified Device Architecture (CUDA) card.

Recently, graphics processing units (GPUs) have emerged as viable alternatives to general purpose microprocessors for performing complex computations. Medical image processing is an area that can take full advantage of this new paradigm due to the amount of data and/or the complexity of some of the processing methods associated with computer-aided detection applications. For example, Moulik and Boonn [4] presented examples from tracking and registration, whereas Suh et. al. presented an implementation for multiple sclerosis in [5]. More generally, several CUDA implementations of Wavelets have appeared since 2008 [6], [7], [8], [9]. Additionally, Yoo et al. presented an investigation of multiscale approaches for image fusion algorithms in [10]. In this manuscript, our focus is on the development of a real-time, multiscale, amplitude-modulation frequency-modulation (AM-FM) analysis system for medical applications. In terms of the hardware/software implementation, the goal of this work is to reduce the memory requirements below the maximum global memory available in low-level CUDA cards, and to be able to use the GPU to reduce the processing time of the AM-FM decomposition to only a few seconds.

This manuscript is organized as follows. In section II we describe the methodology used for multiscale image processing given the hardware constraints. section III shows the performance results. Conclusions are given in section IV.

II. Methods

A. Amplitude-Modulation Frequency-Modulation (AM-FM) Decomposition

A digital image can be represented in terms of its instantaneous amplitude and frequency-modulation components as [11]:

$$I(k_1, k_2) \approx \sum_{n=1}^{M} a_n(k_1, k_2) \cos \varphi_n(k_1, k_2), \qquad (1)$$

where (k_1, k_2) represents the discrete coordinates, M is the number of AM-FM frequency scales, a_n represent the instantaneous amplitude functions (IA), and $\varphi_n(x, y)$ represent the instantaneous phase functions.

For each AM-FM component, the instantaneous frequency (IF) is defined in terms of the gradient of the phase $\varphi_n(x, y)$:

$$\nabla\varphi_n(x, y) = \left(\frac{\partial\varphi_n(x, y)}{\partial x}, \frac{\partial\varphi_n(x, y)}{\partial y} \right). \quad (2)$$

IF estimates are obtained using the variable spacing, local linear phase method described in [11]. AM-FM demodulation is performed using 31 bandpass filters associated with a 5-scale filterbank. These frequency scales are used to define thirteen Combinations of Scales (CoS) that span the image frequency spectrum (see [2] for its application to diabetic retinopathy). From each-scale, an AM-FM component is estimated using dominant component analysis applied to the channels that fall within each scale.

B. CPU-GPU System and Resources

All the implementations in this work (MATLAB and C/CUDA AM-FM decomposition) were developed and tested using the same system. Our hardware system is composed of: (i) an Intel Xeon Processor W3520 (CPU) running at 2.67GHZ, with 6GB of memory, (ii) a NVIDIA GeForce GTX465 card with 1GB of global memory and 352 CUDA cores running at 607MHz, and (iii) a Hard Drive Seagate ST3500418AS, 7200 RPM, SATA 3Gb/s. Our software system was implemented using a Windows 7 Professional, 64-bit OS, running MATLAB 7.11.0.584 (R2010b 64 bit), CUDA SDK v3.2 (64bit), and Microsoft Visual Studio 2010 Professional (64bit).

C. MATLAB implementation of the multiscale AM-FM decomposition

This version was implemented based on [11], [2], using a 5-scale filterbank, 31 filters and 13 CoS. The required algorithm for computing AM-FM decompositions can be summarized as follows:

- Read image from hard drive, decompress, extract green channel, and zero-pad the image.
- Generate the analytic image.
- Generate the filterbank (31 bandpass filters).
- Generate the 13 CoS.

This implementation keeps all the data in memory, which is about 3GB, and is designed to be executed sequentially. The use of the MATLAB Parallel Toolbox allows us to parallelize the filtering of the images among the 31 bandpass filters with a performance gain when compared to a purely sequential implementation. The other functions work sequentially.

D. C/CUDA implementation of the multiscale AM-FM decomposition

We begin by showing in Fig. 1 a general representation of the system that it is used in this work. The image to be processed is loaded from the hard drive in the personal computer to the system memory (host, left part in Fig. 1) and then it is transferred to the global memory (device, right part in Fig. 1) via the PCI Express bus. Next, the CUDA cores (processors) perform the AM-FM decomposition and the results are stored in the global memory to be transferred back to the host. To perform the image processing computation at high speed, all the processing must be performed on the device memory by using the cache memory as much as possible. There is an L1 cache for each multiprocessor and an L2 cache shared by all multiprocessors. Both are used to cache accesses to global memory. The same on-chip memory is used for both L1 and shared memory.

