A Parallel PSO Algorithm for a Watermarking Application on a GPU

Edgar García Cano[1] and Katya Rodríguez[2]

[1] Posgrado en Ciencia e Ingeniería de la Computación,
Universidad Nacional Autónoma de México,
Mexico

[2] Instituto de Investigaciones en Matemáticas Aplicadas y en Sistemas,
Universidad Nacional Autónoma de México,
Mexico

eegkno@gmail.com, katya@uxdea4.iimas.unam.mx

Abstract. In this paper, a research about the usability, advantages and disadvantages of using Compute Unified Device Architecture (CUDA) is presented, implementing an algorithm based on populations called Particle Swarm Optimization (PSO) [5]. In order to test the performance of the proposed algorithm, a hide watermark image application is put into practice. The PSO is used to optimize the positions where a watermark has to be inserted. This application uses the insertion/extraction algorithm proposed by Shieh *et al*. [1]. This algorithm was implemented for both sequential and CUDA architectures. The fitness function—used in the optimization algorithm—has two objectives: fidelity and robustness. The measurement of fidelity and robustness is computed using Mean Squared Error (MSE) and Normalized Correlation (NC), respectively; these functions are evaluated using Pareto dominance.

Keywords. Parallel particle swarm optimization, watermarking, CUDA, image security.

Algoritmo paralelo PSO para una aplicación de marcas de agua en un GPU

Resumen. En este artículo se presenta una investigación de la usabilidad, ventajas y desventajas de usar Compute Unified Device Architecture (CUDA) implementando un algoritmo basado en poblaciones, Optimización por Cúmulo de Partículas (PSO) [5]. Para probar el rendimiento del algoritmo propuesto, se realizó la implementación de una aplicación de marcas de agua ocultas. El PSO es usado para optimizar las posiciones donde la marca de agua debe ser insertada. Esta aplicación usa el algortimo de inserción/extracción propuesto por Shieh *et al.* [1]. El algortimo completo fue implementado para las arquitecturas secuenciales y CUDA. La función de optimización —usada en el algoritmo de optimización— es la unión de dos objetivos: fidelidad y robustez. La medición de la fidelidad y robustez es procesada usando el Error Cuadrático Medio (MSE) y la Correlación de Normalización (NC) respectivamente; estas funciones son evaluadas usando dominancia de Pareto.

Palabras clave. Optimización por cúmulo de partículas en paralelo, marcas de agua, CUDA, seguridad en imágenes.

1 Introduction

The digital age introduced a new way to share information (files, audio, video, image, etc.), and there is no guarantee that someone else may use it without authorization, that is why watermarking appeared as an innovative way to protect information.

Digital watermarking is presented when a pattern is inserted in an image, video or audio file, it helps to copyright the information in the files. In the case of image watermarking, it is divided in two groups: visible and invisible watermarks.

A *visible* watermark is a visible semi-transparent text or image overlaid on the original image. It allows the original image to be viewed but still provides copyright protection by marking the image as its property. Visible watermarks are more robust against image transformation (especially if one uses a semi-transparent watermark placed over the whole image). Thus

they are preferable for strong copyright protection of intellectual property in digital format [2].

An *invisible* watermark is an embedded image that cannot be perceived with human eyes. Only electronic devices (or specialized software) can extract the hidden information to identify the copyright owner. Invisible watermarks are used to mark a specialized digital content (text, images or even audio content) to prove its authenticity [2].
Evolutionary computation, a subfield of Artificial Intelligence, uses models based on populations to solve optimization problems. Basically, these models were inspired by the mechanisms of natural evolution. Another set based on biological models and classified as bioinspired algorithms includes Ant Colony and Swarm-based algorithms. These are a different way to solve problems based on the behavior of animals or systems whose evolution lasts for centuries.

In recent years, new and cheaper technologies such as CUDA architecture have emerged with the concept of massive parallelism for general-purpose problems. The advantage of this technology is that every person with a personal computer has the possibility of taking advantage of the massive parallelism to accelerate procedures.

