Ain Shams Engineering Journal xxx (2016) xxx–xxx

Contents lists available at ScienceDirect

Ain Shams Engineering Journal

journal homepage: www.sciencedirect.com

Electrical Engineering

An accelerated shape based segmentation approach adopting the pattern search optimizer

Ahmed H. Yousef *, Hossam E. Abdelmunim

Computers and Systems Engineering Department, Ain Shams University, Egypt

ARTICLE INFO

Article history:
Received 3 April 2016
Revised 14 October 2016
Accepted 2 November 2016
Available online xxxx

Keywords:
Shape based segmentation
GPU
CUDA
Global optimization
Pattern search
Genetic algorithms

ABSTRACT

All known solutions of the shape based segmentation problem are slower than real-time application requirements. In this paper, the problem is formulated as a global optimization problem for an energy objective function with several constraints. This formulation allows the use of the global optimization solvers as a solution. However, this solution will be slow as it requires the evaluation of the objective function for several thousand times. The objective function computation is one of the critical factors that affect the time needed to reach a solution. The authors implemented two accelerated parallel versions of the solution that integrates the objective function and the pattern search solver. The first uses a GPU accelerated implementation of the objective function and the second uses a CPU parallel version which is executed on several processors/cores. The results of the proposed solution show that the GPU version has substantial speed compared to other approaches.

1. Introduction

One of the most difficult challenges in the field of shape based segmentation problem is that high computational power is necessary to reach fast accurate performance. The real-time processing of image shape based segmentation is difficult to attain without the use of parallel computing. High-resolution cameras and increasing accuracy requirements challenge the real-time performance realization with the best ordinary modern CPU [1].

The shape based segmentation problem uses an energy objective function that contains multiple minima and several constrains that should be satisfied [2,3]. The constraints of the objective function are non-smooth [4–6]. The problem is characterized by having a linear equality constraint and lower and upper bounds for the parameters. This makes the optimization problem suitable to be solved by both the pattern search and the genetic algorithm optimization [7–12]. However, up to the authors' knowledge, this was not evaluated or covered in literature.

In the same time, the parallel systems that use CPUs with multiple processors and cores or GPU are continuously evolving into a powerful and cheap computation resource [1,13,14]. For example, Matlab® has the parallel computation toolbox to support both variants. Moreover, GPUs support several high-level languages including C.

Because computer vision and image processing algorithms usually perform the same calculations on each pixel of the image, which is a typical single instruction multiple data (SIMD) scenario, these calculations can be parallelized on CPUs with multiple cores and GPUs [15–19]. Shen et al. implemented processing of MPEG video encoding [13]. GpuCV [15], MinGPU [16] and OpenVIDIA [17] are examples of libraries that are created to implement computer vision on the GPU platform. In [14], the non-rigid registration is solved. Other attempts to speed up similar applications include segmentation and integral image computation [18–20], medical image registration [21] and high performance Iris recognition systems [22].

The GPU is used in several scientific applications to speed up the slow operations [23]. Zhu attempted to speed up the classical local pattern search using the GPU for several simple objective functions including Step, Sphere, Ackley, Rastrigin and Rosenbrock [24,25]. Computational results on the global optimization toolbox [26] indicate that the GPU-accelerated method is orders of magnitude faster than the corresponding CPU implementation. However, this contribution did not take into consideration that the objective functions in computer vision algorithms are very slow compared to both the search solver execution itself and to the simple benchmark objective functions used in [24,25]. On the other hand, this paper focus on the situation when the objective function is very slow and needs substantial acceleration by executing it in parallel.

Peer review under responsibility of Ain Shams University.

* Corresponding author.
E-mail address: ahassan@eng.asu.edu.eg (A.H. Yousef).

http://dx.doi.org/10.1016/j.asej.2016.11.002
2090-4479/

Solving the shape based segmentation problem involve many mathematical and logical operations for each pixel of the images. This makes the GPU very suitable for the GPU provides a massive data parallel processing capability that facilitates reasonable acceleration. However, porting the shape based segmentation to GPU with a reasonable speedup is not a trivial task. The proposed parallel target algorithm is designed carefully in a way that considers the hardware limitations of the GPUs to achieve a reasonable performance gain against ordinary CPUs.

We selected NVIDIA CUDA framework to be used in this paper because it enables the programmer to control the parallelism by specifying the grid size and number of threads on the block. In addition, the programmer can define which data will be stored in the global memory space and which data should be used in the faster shared memory to be used simultaneously by parallel threads.

The contributions of this paper are three fold: First, a parallel solution for an energy objective function is proposed using CUDA programming model to accelerate the execution time. Second, the standard Matlab® pattern search solver is integrated with the accelerated objective function to solve the problem. The results are compared to the genetic algorithm solver using the accuracy and execution time measurable metrics to numerically evaluate the effectiveness of the integrated solution. Then, the time execution results of the integrated solution are compared with another parallel implementation on different processors/cores/workers.

The remainder of this paper is organized as follows: Section 2 provides the necessary background including a description of the shape based segmentation problem, global optimization, and GPU/CPU parallel platform setup. Section 3 addresses the proposed methodology including the performance criteria and problem constrains, the detailed implementation, the environment of CPU and CUDA, the benchmarking and optimization tools, the description of the proposed parallel solution and its detailed GPU mapping. Experimental results and validation are presented and discussed thoroughly in Section 4. Finally, Section 5 concludes this paper.

2. Background

2.1. Shape based segmentation problem

Shape based segmentation is one of the most difficult tasks under the umbrella of computer vision. The target is to mark boundaries of objects of interest. Image noise, in-homogeneities, lack of strong edges, and occlusion represent the challenges. Unfortunately, there is no crisp definition for object boundaries. Shape based segmentation is considered one of the preliminary tasks for computer graphics, visualization, and diagnostics in medical imaging, military applications and object character recognition.

When large in-homogeneities and occlusions exist, most approaches fail in marking the object boundaries correctly. This includes graph theoretic, region-based techniques, clustering, edge, and pixel approaches. This raise the need of more sophisticated techniques by adding prior knowledge about object shapes. Prior knowledge of objects includes the necessary information about shape boundaries as well as intensity variations. Level sets are used to meet these requirements due to their smoothness properties and capability to fit to ill-defined boundaries [27]. Level sets are preferred over deformable model due to its flexibility of merging and splitting without the need of any parameters. In addition, its implementation in 2D and 3D is straightforward.