In terms of the implementation, this version was based on the work presented in [11], [2] but with memory and parallel optimizations for CUDA. Fig. 2 shows the block diagram of the algorithm's implementation.

First, the algorithm generates the analytic image (Fig. 2, ***block (a)***), then one custom filter per direction (two in total for an image) is generated at a time (Fig. 2, ***block (b)***). With those three computed signals, we process one of the components of the filterbank (Fig. 2, ***block (c)***), and generate a CoS result that is sent to the host for storage (Fig. 2, ***block (d)***). This process is repeated 31 times (from the 31 bandpass filters used) and at the end we complete the computation of the 13 CoS's in the host side.

The main bottleneck in the use of GPUs is the data transfer between the host and the device (with peaks up to 3GB/s). In order to minimize this issue, there are only 32 significant transfers in the whole process: (i) 1 to move the green channel from host to device, and (ii) 31 transfers to move each CoS from the device to host. Additionally, a small amount of data is transferred to the device to generate the multiscale filterbank (bandpass filters' coefficients).

The AM-FM decomposition is performed by dividing the task into several threads. The type of threads can be divided into three groups: (i) FFT (Fast Fourier Transform) threads, (ii) BLAS (Basic Linear Algebra Subprograms) threads and (iii) custom threads (see [12]):

- *FFT threads:* Implemented using the CUFFT Library (CUDA FFT) that implements the following DFT (Discrete Fourier Transform) building blocks: radix-2, radix-3, radix-5 and radix-7. The performance of any transform size that can be factored as $2^a \times 3^b \times 5^c \times 7^d$ (where $a, b, c,$ and d are non-negative integers) is optimized in the CUFFT library. In our particular case, our images of 2224×1888 are padded to $2304 \times 2048 = (2^8 \times 3^2) \times (2^{11})$.
- *BLAS threads:* Implemented using the CUBLAS Library (CUDA BLAS). The basic operation to use the CUBLAS library is to create matrix/vector objects in the global memory, fill them with data, call a sequence of CUBLAS functions, and store the results in the same global memory. We use BLAS functions to create matrices/vectors, find maximum values and to perform matrix multiplications.
- *Custom Threads:* They perform specific tasks for the AM-FM decomposition: (i) point to point matrix multiplications, (ii) create masks to generate analytic images, (iii) complex numbers operations, (iv) image padding/cropping, (v) vector normalization, and (vi) cal-

TABLE I
RUNNING TIME TO PROCESS ONE IMAGE OF 2224×1888 PIXELS UNDER DIFFERENT IMPLEMENTATIONS.

	Standard Desktop	**Optimized system and code**	**Parallel toolbox**	**Port to compiled code/use GPUs**
Code	MATLAB	MATLAB	MATLAB	C/CUDA
Running Time	24min	6.5min	5min	15sec
Specific Changes	(Original version.)	Increased CPU power, more RAM memory. Removed HD access for buffering.	Use of multi-core processing (4 cores).	Move from MATLAB to C. Use of GPUs (352 cores).

TABLE II
RUNNING TIME TO PROCESS THE MULTISCALE AM-FM DECOMPOSITION IN THE CUDA CARD (SEE FIG. 2).

Process	**Time (ms)**
Read, decompress and transfer image and filters data from the host to the device.	110
Generate padded image.	14
Generate the analytic image and produce 2D filters.	20
Filter image in the frequency domain using FFT.	$31 \times 22 = 704$
Space domain filtered image and pad removal.	$31 \times 42 = 1344$
Computation of AM-FM estimates (see (1) and (2)).	$31 \times 160 = 5120$

TABLE III
DIFFERENCES IN THE AM-FM ESTIMATIONS BETWEEN THE FASTEST MATLAB IMPLEMENTATION AND THE C/CUDA VERSION.

AM-FM estimate	**Average**
Instantaneous Amplitude	0
Instantaneous Frequency (magnitude)	$< 10^{-11}$
Instantaneous Frequency (angle)	$< 10^{-11}$

culus of the displacements. In all cases, the threads are designed to maximize all the parallel processing, memory coalescing and minimizing conditional branches. In particular, the displacements in the AM-FM computation (see [11] for more details) are performed in place to avoid intermediate storage.

III. RESULTS

For the test, we used retinal images of size 2224×1888 pixels. In Table I we compare the computation time to process one image under different implementations (MATLAB in different versions and CUDA). The first three cases show the improvement in the running time for the MATLAB implementation, and the last one shows our results using GPUs.

