Fingerprint grid enhancement on GPU

Raja Lehtihet[1], Wael El Oraiby[2], and Mohammed Benmohammed[3]

[1]Computer Science Departement, University of Constantine, Constantine, Algeria
[2]AIFU Ltd, Montreal, Quebec, Canada
[3]LIRE Laboratory, University of Constantine, Constantine, Algeria

Abstract— *This paper presents an optimized GPU (Graphics Processing Unit) implementation for fingerprint images enhancement using a Gabor filter-bank based algorithm. Given a batch of fingerprint images, we apply the Gabor filter bank and compute image variances of the convolution responses. We then select parts of these responses and compose the final enhanced batches. The algorithm exploits GPU parallelism by partitioning the data elements on the GPU parallel threads. The implementation was tested on different batch sizes and different image qualities from the FVC2004 DB2 database. We then compare the execution speed between the CPU and GPU. This comparison shows that the algorithm is by order of magnitudes faster on a GPU than the CPU.*

Keywords: biometrics, fingerprints, enhancement, Gabor filtering, GPGPU.

1. Introduction

fingerprint identification represents one of the most efficient and lowest cost detection systems in the biometric security market.

A fingerprint image presents a flow-like ridge structure. The structure of the ridges contains many local interesting characteristics such as islands, short ridges, enclosures, ridges endings and bifurcations. The ridge endings and bifurcations (called minutiae) are the most prominent identification characteristics.

A person is usually identified by an automated Fingerprint Identification Systems (AFIS) by matching his fingerprint minutiae-based signature with registered ones [14]. The result of such matching depends heavily on the quality of the input fingerprint image. However, the ridge structures and minutiae are not always well defined because of the presence of spurious features and discontinuities due to acquisition parameters and/or to reasons inherent to the fingerprint owner. Thus, fingerprint enhancement is a crucial step in a fingerprint identification process where an enhancement algorithm must retrieve and enhance the ridge structure for further minutiae extraction.

Several approaches for fingerprint image enhancement were proposed. they are often based on flow orientation and local ridge directional binarization [17]. In [13] frequency and orientation filters in the Fourier domain were designed. This method is computation intensive since it involves transformation to the frequency domain and multiple filtering. In [10] the properties of orientation and ridge frequency as parameters for a single Gabor filter were used and a short-time Fourier transform was proposed in [5].

Enhancing fingerprint images in real-time is a challenge given the computation time required in the process. With the current generation of programmable Graphics Processing Units (GPU), this is now possible since they currently have teraflops of floating point performance. General Purpose computing or programming on GPUs (GPGPU) was introduced as a parallel programming model for these devices, where developers decompose the problem into sub processing elements to exploit high level parallelism of the GPU.

Computer scientists and researchers are starting to use GPUs for running computational scientific applications. First, in [4], color image processing was mapped for GPU programming. In [1], Fast Fourier Transform (FFT) on GPU was computed giving faster execution time than on CPU. A set of frameworks for GPGPU processing are proposed such as: Image processing framework [12], the OpenVidia library [6], GPU accelerated generalized bi-dimensional distance transform [19], motion estimation [18], GPU4Vision for real-time optical flow [22] and total variation based image segmentation on GPU [21]. Now Cuda [15], DirectCompute [16] and OpenCL [8] are proposed as GPGPU programming APIs, allowing programmers to interface with the GPU directly to make massively parallel programs.

This paper presents a GPU implementation for enhancing batches of fingerprint images using a Gabor filter bank based algorithm in an accelerated execution time. The algorithm selects pixels corresponding to the maximum values of variances in the Gabor responses. The Gabor based enhancement of fingerprints has shown good results in works of [10], [9] and [3]. The algorithm scales very well on multiple core GPUs.

Experimental results with fingerprint images from the FVC2004 DB2 database [7] show that the execution of the algorithm gives enhanced images by order of magnitudes faster on the GPU in comparison to the CPU.

2. GPU Programming

In the GPGPU programming model, the CPU acts as a master controlling the GPU which acts as a slave. The GPU is composed of multiple cores and is designed to execute the same program called a kernel on different data elements simultaneously. These kernels are executed in threads divided across the multiple cores (see Figure 1). These threads are mapped relatively to the data elements, where each element is consumed in its own thread. For image processing algorithms such as filtering, that are designed to work on pixel blocks independently of previous steps, the GPGPU model is ideal. When a step is dependent on the previous one, the algorithm can be divided into multiple execution passes executed one after the other on the GPU as well.

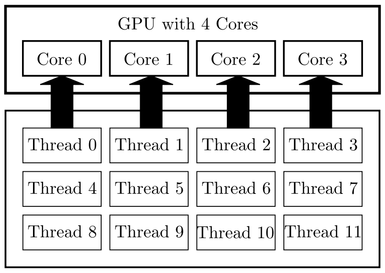


Fig. 1: Multi threaded program partitioned into blocks of threads.