The process of watermarking can be applied to copyright any sort of digital information. In some fields like financial banking, it is necessary to perform a process involving a big quantity of information as soon as possible. On one hand, this is a reason to look for a new and cheaper technology such as CUDA to accelerate the process. On the other hand, the need to improve the watermarking process against modifications such as cropping, rotation, flipping, scaling, changing colors, etc., was the reason to use an optimization process. The idea of using PSO as an optimization algorithm comes forward owing to the fact that it has few parameters to adjust.

The rest of the paper is organized as follows. Section 2 offers an explanation of how the watermarking algorithm works and the metrics used to evaluate the watermarked image quality. In Section 3, an overview of the PSO algorithm is presented. Section 4 shows the optimization algorithm used in this work. In Section 5, tests and results are presented and analyzed. Finally, a discussion concerning advantages and disadvantages of using GPUs as a technology to implement algorithms based on populations is presented.

2 Methods

With a vast volume of information flowing on the Internet, watermarking is widely used to protect the information authenticity. The need to copyright a huge quantity of digital files spending the less possible amount of time and avoiding information loss were the reasons to propose the use of an algorithm for watermarking —Shieh algorithm—, Particle Swarm Optimization as an optimizer, and finally a GPU —based in CUDA architecture— to accelerate the process.

2.1 The Watermarking Algorithm

Shieh *et al*. [1] have proposed an algorithm to insert and extract watermark based on Discrete Cosine Transformation (DCT). This transformation is used due to the fact that it is not necessary to have the original cover to extract the watermark. Dealing with a huge number of images, it would be very expensive to store all cover images for the watermark extraction. Figure 1 shows an adjustment of the Shieh algorithm used in this work.

Once the original image is loaded in memory and after the DCT, the ratio values are calculated using DC and AC coefficients. Next step calculates the relation between the image content and the embedding frequency bands (polarities). Then, the watermark image is inserted in the selected bands of each 8x8 block. Quantization is used as an attack to the watermarked image, and it is necessary for the optimization process. Finally IDCT is computed and the watermarked image is obtained (see [3] for more details about the CUDA implementation).

2.1.1 Watermarking Metrics

In order to evaluate the performance of a watermark algorithm to hide the information, some metrics have been proposed. The watermark algorithm has to be capable to hide the mark data and to prevent distortions of the image. In order to propose a simpler way to measure the

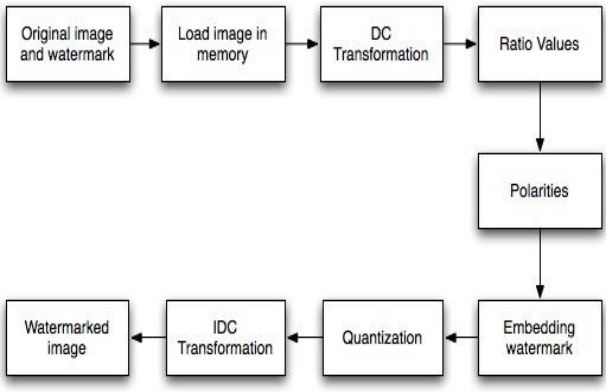

Fig. 1. Watermarking algorithm

fitness and the robustness spending the shortest time possible, the MSE and the NC were used.

– **Watermark Fidelity.** Fidelity represents the similarity of the watermarked image with the original image. Thus, mean squared error (see Equation 1) was utilized to measure fidelity. It ought to be close to zero to have a good correspondence between the non-watermarked and the watermarked image.

$$MSE(I,K) = \frac{1}{M}\frac{1}{N}\sum_{i=0}^{M-1}\sum_{j=0}^{N-1} \| I(i,j) - K(i,j) \|^2 \qquad (1)$$