Gathering prior knowledge about object boundaries is done by representing shape variations implicitly by level set functions after removing differences between the training curves/surfaces. In many cases, a prior shape/boundary model is a weighted sum of implicitly represented shapes. Such a model is embedded within the image domain by registration. Image intensity information is modeled by another implicit function. Successfully registering the two implicit functions will result in marking the object boundaries accurately. Registration of implicitly represented shapes is formulated as a minimization of sum of squared differences of the two implicit functions. In this case, the energy function is not convex and has a big number of parameters. Therefore, a smart optimization technique is required.

2.1.1. Shape representation [27,28]

A planar smooth curve can be defined as $\mathbf{C}(p)$: $R \rightarrow R^2$ with a parameter $p \in [0,1]$. The point vector is defined as $\mathbf{C}(p) = [x(p)\ y(p)]^T$ where $x \in [0, X_{\max}]$ and $y \in [0, Y_{\max}]$. This is an explicit representation of the given contour/shape **C**. An implicit shape representation will be demonstrated as follows:

Given a smooth curve, **C** (defined above), an implicit function can be defined by

$$\Phi(X) : \Omega \in R^2 \rightarrow R \text{ where }, \Phi(X) = \|X_0 - X\|, \ X \text{ and } X_0 \in \Omega, \quad (1)$$

The closest point on the shape **C** to **X** is $\mathbf{X_0}$. The shape/boundary points always satisfy the condition $|\Phi(\mathbf{C})| = 0$.

2.1.2. Global registration of shapes

The global registration process is formulated as an objective function minimization as below:

$$E = \int_{\Omega} \delta'(\phi_\alpha, \phi_\beta)(s\phi_\alpha(X) - \phi_\beta(A))^2 d\Omega \quad (2)$$

A transformation with scales, rotation, and translation is defined by the matrix, **A**. This transformation changes the source shape α to match a target β. The parameter s is defined as the maximum of the axial scaling parameters (s_x, s_y). The function (δ') is used to narrow the matching space between the two domains. The above functional describes the registration in a better way since it incorporates a scaled version of the source shape representation. It is worth mentioning here that the involved functions are not differentiable at the line ($s_x = s_y$). The parameters {s_x, s_y, θ, T_x, T_y} are required to minimize the energy functional E.

2.1.3. Model-based boundary detection

A boundary model is represented implicitly by the function $\phi_p = \sum_{i=1}^{N} \omega_i \phi_i$ where N is the number of training boundaries, ϕ_i is the implicit representation of the boundary i, and $\omega_i \in R^+$. This representation is a weighted-sum of the implicit representations of the training shapes. Intensity-based segmentation of the region of interest is represented implicitly by the function ϕ_g [2,27]. The parameters vector, $U = [s_x, s_y, \theta, T_x, T_y, \omega_1, \ldots, \omega_N]^T$ is obtained by minimizing the following objective function (defined as above):

$$E(U) = \int_{\Omega} \delta'(\phi_p, \phi_g)(s\phi_p(X) - \phi_g(A))^2 d\Omega \quad (3)$$

where **A** is defined as above [27]. The minimization problem can be written as

$$\hat{U} = \arg\min_{U} E \quad (4)$$

This optimization problem is subject to the following constrains: (1) Scales are positive real numbers. (2) Weights are positive real numbers. (3) Sum of weights is unity or $\sum_{i=1}^{N} \omega_i = 1$. (4) Also, each parameter value takes a value between a lower and upper bounds. Such an optimization problem is a challenging one because: (1) Convexity of such a function is not guaranteed. (2) The objective function is not proved to be smooth. In addition, it is not differentiable at all points. These issues limit the use of some optimization techniques. (3) The number of parameters that the

Please cite this article in press as: Yousef AH, Abdelmunim HE. An accelerated shape based segmentation approach adopting the pattern search optimizer. Ain Shams Eng J (2016), http://dx.doi.org/10.1016/j.asej.2016.11.002

function takes is large (N + 5). This makes the visualization of the function surface impossible and prevent checking its characteristics for optimization.

2.2. Global optimization algorithms [26]

Finding the minimum value of the objective function under certain constrains is a global optimization problem. This problem is solved by global optimization solvers that iterate to find the solution. Both Pattern Search and Genetic Algorithm proved their abilities to be effective solvers and handle all types of constrains for a wide range of problems. Direct Search is characterized by their provable ability to converge using deterministic iterates approach. Genetic Algorithms use random numbers and have stochastic iterates and usually converge in a reasonable amount of time. The algorithm of the pattern search can be found in [8,12,25] while the Genetic algorithm is well described in [5,7].

2.3. GPU Architecture and problem characteristics

A GPU contains several multiprocessor cores. These cores can execute the same code concurrently using tens or hundreds of threads. A parallel function that is executed many times on the image pixels, is executed on the device in parallel and called a kernel. Each processor of the multiprocessor works on different data simultaneously. The device maintains its own global memory. Multiprocessors also have different levels of faster memory including: registers (32 bits each, total 64 KB), shared memory (48–64 KB), constant cache, and texture cache.

Shape-based segmentation algorithms involve independent processing of a large pixel set which can benefit from the parallel execution of the GPU. In addition, shape based registration algorithms use several model image files which require large memory to store pixel data. Because the access of such files is often regular and sequential, it suits very well the GPU model. For a GPU device, the decision of using shared memory is important to minimize the latency problem and to achieve reasonable memory bandwidth.

The shape based registration code using pattern search or genetic algorithm is characterized by having some code that can be converted to parallel, for example calculating the fitness function in the genetic algorithm and computing the objective function for the mesh points in the pattern search algorithm. However, there are other parts of both algorithms that are sequential. For example, the selection process in the genetic algorithm and the mesh sizing in the pattern search.