The current processing time can be divided into:

- About 7 seconds are spent in the device side performing the AM-FM decomposition. Some specific times for each process are shown in Table II (see also small typeface letters in Fig. 2).
- About 1 second is used at the beginning to read the image, initialize the CUDA card, read the image, decompress and transfer it.
- The remaining 7 seconds are spent saving the results in the hard drive.

Since the MATLAB implementation uses different resources of the system than the C/CUDA version, we measured the differences in the results as shown in Table III. The differences are negligible, thus the results in practice are the same (note that MATLAB version was tested in [11]).

IV. CONCLUSIONS

We have presented a real-time CUDA implementation of a multiscale AM-FM based diabetic retinopathy (DR) screening system. This is the first implementation of AM-FM on a GPU for medical applications and the dramatic reduction in the running time will allow its widespread use in complex medical problems. In this particular case, the DR screening system is now able to process the patient images in real-time. C/CUDA code implementation reduces the running time significantly. However, as a trade-off, the time and resources needed to write the code in C/CUDA is higher than MATLAB. Also, the difference in the cost between the hardware with and without the CUDA card is less than 20%. Porting the code to C/CUDA is not straightforward. To achieve an efficient and fast implementation, specific changes in the algorithm are needed, and strong knowledge of the GPU hardware architecture is required.

The work developed in C/CUDA in this manuscript requires a NVIDIA CUDA card with compute capability of 2.0 or higher (double precision floating point unit). This code is portable to other CUDA cards with that compute capability and at least 1GB of global memory.

ACKNOWLEDGMENT

This work was supported by the National Eye Institute under Grants EY018280 and EY020015.

REFERENCES

[1] U.S. Department of Health and Human Services, "National diabetes statistics," February 2011, NIH Publication No. 11-3892.

[2] C. Agurto, V. Murray, E. Barriga, S. Murillo, M. Pattichis, H. Davis, S. Russell, M. Abramoff, and P. Soliz, "Multiscale AM-FM methods for diabetic retinopathy lesion detection," *IEEE Transactions on Medical Imaging*, vol. 29, no. 2, pp. 502–512, 2010.

[3] C. Agurto, S. Barriga, V. Murray, S. Murillo, G. Zamora, W. Bauman, M. Pattichis, and P. Soliz, "Toward comprehensive detection of sight threatening retinal disease using a multiscale AM-FM methodology," in *Medical Imaging 2011: Computer-Aided Diagnosis*, Ronald M. Summers and Bram van Ginneken, Eds. 2011, vol. 7963, p. 796316, SPIE.

[4] S. Moulik and W. Boonn, "The role of GPU computing in medical image analysis and visualization," in *Medical Imaging 2011: Advanced PACS-based Imaging Informatics and Therapeutic Applications*, William W. Boonn and Brent J. Liu, Eds. 2011, vol. 7967, p. 79670L, SPIE.

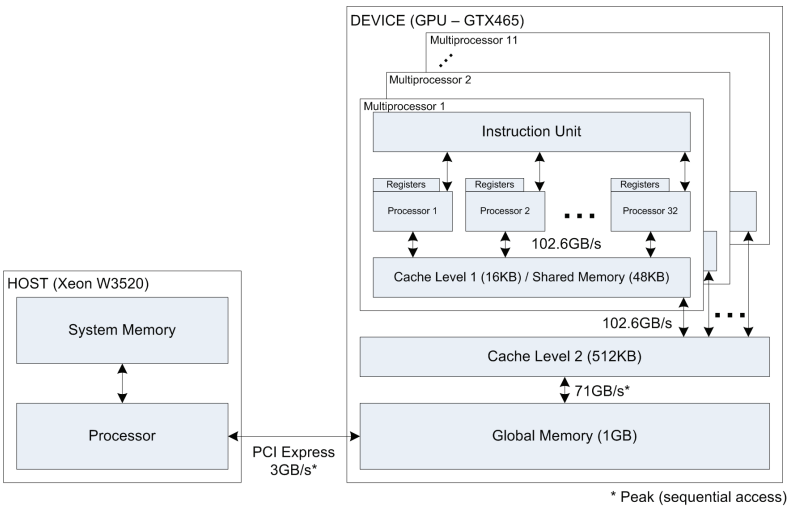

Fig. 1. Representation of the system in terms of the hardware used. The block of the left represents the host system (the personal computer). The block of the right represents the device where all the computations are performed (the CUDA card).