– **Watermark Robustness.** Robustness represents the resistance of the watermark against attacks (compression, rotation, scaling, among others) done on the watermarked image. The normalized correlation NC (see Equation 2) is used to measure robustness. It applies the logical operation *exclusive disjunction*, also called *exclusive or*. Bitwise operations are faster therefore, reduce the runtime. The NC value must be close to zero between the original watermark (W) and the extracted watermark (W'), to prevent a watermark image information loss.

$$\mathrm{NC} = \frac{\sum_{i=1}^{M_w}\sum_{j=1}^{N_w}[W(i,j) \oplus W'(i,j)]}{\text{Bands per block}} \qquad (2)$$

Table 1. Exclusive OR

W	W'	Output
0	0	0
0	1	1
1	0	1
1	1	0

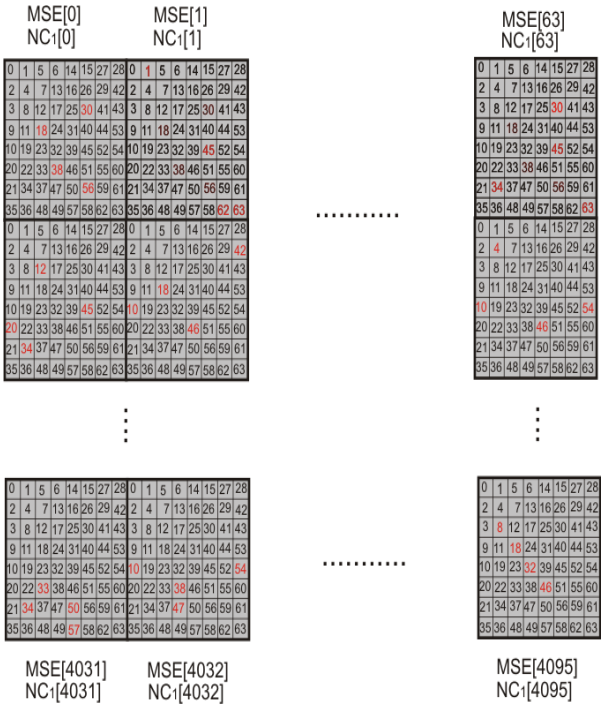

Fig. 2. Block organization to calculate MSE and NC

The *exclusive or* calculation is shown in Table1.

The NC and the MSE are computed for each 8x8 block as shown in Fig. 2. This was carried out with the purpose of dividing, as much as possible, the data in GPU. When measuring the MSE in each block, just 64 comparisons are necessary, they are executed at the "same time" in the other blocks. In the sequential process, 512x512 evaluations one after another are needed for a 512x512 image size. The same case was applied for the NC: instead of being calculated for the whole image (as in the sequential form), it was computed for each block.

2.2 PartiCle Swarm Optimization

Particle Swarm Optimization (PSO) is a population-based stochastic optimization technique developed by Eberhart and Kennedy in 1995 inspired by social behavior of bird flocking or fish schooling [10].

PSO shares many similarities with evolutionary computation techniques such as Genetic Algorithms (GA). The system is initialized with a population of random solutions and searches for optima by updating iterations. However, unlike GA, PSO has no evolution operators such as crossover and mutation. In PSO, potential solutions, called particles, fly through the problem space by following the current optimum particles. It has been successfully applied to many problems in several fields such as biomedicine [7] and energy conversion [4]. Image analysis is one of the most frequent applications and it is performed for biomedical images [9], microwave imaging [6, 8], among others.

2.2.1 Basic PSO Algorithm

Each particle keeps track of its coordinate in the problem space, which is associated with the best solution (fitness) achieved so far (this fitness value is stored). This value is called *pbest*. Another "best" value that is tracked by the particle swarm optimizer is the best value, obtained so far by any particle in the neighborhood of a given particle. This location is called *lbest*. When the particle takes all the population as its topological neighbors, the best value is a global best and is called *gbest*.

At each time step, the PSO concept consists of changing the velocity (acceleration) of each particle toward its *lbest* and *gbest* locations. Acceleration is weighted by a random term, with separate random numbers being generated for acceleration toward *lbest* and *gbest* locations.