3. Methodology

In this paper, we formulated the model based object boundaries marking as an optimization problem. The objective function calculation is accelerated by implementing it on GPU using CUDA. Then, it is integrated with the standard pattern search solver. Then the results of the proposed solution are compared to the results obtained from genetic algorithm optimization technique. Then, a comparison is performed with the results obtained from the parallel implementation on several number of CPU processors/cores by the Matlab® parallel computation toolbox. Several exhaustive experimentation and validations are used to illustrate the object boundary extraction of different images. In other words, this paper compares the results of the Matlab® based sequential implementation, the proposed combined Matlab-CUDA-GPU parallel implementation of the pattern search and genetic algorithms and Matlab® based CPU parallel implementation.

The CPU parallel version of the proposed solution used the standard Matlab® parallelized version of the Pattern Search and Genetic solvers since the standard Matlab® Parallel Computing Toolbox™ support the code execution on multicore cores/processors.

We selected both Direct Pattern Search and Genetic Algorithms to solve the problem because they do not use derivatives, do not require the objective function to be smooth in addition to their effectiveness and efficiency. In the same time, their Matlab implementation support vectorized functions [26]. This means that they can optionally compute the objective functions of a collection of vectors in one function call, causing the execution time to be much less than in the case of serial computation [26].

Other metaheuristics based techniques including Particle Swarm Optimization (PSO), Differential Evolution (DE), Harmony search (HS) do not support vectorization implementation which prevent them from being used with parallel Matlab® execution on CPU. Other techniques like Simulated Annealing and differential evolution is found to be very slow in GPU [29].

The objective function is rewritten to accept a matrix in order to benefit from the acceleration resulted from converting the objective function to parallel implementation. This computation the objective function in vectorized fashion [26] allows the use of a GPU with large number of processors (tens, hundreds or thousands) in parallel. It is expected that this will be much faster than other SIMD streaming extension vectorization techniques that uses 128 bit registers to execute code for four integers simultaneously instead of one [29]. The constraints are implemented with the global optimization toolbox as a penalty function whose value is added to the objective function.

We used a dataset that includes twelve model images of size 128×128 pixels. The target image size is 512×512. The size of the model images and target images are selected to be different to ensure generality. We selected smaller size for model images because it is known that GPU is scaled well for large images [1].

In order to ensure fair comparison between algorithms, several precautions are used. For example, the test of each solver is composed of twenty independent runs. For each run, the following results are reported: the obtained minimum value of the objective function (as a measure of accuracy), the best parameter set values (translation, scale, rotation and weights). We used several runs because the execution time varies from a run to another according to other processes working by the operating system. In addition, the accuracy vary from a run to another because of the stochastic nature of the algorithms. Therefore, we reported the execution time and accuracy for several runs and reported the average, minimum and maximum values.

Although we can start all the solvers with the same set of random points as well, we used different sets of random points for each run to ensure that similar final results are achieved even with the stochastic nature of solvers. Other precautions include using the same values of 10^{-6} are used for the tolerance in function and the tolerance in parameter value for both the Genetic Algorithm and Pattern Search.

Genetic Algorithm is customized to have the initial population around an initial point. In this paper, Genetic algorithm is customized by modifying initial population size to cover multiple basins. Other customizations that can be tackled in the future include changing fitness functions, scaling options or defining new parent selection, crossover, and mutation functions. Pattern search can be customized in the future by defining polling, searching, and other functions. In order to have an accurate comparison, we used the default settings for both pattern search and genetic algorithms solvers [7–9,26].

Please cite this article in press as: Yousef AH, Abdelmunim HE. An accelerated shape based segmentation approach adopting the pattern search optimizer. Ain Shams Eng J (2016), http://dx.doi.org/10.1016/j.asej.2016.11.002

3.1. Performance criteria and problem constrains

Although the Number of function evaluations (NFEs) is an important criterion to compare different algorithms for obtaining a value to reach, we consider the execution time as a fairer criterion that takes vectorization and multiple core support and overhead into consideration while NFEs do not reflect these factors.

In the shape based segmentation process, an approximate answer could be fine unless we have the resources to exhaust a result. The problem is considered an "online" problem that must be solved many times within a short time frame. Therefore, the necessary CPU time matters a lot.

For each test, the number of function execution is reported along with the minimum, average, maximum and standard deviation of the execution time and the objective function minimum values. These measures are used to compare time, accuracy and reliability of the solvers.

The following constraints are applied: (1) The sum of weights of different images similarity equals one. (2) Upper bound and lower bounds are applied for all parameters.

3.2. Detailed implementation

To simplify the implementation, we used the standard Matlab® implementation of the genetic algorithm solver and the pattern search solver to find the best value of the weights, translation, rotation and scaling parameters that minimizes the objective function. The GA is configured with the default value of 0.8 for Crossover fraction and with decreasing Gaussian default mutation option.

The objective function is implemented in CUDA. The C-MEX is used as an interface between the solvers Matlab® code and the objective function CUDA code. The C-MEX function is responsible from converting the column-major order to row-major order matrices from Matlab to C and vice versa.

3.3. Environment and used tools

In order to compare the different solver algorithms, Matlab® programs are written for the solver part and CUDA is used to implement the objective function. The programs are executed on a machine with the following configuration and CPU Specifications: The used processor is an Intel(R) Core(TM) i7-4702MQ CPU @ 2.20 GHz. The used processor has 4 cores and 8 threads. The memory was 8 GB RAM. The used Graphics card is NVIDIA GeForce GT 720 M, with the following GPU specifications: CUDA Compute Capability: '2.1', two symmetric multiprocessors, 96 CUDA Cores. The GPU can handle up to 1024 simultaneous threads per block with maximum shared memory of 48 KB per block. The Total Memory is 2 GB.

In order to test the scalability of the solution on larger GPUs with larger number of cores, we used a K20 GPU based machine with 2496 cores on an Intel Xeon E5-2609 2.5 GHz host processor.

3.4. Objective function implementation on the GPU

The target image input and the models images are stored in the global memory because their size was very large (512×512 pixels of floating point numbers for the target input image and 128×128 pixels for the 12 model images, respectively). This means they cannot fit in either the constant memory or texture memory. Preliminary tests showed putting one of the images only in the constant memory will not improve the speed significantly. Other investigated option was the compression of the images by quantizing the levels of pixels but this option degraded the accuracy significantly.