In this direction, computing the FFT on GPU has been addressed successfully and efficiently with fast algorithms such as [2], [15], [20]. Most of these algorithms use butterfly algorithm [2] with multi-passes for different levels of the algorithm (where each level requires a pass).

3. Fingerprint enhancement

3.1 2-D Gabor wavelets

Let $G(x, y; \theta, \lambda)$ be the Gabor filter function centred at the origin with $1/\lambda$ as the spatial frequency and θ as the orientation. The response of a Gabor filter to an image is obtained by a 2D convolution operation, we can proceed by convoluting pixels of the image with an even symmetric filter [11] that can be constructed as follows:

$$g(x, y; \theta, \lambda) = exp\left(-\frac{1}{2}\left(\frac{x_\theta^2}{\sigma_G^2} + \frac{y_\theta^2}{\sigma_G^2}\right)\right)\cos(\frac{2\pi}{\lambda}x_\theta) \quad (1)$$

$$x_\theta = x\cos\theta + y\sin\theta \quad (2)$$

$$y_\theta = -x\sin\theta + y\cos\theta \quad (3)$$

Figure 2 shows one response of a Gabor filter: a filter with 45° of orientation, here we notice that high responses are located wherever there are ridges with the same orientation.

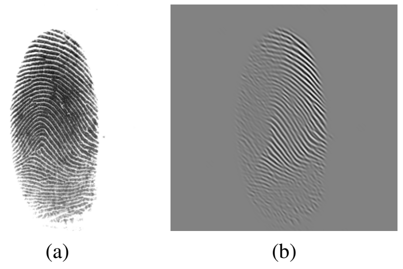


Fig. 2: (a) Original fingerprint image. (b) Response for a Gabor filter of orientation $\theta = 45°$

3.2 Gabor filter based enhancement algorithm

First we combine a set of $m = n^2$ 2D discreet fingerprint images with L-Gray levels into a tiled image I of $n \times n$ tiles. The image I is made of $W \times H$ pixels and $I(x, y)$ designates a pixel in this image (where W is the image width, H the image height, $0 \leq x < W$ and $0 \leq y < H$).

The proposed algorithm is composed of several stages as resumed in Figure 3:

- **Gabor filtering:** Apply a Gabor filter bank of 8 different orientations and 3 different frequencies to I. The result is made of 24 response images $\{R_0 \dots R_{23}\}$ for every set.
- **Variance images computing:** Compute the local variance on the pixel neighbouring $b \times b$ of each Gabor response image of the 24 images resulting from the precedent filtering, this will give us 24 variance images, $\{V_0 \dots V_{23}\}$.

$$V(x, y) = \frac{1}{b^2}\sum_{s=0}^{b-1}\sum_{t=0}^{b-1}(I(x-s, y-t) - \mu(x, y))^2 \quad (4)$$

where μ is the mean gray level of the $b \times b$ block:

$$\mu = \frac{1}{b^2}\sum_{s=0}^{b-1}\sum_{t=0}^{b-1} I(x-s, y-t) \quad (5)$$

The mean was computed using a two-passes filter (one for x and then for y). This leads to significant increase in speed and reduces the bottlenecks on the GPU significantly.

- **Best coefficient selection:** Let $T_i(x, y) = (R_i(x, y), V_i(x, y))$, be the tuple linking the pixel $R_i(x, y)$ with the variance value $V_i(x, y)$. We select the pixels with maximum local variance such that the final pixel $P(x, y) = \mathrm{Arg}\max_{i \in [0,23]} T_i(x, y)$.
- **fingeprint binarization:** Binarize the image using a pixel values threshold.

4. Implementation

For the implementation we chose the following values: $b = 15$, $n \in \{2.04.0, 8.0\}$, $\sigma = 4$, $\lambda \in \{6.0, 8.0, 10.0\}$ and

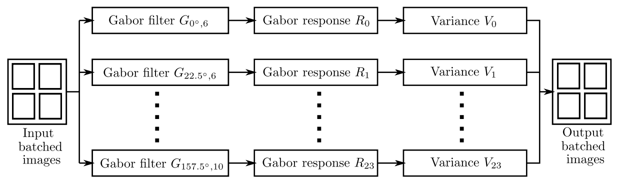


Fig. 3: Overview of the Gabor based enhancement algorithm.

$\theta \in \{0, 22.5, 45, 67.5, 90, 112.5, 135, 157.5\}$. Both λ and σ are empiric values since they provide best response to our image set. The orientations are chosen based on the work done in [9], [11] while the λ values were chosen related to fingerprint image resolutions. $\lambda = 8$ is the average value.