After finding the two best values (*lbest* and *gbest*), the particle *i* updates its velocity and position with Equations 3 and 4, where *i = 1, 2, 3…NS*.

$$V_i(t+1) = V_i(t) + \phi_1 r_1\left(B_i(t) - X_i(t)\right) + \phi_2 r_2\left(B_i^g(t) - X_i(t)\right) \quad (3)$$

$$X_i(t+1) = X_i(t) + V_i(t+1) \quad (4)$$

ϕ_1 and ϕ_2 are positive constants called acceleration coefficients, *NS* is the total number of particles in the "swarm", r_1 and r_2 are random vectors, each component is generated within [0,1], and *g* represents the index of the best particle in the neighborhood. The other vectors $X_i=[x_1, x_2,\ldots, x_{iD}]$ ≡ position of the *i-th* particle; $V_i = [v_1, v_2,\ldots,v_{iD}]$ ≡ velocity of the *i-th* particle; B_i ≡ best historical value for the *i-th* particle found, B_i^g ≡ best value found for the *i-th* particle in the neighborhood [5]. For details about the CUDA implementation see [3].

Algorithm 1. Basic PSO Algorithm

```
1: Initialize particles population
2: while do not get the max number of iterations or
     the optimal solution do
3:         Calculate the fitness for each particle i
4:         Update Bi if pbest is better than the last one
5:         Calculate Big i of the neighbors
6:         for each particle i do
7:                 Calculate Vi according to 3
8:                 Update Xi  according to 4
9:                 Update best global solution (gbest)
10:        end for
11: end while
```

2.3 The Optimization Algorithm

The objective of optimization is to find the best frequency band set to insert the watermark within the image. Different frequency bands are tested through the iterations of the algorithm finding out the best solution. At the end of the execution the application has as results the watermarked image and a matrix with the whole best positions (frequency bands) to insert the complete watermark.

The algorithm in charge of doing the watermark optimization is the PSO, and at the same time it uses Pareto dominance to evaluate the fitness function through the MSE (fidelity) and NC (robustness). This process is detailed as follows.

1. Using the DCT idea to split the image in 8x8 blocks, each block is used as a swarm. An image of 512x512 has 4096 blocks; hence

each block will be a swarm. The number of particles by swarm is specified as a configuration parameter of the algorithm.

2. Each particle has a position vector. The vector size depends on the number of watermark bits used to be inserted in each block of the image. If the watermark size is 128x128 and if it is divided uniformly into 4096 blocks of the image, then 4 bits are inserted in each block. Each position corresponds to a band in the 8x8 block where the watermark bits are inserted.

At the beginning, all the swarms are initialized randomly (each swarm must have the same particle number). If 4 bits are to be inserted, 4 bands are required, and then 4 random numbers must be created between 1 and 63. This means that each particle will consist of 4 bands (position vectors).

If each swarm has 5 particles, every particle has a set of 4 bands used to originate 5 different solutions. To generate solution 1, all the particles with index 1 are taken from every swarm and joined; to generate solution 2; all the particles with index 2 are taken from every swarm and joined, and so on. This procedure is shown in Fig. 3.

3. After the insertion and extraction operations, the MSE (see Equation 1) and the NC (see Equation 2) are calculated. Based on MSE and NC, the fitness value is estimated.
4. One of the particles must be selected as the best global. Among the best options generated, one of them is chosen to be the best global. To choose the local best, the particle is considered to add up the MSE and the NC. If the new value is closer to zero than the old one, the new particle replaces the old one; otherwise the old one continues in the process.
5. In the last step, the velocity and the new position of the particles are calculated according to Equations 3 and 4. This generates the new bands and new iteration begins. Fig. 4 shows the whole algorithm.