For each parameter set instance (a row of 17 parameters: two for scaling, two for translation, one for rotation and twelve for weights), there is a need to transfer model and target images and parameter set instance from the CPU to GPU (HostToDevice) and to transfer the results back from the GPU to CPU (DeviceToHost). The execution time and performance of the objective are affected by the memory transfer overhead. Therefore, our solution proposed the use of a parallel implementation of the vectorized function on the CUDA level by sending several parameter set instances at the same time. The input image and the model images are passed from the host memory to the GPU device memory only once for several parameter set instances. Then the computation of several sets of parameter set instances ($nRows * 17$ parameters) is executed in parallel to generate the nRows results, as shown in Fig. 1. This maximizes the number of computations per this large memory transfer.

Because the resources of the GPU device used was limited. We define a chunk size of 1024 which is the maximum number of parameter set instances that are computed in parallel. If the number of parameter set instances exceeds this limit, it will be sent in several chunks sequentially from the host code to the GPU and each chunk will be executed in parallel.

The GPU parallel implementation consists of three kernel function. The first one is designed to have the same number of threads as the number of pixels and perform the average shape calculation and the energy function computation in a per-pixel/per-thread manner. The first kernel is designed to use global memory coalescing in order to conserve bandwidth, while reducing effective latency. In the first kernel, shared memory is used to allow 16×16 blocks to get a tile from the input image. Also, another shared memory with size 16×16 is used to store the results of calculating the energy function per block. The use of shared memory is much faster than the use of the global memory. This is shown in the right part of Fig. 2. We used a thread block of (16, 16, 2) and a grid size of (8, 8, NumberOfRows/2). Using the CUDA Occupancy Calculator, this is equivalent to 67% Occupancy of each Multiprocessor.

The second kernel uses the well-known resource efficient list reduction algorithm in order to calculate the sum of the several pixels energy function to have the partial sum per block. This is shown in the left portion of Fig. 2.

The reduction algorithm is selected to minimize branch diversity. Each thread is used to calculate the sum of the energy function of two pixels. Then half of the threads are used to compute the sum of four pixels and the loop continues till the entire block sum is calculated. This kernel is characterized by data dependency. Parallelism among threads in a thread block is serialized when some threads need to synchronize to share data between each other through memory access [1]. A third kernel is used to calculate the gross sum for each parameter set instance. It uses the same technique used with the second kernel but for one dimensional array instead of two dimensional one.

4. Result and discussion

4.1. Using vectorization and CUDA on the objective function

Matlab, C and CUDA are implemented for both the vectorized and non vectorized version of the objective function. Both Matlab and C are implemented sequentially and CUDA is implemented on a parallel GPU. They are compared for different number of parameter set instances. Table 1 shows the average time per parameter set instance for different implementations.

If we compare each column that represent the vectorized version and the corresponding column that represent the non

Please cite this article in press as: Yousef AH, Abdelmunim HE. An accelerated shape based segmentation approach adopting the pattern search optimizer. Ain Shams Eng J (2016), http://dx.doi.org/10.1016/j.asej.2016.11.002

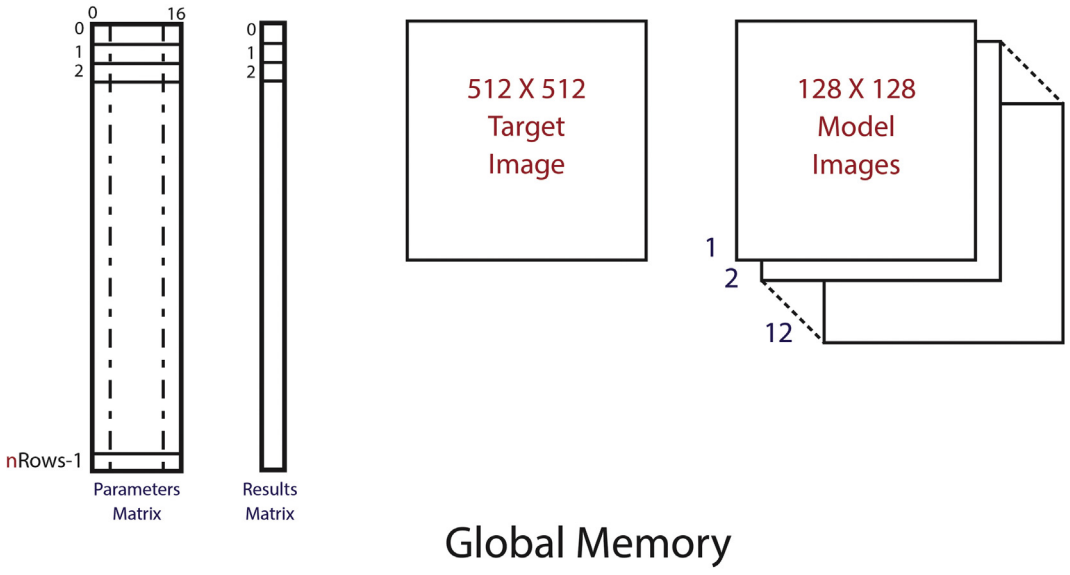

Figure 1. Vectorized inputs/vectorized outputs of the shape based segmentation problem (global memory view).