First the $n \times n$ target images are grouped into one image, then 24 Gabor bank images are constructed in the frequency domain. The resulting image is then transformed to frequency domain using FFT as well. We then multiply this image with all the 24 filters and store the results in 24 Gabor response images. These Gabor response images are then transformed back with inverse FFT to spatial domain. For each transformed Gabor response image we compute the variance using (4) in a different result image where the mean (5) was computed through FFT. Processing variance in this way is numerically more stable. Finally the pixels who have the maximum variance from the 24 Gabor response images are copied to the final image.

The CPU implementation was done in C language using FFTW in single precision, while the GPU version was coded in CUDA given its fast and stable implementation of FFT. On the GPU, the 24 Gabor filter banks are created in the frequency domain. These filters are noted G_i^F. The original image is converted to the frequency domain image I^F. The algorithm works in 4 passes, and for each pixel in every pass, a thread on the GPU is allocated. The memory needed for all operations is allocated before we enter the execution phase. This is needed to prevent overhead spent in data copy and resource synchronization between the GPU and the CPU.

Moreover, shared memory was used to compute variance. The speed increase is dramatic since it reduces GPU cores idling time waiting for memory fetches. There were between 12 to 16 speed increase using this approach on the current test configuration.

The 4 passes of the algorithm are as follows:

- **Pass 1:** Each gabor filter is multiplied with the frequency domain image, this will give a Gabor response image: $R_i^F = G_i^F I^F$. Depending on the GPU power and memory, these multiplications can all be done simultaneously.
- **Pass 2:** R_i^F is converted back to spatial domain, giving R_i.
- **Pass 3:** The variance image V_i is computed from R_i.
- **Pass 4:** Once all variance images are done, the last step is to select pixels with highest variance.

Thus, the chosen pixel is the one for $P(x, y) = \operatorname{Arg\,max}_{i \in [0,23]} T_i(x, y)$.

5. Experimental results

Table 1: Performance on GPU in milliseconds.

	GPU		
Batch size	4	16	64
Algorithm	151.3	413.09	1707.63
FFT/Multiplication (24 imgs)	99.17	257.08	1087.02
Variance(24 imgs)	19.68	77.28	348.24
ArgMax(24 imgs)	28.31	65.23	212.65
CPUMem to GPUMem	0.77	3.95	15.71
GPUMem to CPUMem	3.37	9.55	44.01

Table 2: Performance on CPU in milliseconds.

	CPU		
Batch size	4	16	64
Algorithm	1128.74	5025.56	20115.39
FFT/Multiplication (24 imgs)	1012.10	4467.28	17838.72
Variance(24 imgs)	31.92	228.4	970.54
ArgMax(24 imgs)	84.72	329.88	1306.13

We tested the implementations of the algorithm on an nvidia GeForce 560 Ti GPU with 192 cores running at 900Mhz and with 1024MB video memory. The test was also performed on an intel i7-2600K CPU at 3.40GHz and with 8GB of memory. The tests were performed on both Windows 7 and Ubuntu 12.04.

Results on GPU and CPU are shown in Tables 1 and 2. We applied the algorithm implementation on image batches from the FVC2004 DB2 database with batches of 4, 16 and 64 images. The graphical representations of these tables (as shown in Figure 4) show that for our fingerprint enhancement implementation, the GPU is at least 11 times more efficient than a CPU running at a significantly higher clock rate.

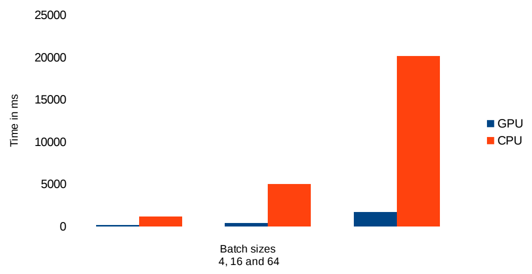
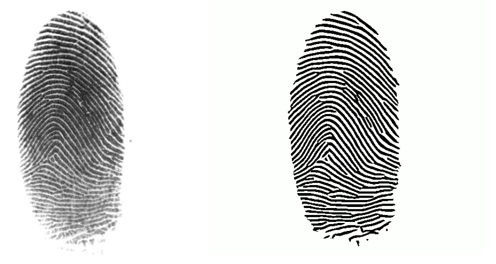


Fig. 4: GPU vs CPU performance

Figure 5 shows an example of the final result of the enhancement algorithm.



(a) (b)

Fig. 5: (a) Original fingerprint image. (b) fingerprint image after Gabor enhancement application.