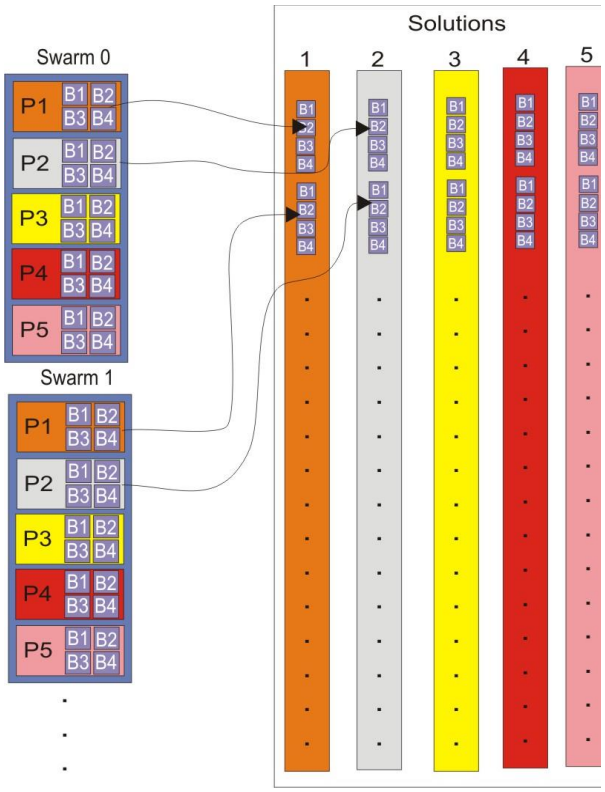

Fig. 3. Generation of solutions taking particles P1 and P2 from the different swarms, bands B1, B2, B3 and B4 generate the corresponding solution

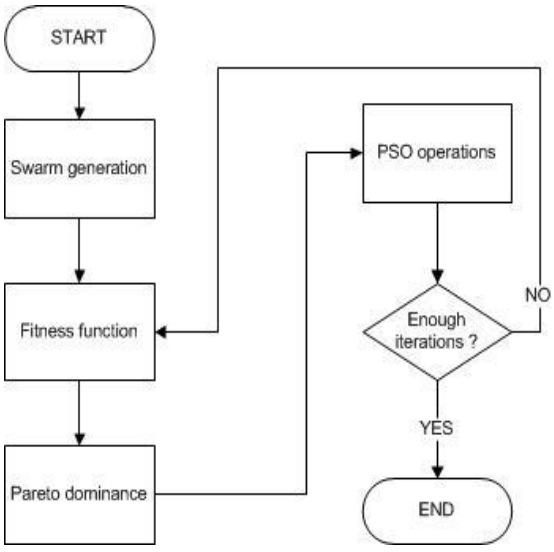

Fig. 4. The optimization algorithm

3 Test and Results

All tests were executed on two different servers with the features shown in Tables 2 and 3.

In order to test the implementations, Fig. 5 shows the original image used in the algorithm and Fig. 6 shows the watermark image. For the experiments, the size of the images is 515x512 and they are in gray scale.

3.1 Results of the Algorithm

Tables 4 and 5 show the runtime of the watermarking optimization. Parts A and B show five experiments with different number of iterations using the sequential and CUDA versions. These experiments were performed to compare the amount of time used for the algorithm and the quality of the results based on the idea that the operations executed in the GPU must be faster than the ones computed in the CPU.

Part C of Tables 4 and 5 show the runtime for the sequential implementation with 10 and 30 iterations, but without random number generation. This was made to compare CUDA runtime and check if CUDA implementation is faster than the sequential implementation without random numbers generation.

Fig. 5. Original image (Barbara)

Fig. 6. Watermark image (© 2012 BancTec, Inc., All rights reserved)

Table 2. CPU server features

Server name	Cores	CPU type
Uxdea	8	Intel Xeon E5620 @ 2.4GHz
Geogpus	8	Intel Xeon E5677 @ 3.47GHz

Table 3. GPU server features

Server name	Cores	GPU
Uxdea	240	Tesla C1060
Geogpus	240	Tesla C1060

Using random numbers in the sequential version produced a remarkable difference in time. The use of those numbers consumes a big quantity of time due to its necessity to spend time in the CPU to generate different numbers. For the sequential version, random numbers are generated using the C function drand48 that returns a pseudo-random number in the range [0.0,1.0). On the GPU, random numbers are generated using a library called *curand*.