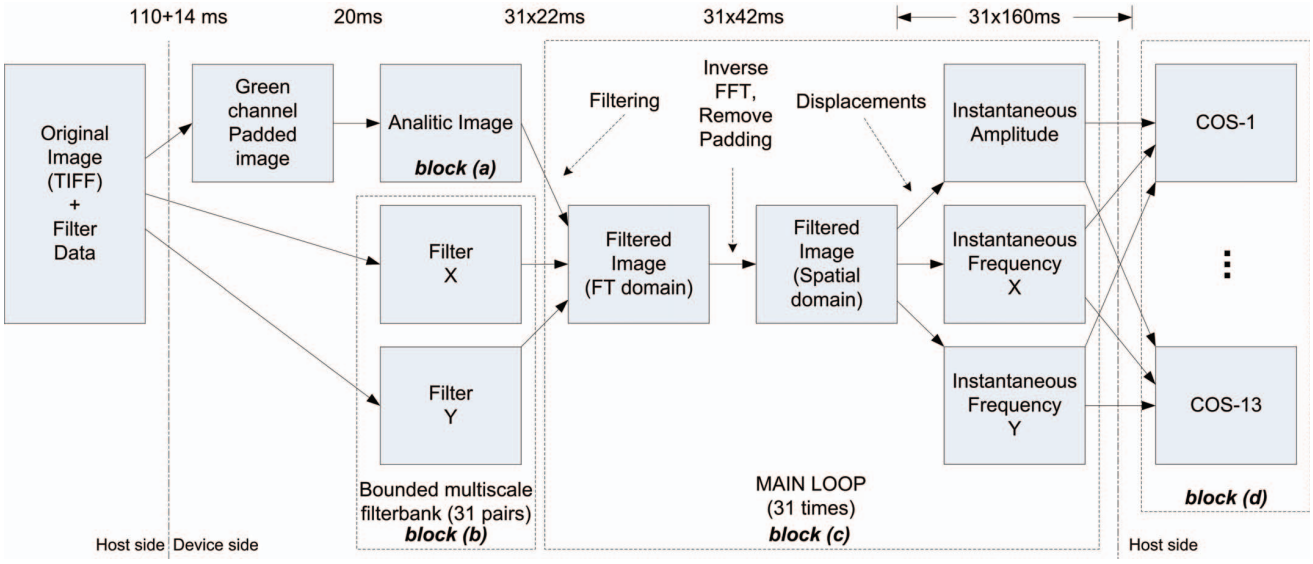

Fig. 2. AM-FM Decomposition block diagram of the algorithm to be processed on a GPU. The small typeface letters on the top of each block shows the estimated time used for each sub-process.

[5] Joohyung Suh, Kevin Ma, and Anh Le, "Improvement of MS (multiple sclerosis) CAD (computer aided diagnosis) performance using C/C++ and computing engine in the graphical processing unit (GPU)," in *Medical Imaging 2011: Advanced PACS-based Imaging Informatics and Therapeutic Applications*, William W. Boonn and Brent J. Liu, Eds. 2011, vol. 7967, p. 79670V, SPIE.

[6] V. Simek and R. R. Asn, "GPU acceleration of 2D-DWT image compression in MATLAB with CUDA," in *Proc. Second UKSIM European Symp. Computer Modeling and Simulation EMS '08*, 2008, pp. 274–277.

[7] W. J. van der Laan, J. B. T. M. Roerdink, and A. C. Jalba, "Accelerating wavelet-based video coding on graphics hardware using CUDA," in *Proc. 6th Int. Symp. Image and Signal Processing and Analysis ISPA 2009*, 2009, pp. 608–613.

[8] Wang Jianjun, Lu Wenlong, Liu Xiaojun, and Jiang Xiangqian, "High-speed parallel wavelet algorithm based on CUDA and its application in three-dimensional surface texture analysis," in *Proc. Int Electric Information and Control Engineering (ICEICE) Conf*, 2011, pp. 2249–2252.

[9] J. Franco, G. Bernabe, J. Fernandez, and M. E. Acacio, "A parallel implementation of the 2d wavelet transform using CUDA," in *Proc. 17th Euromicro Int Parallel, Distributed and Network-based Processing Conf*, 2009, pp. 111–118.

[10] Seung-Hun Yoo, Jin-Hyung Park, and Chang-Sung Jeong, "Accelerating multi-scale image fusion algorithms using CUDA," in *Proc. Int. Conf. of Soft Computing and Pattern Recognition SOCPAR '09*, 2009, pp. 278–282.

[11] V. Murray, P. Rodriguez, and M. S. Pattichis, "Multiscale AM-FM demodulation and image reconstruction methods with improved accuracy," *IEEE Transactions on Image Processing*, vol. 19, no. 5, pp. 1138–1152, 2010.

[12] NVIDIA, "CUDA toolkit & SDK," 2011.