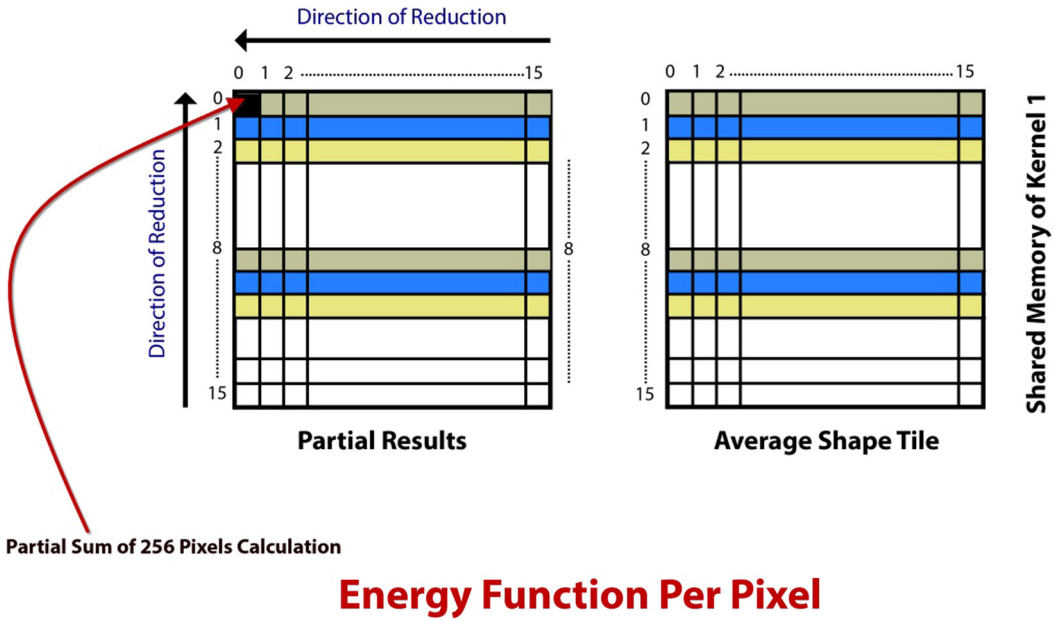

Figure 2. Per pixel energy function calculation and partial sum computation.

Table 1
Average execution time (in seconds, per one parameter set instance) of the objective function for CUDA, C and Matlab.

Number of parameter set instances	CUDA (parallel)		C (sequential)		Matlab (sequential)	
	Vectorized	Non-vectorized	Vectorized	Non-vectorized	Vectorized	Non-vectorized
10	0.0034	0.0130	0.0014	0.0024	0.0258	0.0260
100	0.00282	0.01054	0.00146	0.00154	0.02830	0.02881
1K	0.000722	0.006274	0.001294	0.001345	0.023707	0.023805
10K	0.00065822	0.0049004	0.0013505	0.0013818	0.0264232	0.0264901

vectorized version, we can conclude from the table that the parallel vectorized version of CUDA is faster than the corresponding non vectorized version of CUDA by 8 times (for large number of parameter set instances). Matlab and C are characterized by very small difference in the time of vectorized and non-vectorized cases. This means that there is some optimization engines in Matlab and C that are able to cache repeated similar parameters (like the images). The CUDA version is faster than the Matlab® by about 35+ times. Although this comparison might not be fair because Matlab is not designed for high performance, it ensures that CUDA offers a very powerful high performance computation.

Please cite this article in press as: Yousef AH, Abdelmunim HE. An accelerated shape based segmentation approach adopting the pattern search optimizer. Ain Shams Eng J (2016), http://dx.doi.org/10.1016/j.asej.2016.11.002

4.2. Comparing proposed solution to genetic algorithms with different population size

In this experiment, both the vectorized and non-vectorized versions of the pattern search are compared to the Genetic algorithm with different population size. The pattern search vectorization is inspected with the complete search option. The non-vectorized version of the GA represent the sequential execution of the default number of chromosomes (20 chromosomes). The used genetic algorithm is configured with the default population size (20 chromosomes). Then, several population sizes are used other than the default population size (50 chromosomes and 100 chromosomes).

Fig. 3 shows the average value of the best minimum value of the energy objective function (E), defined in Eq. (3), obtained by each solver for all runs. It shows also the minimum and maximum values obtained and the standard deviation. One solution is better than another if its objective function value is smaller than the other. Pattern search takes reasonable number of function evaluations, and searches through several basins, arriving at a reasonably good solution [25].

The results of the accuracy (measured according to the error function: defined as the amount/percent of overlapping between the resulting shape and the ground truth divided by the ground truth volume, and measured in percentage) and normalized execution time (defined as the solver execution time divided by the fastest solver execution time) are shown in Figs. 4 and 5 to compare the different algorithms.

It is noted that the reported time and minimum function value varies from a run to run, represented by the change between minimum time, average time and maximum time. This was expected for genetic algorithm because it is a stochastic solver. Although pattern search is a deterministic search algorithm, the decision to start each test from a random initial point between the lower bound and upper bound causes the execution time to change slightly from a test to test and from a run to another run.

It is clear from the previous figures that pattern search is considered faster and more accurate than genetic algorithm. Genetic Algorithm takes many more function evaluations than pattern search. It may reach a better solution by chance. Genetic algorithm is stochastic, so its results may change with every run [25].

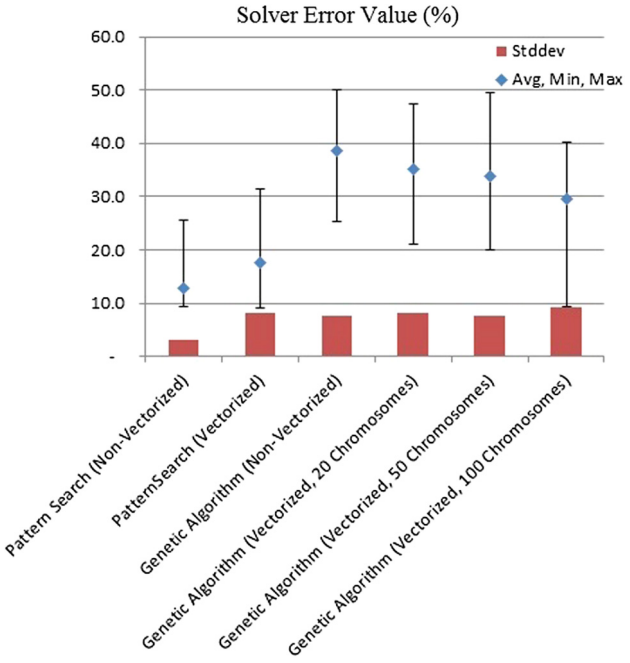

Figure 4. Error comparison of pattern search and genetic algorithm solvers.

In addition, vectorization of the same configuration of any algorithm leads to a faster solution. Also, it is concluded that the increase in the number of chromosomes (population size) in the genetic algorithm leads to a solution with lower objective function value (better solution) and less error.

It is worth mentioning that finding the accurate global optimal solution is usually accompanied by some model noise. Usually the near optimal solutions represent a reasonable and acceptable solution.