In the case of the GPU (Part D of Tables 4 and 5), random numbers are generated directly in the constant GPU memory; there is no need to transfer them from the host to the device. This is why the difference, in terms of runtime, between the option with random numbers and the option without random numbers in CUDA is minimal.

Reviewing the values (Tables 4 and 5) of the initial fitness and the final fitness, it is noteworthy that the sequential version gives better results than the ones obtained from the GPU. For all the cases, the runtimes indicate that GPU is faster than CPU, even when all data have been loaded or when static numbers in the CPU version are used. Thus, it is possible to set up that, at least for this version of the application, if the user wants a good optimization for the watermarking, the sequential version must be used. However, if the user needs a quick approximation, the GPU version must be applied (for other results see [3]).

Table 4. Runtime for PSO on Geogpus

10 Iterations

A	Sequential (min)	Initial fitness	Final fitness	CUDA (s)	Initial fitness	Final fitness
1	53.7133	3.0079	0.24625	6.8422	5.3136	4.9884
2	53.6950	5.7397	0.27473	6.6205	0.66101	0.33551
3	53.6600	2.9303	0.25068	6.6103	0.82102	0.50254
4	53.6517	3.5922	0.27855	6.6197	15.315	14.716
5	53.6433	4.2929	0.2387	6.6786	1.9163	1.5937
	53.6727			**6.6743**		

30 Iterations

B	Sequential (min)	Initial fitness	Final fitness	CUDA (s)	Initial fitness	Final fitness
1	161.35	2.4226	0.21096	18.126	1.7282	1.324
2	160.6917	3.4672	0.2026	18.516	0.78431	0.39118
3	160.635	5.8846	0.23375	18.081	0.81376	0.40794
4	161.075	3.1447	0.19965	18.397	1.6347	1.2302
5	161.35	3.3854	0.19588	18.113	1.4422	1.0504
	161.0203			**18.2466**		

Sequential no random numbers

C	10 Iterations	Initial fitness	Final fitness	30 Iterations	Initial fitness	Final fitness
1	26.228	4.7199	0.22448	72.8140	26.167	0.223
2	26.108	5.1654	0.29945	72.6400	6.8238	0.23352
3	26.208	2.4494	0.2621	72.8590	7.7055	0.21568
4	26.279	10.571	0.26432	72.7310	1.9816	0.20543
5	26.167	6.073	0.20027	73.0250	2.9215	2.9215
	26.1980			**18.2466**		

CUDA no random numbers

D	10 Iterations	Initial fitness	Final fitness	30 Iterations	Initial fitness	Final fitness
1	6.5794	2.8593	2.5458	17.7430	1.1509	0.75917
2	6.5145	1.2957	0.97485	17.9710	7.8518	7.215
3	6.6214	0.82386	0.49934	17.8740	0.85486	0.47526
4	6.6271	5.2253	4.8637	17.6850	5.6334	5.2352
5	9.8096	2.321	1.996	17.0650	0.49091	0.1239
	7.2304			**17.6676**		

Table 5. Runtime for PSO on Uxdea

10 Iterations

A	Sequential (min)	Initial fitness	Final fitness	CUDA (s)	Initial fitness	Final fitness
1	75.0933	4.7145	0.288	17.696	2.429	2.1085
2	75.1300	11.316	0.3058	17.754	0.9868	0.64933
3	75.1467	5.4008	0.254	17.744	0.85646	0.53621
4	75.1433	3.1818	0.2784	17.626	2.1421	1.8232
5	75.0767	3.1201	0.2568	17.696	3.9049	3.6015
	75.1180			**17.7032**		