6. Conclusion

In this paper, we presented a study of the implementation on CPU and GPU of a fingerprint enhancement algorithm. The GPU implementation was done in an optimal way giving an accelerated time execution up to at least 11 times the CPU execution time. The used algorithm is based on a Gabor filter bank convolution and variance computing which are both costly on CPU.

Experimentations were realized on the FVC2004 fingerprint image database, as shown in the graphical representations of time execution values, the algorithm gives good enhancement results in an accelerated time execution.

Finally, the obtained results are encouraging to implement other costly fingerprint enhancement algorithms in a batched way in order to reduce processing time. Thus, future work will be oriented to process images using the big possibilities of GPGPU programmming.

References

[1] Edward Angel and Kenneth Moreland. Integrated image and graphics technologies. chapter Fourier processing in the graphics pipeline, pages 95–110. Kluwer Academic Publishers, Norwell, MA, USA, 2004.

[2] Eric Bainville. Opencl fast fourier transform. `http://www.bealto.com/gpu-fft_dft.html`, 2010.

[3] Sylvain Bernard, Nozha Boujemaa, David Vitale, and Claude Bricot. Fingerprint segmentation using the phase of multiscale gabor wavelets, 2002.

[4] Nabil Boukala, Jerome Da Rugna, and Universite Jean Monnet. Fast and accurate color image processing using 3d graphics cards. In *In Proceedings Vision, Modeling and Visualization*, 2003.

[5] Sharat Chikkerur, Chaohang Wu, and Venu Govindaraju. A systematic approach for feature extraction in fingerprint images. In *ICBA*, pages 344–350, 2004.

[6] James Fung and Steve Mann. Openvidia: parallel gpu computer vision. In *Proceedings of the 13th annual ACM international conference on Multimedia*, MULTIMEDIA '05, pages 849–852, New York, NY, USA, 2005. ACM.

[7] FVC2004. Fingerprint database. `http://bias.csr.unibo.it/fvc2004/`, 2004.

[8] Khronos group. Opencl khronos group. `http://www.khronos.org/opencl/`, 2011.

[9] Lin Hong, Anil K. Jain, Sharath Pankanti, and Ruud Bolle. Fingerprint enhancement. Technical Report MSU-CPS-96-45, Department of Computer Science, Michigan State University, East Lansing, Michigan, January 1996.

[10] Lin Hong, Yifei Wan, and Anil Jain. Fingerprint image enhancement: Algorithm and performance evaluation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20:777–789, 1998.

[11] Anil K. Jain, Salil Prabhakar, Lin Hong, and Sharath Pankanti. Filterbank-based fingerprint matching. *IEEE Transactions on Image Processing*, 9:846–859, 2000.

[12] Franck Jargstorff. A framework for image processing. In Randima Fernando, editor, *GPU Gems: Programming Techniques, Tips and Tricks for Real-Time Graphics*, pages 445–467. Addison Wesley, 2004.

[13] T. Kamei and M. Mizoguchi. Image filter design for fingerprint enhancement. *Computer Vision, International Symposium on*, 0:109, 1995.

[14] D. Maltoni, D. Maio, A. K. Jain, and S. Prabhakar. *Handbook of Fingerprint Recognition*. New York, 2003.

[15] Nvidia. Cuda presentation. `http://www.nvidia.com/object/what_is_cuda_new.html`, 2011.

[16] Nvidia. Direct compute. `http://www.nvidia.com/object/cuda_directcompute.html`, 2011.

[17] A. Ravishankar Rao. *A taxonomy for texture description and identification*. Springer-Verlag New York, Inc., New York, NY, USA, 1990.

[18] Robert Strzodka and Christoph Garbe. Real-time motion estimation and visualization on graphics cards. In *Proceedings of the conference on Visualization '04*, VIS '04, pages 545–552, Washington, DC, USA, 2004. IEEE Computer Society.

[19] Robert Strzodka and Alexandru Telea. Generalized Distance Transforms and skeletons in graphics hardware. In *Proceedings of EG/IEEE TCVG Symposium on Visualization (VisSym '04)*, pages 221–230, 2004.

[20] Thilaka Sumanaweera. Medical image reconstruction with the fft. In *GPU Gems 2: Programming Techniques for High-Performance Graphics and General-Purpose Computation (Gpu Gems)*. Addison Wesley, 2005.

[21] Manuel Werlberger, Thomas Pock, and Horst Bischof. Motion estimation with non-local total variation regularization. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, San Francisco, CA, USA, June 2010.

[22] Manuel Werlberger, Werner Trobin, Thomas Pock, Andreas Wedel, Daniel Cremers, and Horst Bischof. Anisotropic Huber-L1 optical flow. In *Proceedings of the British Machine Vision Conference (BMVC)*, London, UK, September 2009.