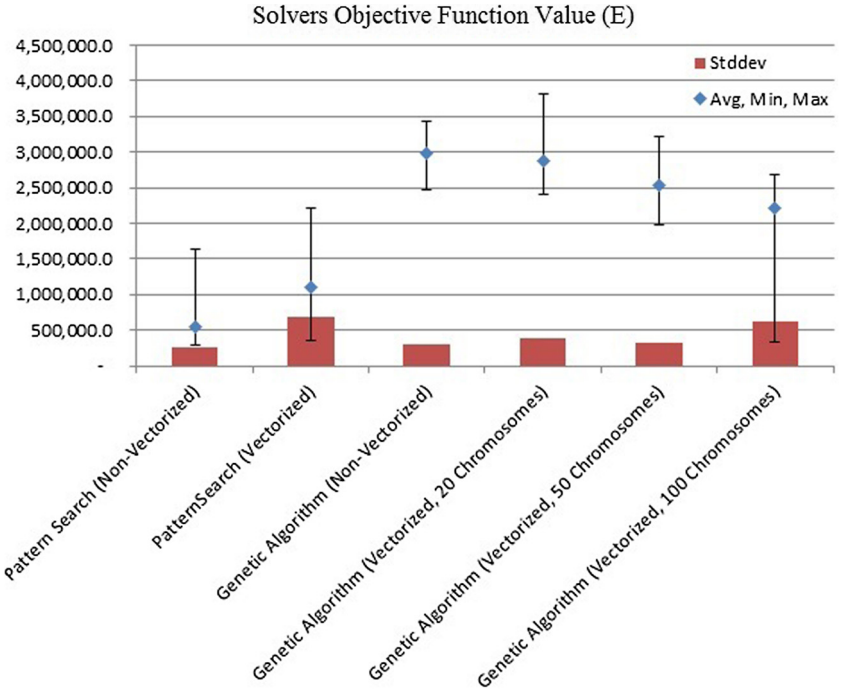

Figure 3. Accuracy comparison of pattern search and genetic algorithm solvers.

Please cite this article in press as: Yousef AH, Abdelmunim HE. An accelerated shape based segmentation approach adopting the pattern search optimizer. Ain Shams Eng J (2016), http://dx.doi.org/10.1016/j.asej.2016.11.002

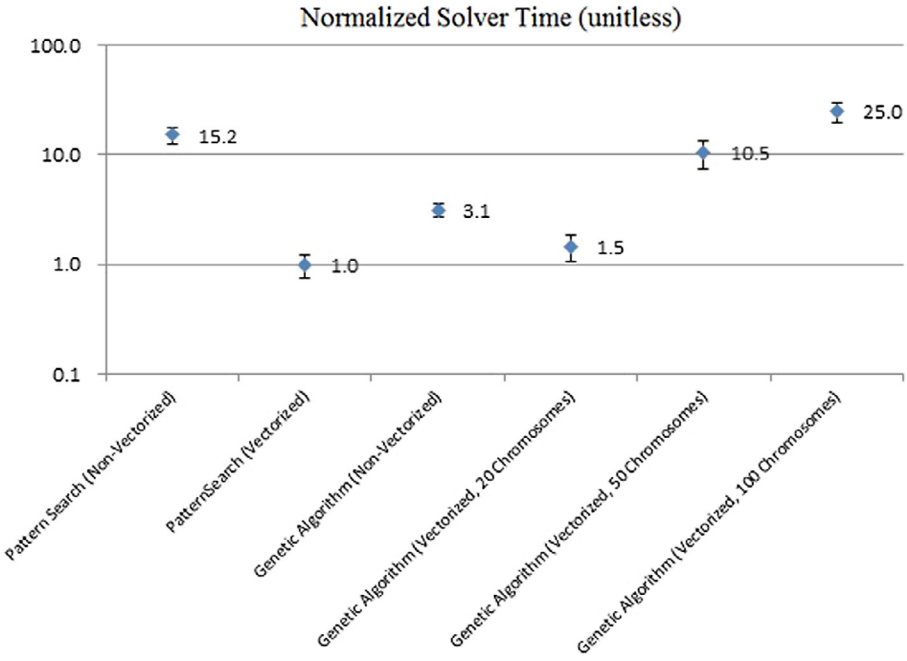

Figure 5. Execution time comparison of pattern search and genetic algorithm solvers.

Part (a) of Fig. 6 shows synthetic images of the fighter jet and number four images. Initialization of a shape model is depicted in Fig. 6 part (b) while final boundaries are detected and illustrated in Fig. 6 part (c).

The results show that the pattern search solver was successful to find the boundaries in spite of the noise and the background lines.

4.3. Comparing proposed solution to the parallel CPU implementation

In Fig. 7, the left part shows a comparison between the Matlab® vectorized version of Pattern Search which runs on different parallel CPU with different numbers of workers and the CUDA version that runs on the GPU. The right part shows the corresponding results for the genetic algorithm.

It is clear from the previous figure that the Matlab® genetic algorithm with CUDA implementation of the objective function is the fastest algorithm. It takes only 1× seconds (as shown in the rightmost orange diamond). On the computation environment mentioned, this takes 7.5 s for one run.

On the other hand, The integrated Matlab® implementation of the pattern search solver with the Matlab® vectorized implementation of the objective function is very slow and takes 20× (as shown by the leftmost red diamond). CPU parallelization of both the pat-

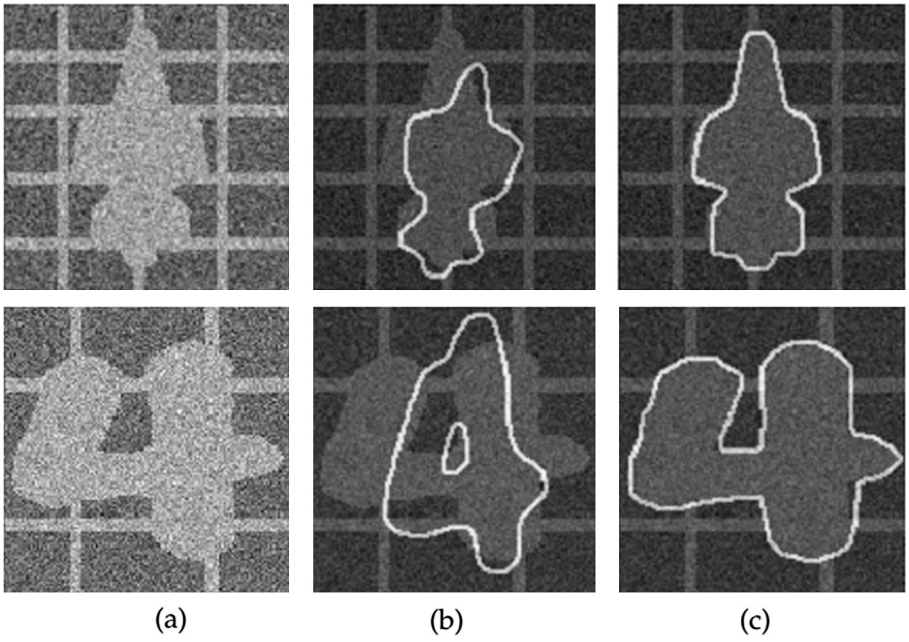

Figure 6. The used data sets, target images, initial shape model and final boundaries.