30 Iterations

B	Sequential (min)	Initial fitness	Final fitness	CUDA (s)	Initial fitness	Final fitness
1	225.3833	3.8861	0.22232	48.535	0.70031	0.30285
2	225.1	4.0846	0.20906	48.635	1.4996	1.1144
3	225.2667	2.5326	0.20198	48.794	2.0107	1.6072
4	224.8833	3.3661	0.2154	48.597	6.3867	5.4775
5	225.3833	4.4556	0.21926	48.757	0.67997	0.28269
	225.2033			**48.6636**		

Sequential no random numbers

C	10 Iterations	Initial fitness	Final fitness	30 Iterations	Initial fitness	Final fitness
1	36.363	6.3162	0.24992	100.54	3.3363	0.21798
2	36.664	3.0594	0.2729	100.64	2.3531	0.22367
3	36.289	3.2031	0.26768	100.5	8.4095	0.20502
4	36.302	2.9081	0.25941	100.59	3.1055	0.20916
5	36.404	3.902	0.32894	100.51	5.426	0.22008
	36.4044			**100.556**		

CUDA no random numbers

D	10 Iterations	Initial fitness	Final fitness	30 Iterations	Initial fitness	Final fitness
1	17.347	1.041	0.72842	47.8740	0.66338	0.26873
2	17.395	1.2237	0.90721	47.9710	7.859	7.255
3	17.59	5.0348	4.1984	47.9920	2.8143	2.4339
4	17.462	5.149	4.8263	47.8480	4.5469	4.1704
5	17.577	1.1466	0.8309	48.1320	0.8529	0.46197
	17.4742			**47.9634**		

4 Conclusions

Since there is no standard configuration for the blocks, threads or memory treatment in the GPU, it is necessary to make analysis and design of the procedures involved in a particular application to take advantage of parallelism. In order to use parallel programing in a GPU, it is necessary to shift from sequential to parallel thinking and to learn how to divide a huge problem into small ones (divide and conquer), attempting to obtain the best performance. For example, in the calculation of the NC, only 4 threads were required to perform comparison, but in the case of the MSE 64, threads working at the "same time" were used. Therefore, the configuration of the blocks and threads for an application on a GPU must be carefully analyzed.

Different options to implement the PSO were analyzed, but the version that uses as much swarms as the number of blocks to divide an image in the DCT was implemented. The idea was to divide a big problem into smaller ones, which suited the parallel paradigm. As it has been established, there is no standard configuration in CUDA architecture, so the accordance between configuration and the need of the function was achieved. The PSO has to evaluate two vectors: velocity and position. Position depends on velocity that is why velocity is to be computed first. If there are 4096 swarms—4096 blocks—and each swarm has five particles, then each of them needs to update the velocity vector. The number of operations to be calculated in a CPU is 4096 (swarms) * 5 (particles) * 1 (operation) = 20480 operations one after another. In the case of the same operations on the GPU, the same 20480 operations are executed, but the difference is that there are 4096 swarms with 5 threads working in parallel computing one operation, hence there are 20480 threads working at the same time. If one thread in the CPU spends 1 second per operation, the runtime will be 20480 s, but in the case of the GPU there are 20480 threads working at the same time, and they spend 1 second to finish the calculus. In the last example, the speed of the processor is not considered, neither CPU nor GPU, nor the upload/download of the data to/from the GPU.

The velocity vector needs random numbers to be calculated (see Equation 1). In order to generate random numbers, a library called *curand* was used. This library is useful because it is easy to generate a lot of numbers in a short time; the problem comes with the memory. If there is a big quantity of these numbers generated and held in global memory, there might be a shortage of space to store other data. In one iteration of the PSO, two random numbers are used to calculate the velocity value. If there are 4096 blocks with 5 particles each, 40960 random numbers for iteration are needed. There is another type of memory on the GPU, the constant memory. This memory is loaded in the GPU but it cannot be changed. This memory was considered to store the random numbers because they do not modify its value on the execution of the calculation of the velocity value.

Another feature that has to be considered (from GPU to GPU) is the processor velocity. This is evident in the experiments because the Geogpus server is faster than the Uxdea server.