Please cite this article in press as: Yousef AH, Abdelmunim HE. An accelerated shape based segmentation approach adopting the pattern search optimizer. Ain Shams Eng J (2016), http://dx.doi.org/10.1016/j.asej.2016.11.002

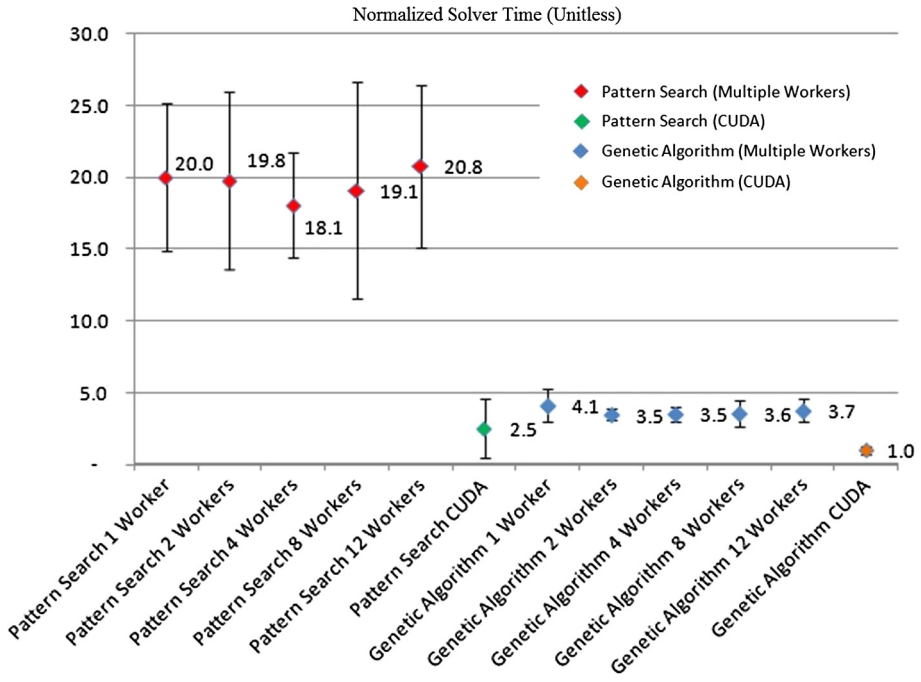

Figure 7. Comparison between multiple CPU and CUDA versions for both pattern search and genetic algorithm solvers.

tern search algorithm and genetic algorithm with different number of workers does not change its speed or execution time significantly. This happens because Matlab parallel computation toolbox is not designed to provide high-performance. In addition, the global optimization process is considered a light process for multiple CPU/Cores (a process that has small number of arithmetic and logic operation that needs small execution time compared with the communication overhead). Aside of the objective function computation which is intensive, other parts of the genetic algorithm and pattern search do not take a lot of computation. For the CPU parallelization to be effective and get the expected speed-up, it is required that each part of the algorithm to be more computationally than that.

The reason for this is that CPU parallelization is implemented in Matlab® by shipping the body of the loop to the workers (separate processes) and then collecting the results. The overhead is large to send the work out, receive the results, and piece the overall answers back together. In brief, the overhead to send the code to multiple processors and collect result and synchronize the processes is larger than the computation.

It is very clear that CUDA implementation of the objective function has superior results compared with CPU implementations for both the pattern search and genetic algorithm.

4.4. Proposed solution performance comparison on different GPUS

In this experiment, the execution time of the GPU implementation is compared on two different GPUs. The first one is the GeForce GT 720M (96 Cores) and the second one is a larger K20 GPU machine with 2496 cores. In Table 2, the results showed a speed up of left part shows a comparison between the Matlab® vectorized version of Pattern Search, different CPU parallel versions with different numbers of workers and the CUDA version. The right part shows the results for the genetic algorithm implemented in both CPU parallel versions and CUDA.

Table 2
C execution time, CUDA execution time and speed up of the objective function on different GPUs.

	GeForce GT 720M	K20
Number of cores	96	2496
Execution time in C (s)	40,836	25,870
Execution time in CUDA (s)	14,694	1080
Speedup	2.8	24

5. Conclusion and future work

In this paper, both the design and implementation aspects of a fast pattern search optimization solver are explored to solve the shape based segmentation problem. The selected objective functions are parallelized efficiently on the GPU. Results show that the accuracy of the proposed solution is better than genetic algorithm. The GPU based implementation is faster than the CPU parallelization option. These results can be used by other researchers to apply the same methodology with other optimization techniques including variants of gradient descend and nature inspired optimization algorithms. We encourage researchers to investigate more GPU features and apply them to achieve higher speedup. For example, the use of surface memory and streams for sending the parameter asynchronously may accelerate the solution.

References

[1] Park In Kyu, Singhal Nitin, Lee Man Hee, Cho Sungdae, Kim Chris W. Design and performance evaluation of image processing algorithms on GPUs. IEEE Trans Parallel Distributed Syst 2011;22(1):91–104. doi: http://dx.doi.org/10.1109/TPDS.2010.115.

[2] El Munim Hossam E Abd, Farag Aly A. Curve/surface representation and evolution using vector level sets with application to the shape-based segmentation problem. Pattern Anal Mach Intelligence, IEEE Trans 2007;29(6):945–58.

[3] El Munim H Abd, Farag Aly A. Shape representation and registration using vector distance functions. In: Computer vision and pattern recognition. CVPR'07. IEEE Conference. IEEE; 2007.

[4] El Munim HE Abd, Farag Aly A. A shape-based segmentation approach: an improved technique using level sets. Computer vision. ICCV 2005. Tenth IEEE International Conference, vol. 2. IEEE; 2005.

Please cite this article in press as: Yousef AH, Abdelmunim HE. An accelerated shape based segmentation approach adopting the pattern search optimizer. Ain Shams Eng J (2016), http://dx.doi.org/10.1016/j.asej.2016.11.002

[5] Aslan Melih S et al. A new shape based segmentation framework using statistical and variational methods. In: Image processing (ICIP), 18th IEEE international conference. IEEE; 2011.
[6] Munim Abd EL, Hassan Hossam El Din. Implicit curve/surface evolution with application to the image segmentation problem. Dissertation Abstracts Int 2007;68(05).
[7] Deb K. An efficient constraint handling method for genetic algorithms. Comput Methods Appl Mech Eng 2000;186(2–4):311–38.
[8] Glover F. A template for scatter search and path relinking. In: Artificial evolution. Berlin Heidelberg: Springer; 1998. p. 1–51.
[9] Kolda TG, Lewis Robert Michael, Torczon Virginia. A generating set direct search augmented lagrangian algorithm for optimization with a combination of general and linear constraints. Sandia National Laboratories; 2006.
[10] Audet C, Savard G, Zghal W. A mesh adaptive direct search algorithm for multiobjective optimization. Eur J Operational Res 2010;204(3):545–56.
[11] Črepinšek M, Liu S-H, Mernik M. Replication and comparison of computational experiments in applied evolutionary computing: common pitfalls and guidelines to avoid them. Appl Soft Comput 2014;19:161–70.
[12] Črepinšek M, Liu S-H, Mernik L. A note on teaching–learning-based optimization algorithm. Inf Sci 2012;212:79–93.
[13] Shen G, Gao G-P, Li S, Shum H, Zhang Y. Accelerate video decoding with generic GPU. IEEE Trans Circuits Syst Video Technol 2005;15(5):685–93.
[14] Modat Marc, Gerard G. Ridgway, Taylor Zeike A., Lehmann Manja, Barnes Josephine, Fox Nick C., et al. Fast free-form deformation using graphics processing units. Comput Meth Prog Biol.
[15] Allusse Y, Horain P, Agarwal A, Saipriyadarshan C. GpuCV: an open source Gpu-accelerated framework for image processing and computer vision. In: Proc. ACM Int'l conf. multimedia. p. 1089–92.
[16] Babenko P, Shah M. MinGPU: a minimum GPU library for computer vision. Real-Time Image Process 2008;3(4):255–68.
[17] Fung J, Mann S, Aimone C. OpenVIDIA: parallel GPU computer vision. In: Proc. ACM Int'l conf. multimedia. p. 849–52.
[18] Lefohn Aaron E, Cates Joshua E, Whitaker Ross T. Interactive, GPU-based level sets for 3D segmentation. In: Medical image computing and computer-assisted intervention-MICCAI. Berlin Heidelberg: Springer; 2003. p. 564–72.
[19] Abramov Alexey et al. Real-time image segmentation on a GPU. Facing the multicore-challenge. Berlin Heidelberg: Springer; 2010. p. 131–42.
[20] Bilgic Berkin, Horn Berthold KP, Masaki Ichiro. Efficient integral image computation on the GPU. In: Intelligent vehicles symposium (IV). IEEE; 2010.
[21] Shams Ramtin et al. A survey of medical image registration on multicore and the GPU. Signal Process Mag, IEEE 2010;27(2):50–60.
[22] Sakr Fatma Zaky, Taher Mohammed, Wahba Ayman M. High performance iris recognition system on GPU. In: Computer engineering & systems (ICCES), 2011 international conference. IEEE; 2011.
[23] Taher Mohamed. Accelerating scientific applications using GPU's. In: Design and test workshop (IDT), 2009 4th international. IEEE; 2009.
[24] Zhu Weihang. Massively parallel differential evolution—pattern search optimization with graphics hardware acceleration: an investigation on bound constrained optimization problems. J Global Optim 2011;50(3):417–37.
[25] Zhu Weihang, Curry James. Multi-walk parallel pattern search approach on a GPU computing platform. In: Computational science–ICCS 2009. Berlin Heidelberg: Springer; 2009. p. 984–93.
[26] The Global Optimization Toolbox User's Guide – MathWorks, <in.mathworks.com/help/pdf_doc/gads/gads_tb.pdf>, Last Accessed May 19th, 2015.
[27] El Munim Hossam E Abd, Farag Amal A. Shape representation and registration in vector implicit spaces: adopting a closed-form solution in the optimization process. Pattern Anal Mach Intell, IEEE Trans 2013;35(3):763–8.
[28] AboelGhar Mohamed. A kidney segmentation approach from DCE-MRI using level sets. In: IEEE computer society conference on computer vision and pattern recognition workshops.
[29] https://blogs.msdn.microsoft.com/nativeconcurrency/2012/04/12/what-is-vectorization/, last accessed: 14/10/2016.

Ahmed Hassan Yousef is an associate professor in the Computers and Systems Engineering Department, Ain Shams University since 2009. He is the vice director of the Knowledge and Electronic Service Center (EKSC), Supreme Council of Universities, Egypt. He got his Ph.D., M.Sc. and B.Sc. from Ain Shams University in 2004, 2000, 1995 respectively. He works also as the vice chairman of the IEEE, Egypt section. His research interests include Parallel Programming, Image Processing, Data Mining, Software Engineering, Programming Languages, Artificial Intelligence and Automatic Control, Technology in Education.

Hossam Eldin Hassan AbdelMunim is an associate professor in the Computers and Systems Engineering Department, Ain Shams University since 2013. He is the director of the Information Technology unit in the Faculty of Engineering, Ain Shams University, Egypt. He got his Ph.D. from the United States in 2007, and got his M. Sc. and B.Sc. from Ain Shams University in 2000, 1995 respectively. His research interests include Parallel Programming, Image Processing, Computer Vision, Medical Imaging, Visualization.

Please cite this article in press as: Yousef AH, Abdelmunim HE. An accelerated shape based segmentation approach adopting the pattern search optimizer. Ain Shams Eng J (2016), http://dx.doi.org/10.1016/j.asej.2016.11.002