The analysis done in the present work shows that the use of CUDA helps to improve the performance of the application and that an algorithm based on population can be implemented in it, as long as the developer is aware of the features of this technology.

5 Future Work

The first phase of this project is focused on the implementation with the elemental CUDA features of a basic algorithm based on population; however, the CUDA architecture may be better exploited. The second phase will be focused on a combination of other CUDA features, such as streams, page locked memory and mapped memory. The aim of the third phase is to improve operations on the CPU and MPI to distribute tasks (in this case different swarms) in a cluster by mixing other technologies, such as Open MP.

Acknowledgements

The first author, postgraduate student in Computer Science and Engineering at the

National University of Mexico, expresses his gratitude to the support received from CONACYT (scholarship number 37617). The authors also express their gratitude to the support received from PAPIIT (project number 109011).

References

1. **Chin-Shiuh, S., Hsiang-Cheh, H., Feng-Hsing, W., & Jeng-Shyang, P. (2004).** Genetic watermarking based on transform-domain techniques. *Pattern Recognition,* 37(3), 555–565.
2. ByteScout (s.f.). Retrieved from http://bytescout.com/.
3. **García, E.E. (2012).** A parallel bioinspired watermarking algorithm on a GPU. MSc thesis, Universidad Nacional Autónoma de México, México, D.F.
4. **Heo, J.S., Lee, K.Y., & Garduno-Ramirez, R. (2006).** Multiobjective control of power plants using particle swarm optimization techniques. *IEEE Transactions on Energy Conversion*, 21(2), 552–561.
5. **Mohemmed, A., Johnston, M., & Zhang, M. (2009).** Particle swarm optimization based multi-prototype ensembles. *11th Annual conference on Genetic and evolutionary computation (GECCO'09),* Montreal, Canada, 57–64.
6. **Donelli, M. & Massa, A. (2005).** Computational approach based on a particle swarm optimizer for microwave imaging of two dimensional dielectric scatterers. *IEEE Transactions on Microwave Theory and Techniques*, 53(5), 1761–1776.
7. **Selvan, S.E., Xavier, C.C., Karssemeijer, N., Sequeira, J., Cherian, R.A., & Dhala, B.Y. (2006).** Parameter estimation in stochastic mammogram model by heuristic optimization technique. *IEEE Transactions on Information Technology in Biomedicine*, 10(4), 685–695.
8. **Huang, T. & Mohan, A.S. (2007).** A microparticle swarm optimizer for the reconstruction of microwave images. *IEEE Transactions on Antennas and Propagation*, 55(3), 568–576.
9. **Wachowiak, M.P., Smolikova, R., Yufeng, Z., Zurada, J.M., & Elmaghraby, A.S. (2004).** An approach to multimodal biomedical image registration utilizing particle swarm optimization. *IEEE Transactions on Evolutionary Computation,* 8(3), 289–301.
10. **Kennedy, J. & Eberhart, R. (1995).** Particle swarm optimization. *IEEE International Conference on Neural Networks*, Perth, Australia, 4, 1942–1948.

Edgar García Cano received his Bachelor degree in Computer Engineering, and Master's degree in Computer Science from the National University of Mexico (UNAM) , specializing in parallel programing and evolutionary computing. He worked as a teacher and IT consultant in the Engineering Faculty at UNAM for eight years. Currently, he is completing his Doctoral studies at the École de technologie superieure in Montreal, Canada.

Katya Rodríguez received her Bachelor degree in Computer Engineering from the National University of Mexico in 1994. She obtained her Ph.D. from the University of Sheffield in 1999. She is currently a researcher at the Institute of Applied Mathematics and Systems Engineering, National University of Mexico. She has published a number of papers in international journals and conferences and has been a member of the Technical Program Committee for Conferences related to Evolutionary Computation. Her research interests are evolutionary and bio-inspired algorithms, multi-objective optimization and its applications in diverse fields.

Article received on 20/01/2013; accepted on 11/08/2013.