A Cellular Automaton Framework for Image Processing on GPU

Claude Kauffmann[1] and Nicolas Piché[2]
[1] *Department of medical imaging, Notre-Dame Hospital, CHUM, 1560 Sherbrooke East, Montreal, QC H2L 4M1 Canada*
[2] *Object Research System Inc. 760 St-Paul W, suite 101 Montreal, QC H3C 1M4 Canada*

1. Introduction

1.1 GPU Computing

Since 1999, which marks the introduction of the Graphics Processing Unit (GPU) for the PC industry, the use of commodity graphics hardware for non-graphic applications has become an important research topic. At the time, GPUs were specially designed for 3D computer graphics based on a fixed-function processor built around the graphic pipeline. These GPUs were difficult to program so the first programming efforts were regarded as academic projects. A major limitation of this generation of GPUs was the lack of floating-point precision in the fragmenting processors. This limitation has vanished with the introduction of floating-point pixel processing with the ATI Radeon 9700 GPU in the late 2002s, and the NVIDIA GeForce FX GPU in the early 2003s. After only a few years, we are now entering the third stage of GPU computing : GPUs are now general-purpose parallel processors (GPGPU) with support for accessible programming interfaces and industry-standard languages. Nowadays, the GPU's have drastically changed the face of computing and a large community of developers successfully applies their findings to GPUs, in various application domains, in order to achieve speedups of orders of magnitude superior to optimized CPU implementations. GPUs are playing an increasing role in non-graphic and scientific computing applications. For a good overview and the state of the art of GPUs architecture, computing and applications, the reader can refer to (Owens et al., 2008, Owens et al., 2007).
The high computational rates of the GPU have made graphics hardware an attractive target for demanding applications such as those in the field of signal and image processing. Among the most prominent applications in this area are those related to image segmentation, a driving application concerning the field of medical imaging.

1.2 Medical imaging context

Medical imaging must deal with increasingly large sized medical image data produced by modern imaging systems such as multidetector computed tomography (MDCT) or magnetic resonance imaging (MRI). The evaluation process of these images is not trivial and can result in missed abnormalities due to the inherent limitations of human perception. These evaluation difficulties increase with the use of 3D or 4D images. In clinical settings it is fundamental to extract quantitative and qualitative information from these images, in a reproducible and reliable way. The basic problem is the identification and extraction of anatomical or pathological features embedded in 2D or 3D images by a process called segmentation. Image processing on large medical volume data rely on computationally demanding applications that can benefit from the development of GPUs.

Medical image processing addresses a large range of problems and for each area of application, the segmentation strategies must be adapted to specific needs and clinical constraints. In several cases, fully automatic segmentation processes fail because the quality of images is directly affected by variability in imaging parameters, human anatomy and pathological changes over time. Therefore, the medical practitioner must take the time to correct the segmentation errors in 3D or 4D images in a tedious and time consuming task that is not compatible with clinical workflow. Medical images are complex and, in absence of higher level knowledge, this may result in several different interpretations. It is therefore imperative to incorporate some user intervention, which adds prior information in order to guide the segmentation process towards a reliable solution. This approach is suited in a clinical context because segmentation errors and computing time can be reduced within the same step. The principle is that high level image understanding is used to initialize low-level tasks operated by the computer. Specifically, the medical practitioner can draw rough scribbles labeling the regions of interest (each label related to a specific object) and then the image is automatically segmented by propagating these labels over the whole image. If needed, the user can add more seeds to achieve the ideal result.

Seeded segmentation approaches have been widely used in a very inspiring way in various domains, such as segmentation and matting of photo/video or in the treatment of natural images (Bai and Sapiro, 2007), cartoons or natural image colorization (Qu et al., 2006, Konushin et al., 2006), and interactive segmentation for image compositing (Vezhnevets et al., 2005, Yin et al., 2004, Rother et al., 2004, Mortensen and Barrett, 1998). These segmentation approaches rely on weighted-distance-based techniques. From each user-provided label, a weighted distance is computed in order to find out the probability for each unlabelled pixel to be assigned to a particular label. In these techniques, the image grid is seen as a graph with pixels as nodes and edges connecting neighboring pixels. Since the work initiated by Boykov and Jolly (Boykov and Jolly, 2000), the Graph Cuts method has become a very attractive way for interactive organ segmentation in medical imaging and different work based on weighted distance for image segmentation has been published (Xu et al., 2007, Protiere and Sapiro, 2007, Chefd'hotel and Sebbane, 2007). Additionally, the geodesic weighted distances can be seen as a specific case of the more general technique presented in (Falcão et al., 2004) as well as the $q = \infty$ case presented by Sinop and Grady (Sinop and Grady, 2007). In this last publication, the authors showed a general seeded image segmentation algorithm based on the minimization of ℓ_q norms. They showed that two popular seeded image segmentation algorithms, Graph Cuts (Boykov and Funka-Lea, 2006) and Random Walker (Grady and Funka-Lea, 2004), correspond to the parameter choices of q = 1 and q = 2 respectively.

Since graph-cuts algorithms are computationally heavy on large datasets, recent work has been done to find solutions to implement these algorithms on graphics hardware. Two different approaches are found.

The first looks at the new GPU implementation of the push-relabel algorithm (Vineet and Narayanan, 2008, Dixit et al., 2005) to solve the mincut/maxflow problem. Reported performances on 2D images show that the GPU approaches are very promising (Vineet and Narayanan, 2008). Nevertheless, it is important to notice that the implementation was performed using CUDA, a parallel computing architecture developed by NVIDIA, which works with all NVIDIA GPUs from the G8X series or newer. Compared to low level languages, CUDA needs only a short learning curve to build proof of concepts but the application is limited to NVIDIA cards.
Another approach is to focus on efficient GPU implementation for large sparse matrix solver (Volkov and Demmel) required by spectral methods for image segmentation such as Normalized-Cuts, or to find the solution to a sparse linear system as needed by the isoperimetric algorithm (Aharon et al., 2005). Implementation of these algorithms is not straightforward and for this reason, we focus our energy on developing simple iterative algorithms which can be formulated as a cellular automaton.

1.3 Cellular automaton: a model suited to GPU computing

Since the days when Von Neumann and Ulam (Von Neumann and Burks, 1966) first proposed the concept of cellular automata (CA) until the recent book of Wolfram `A New Kind of Science' (Wolfram, 2002), the simple structure of CA has attracted researchers from various disciplines. An exhaustive survey of literature, on cellular automata for modeling purposes, can be found in (Ganguly et al., 2003). CA became more practical and immensely popular in the late 1960's, when John Conway developed the cellular automata based *Game of Life* - an elementary computerized model of a colony of living cells (Gardner, 1970). The popularity of cellular automata can be explained by the enormous potential that they hold in modeling complex systems, in spite of their simplicity.
A cellular automaton (CA) is a collection of cells arranged in an N-D lattice, such that each cell's state changes as a function of time according to a defined set of rules that includes the states of neighbouring cells. Typically, the rule for updating the cells state is the same for each cell, it does not change over time and it is applied to the whole grid simultaneously. That is, the new state of each cell, at the next time step, depends only on the current state of the cell and the states of the cells in its neighbourhood. All cells on the lattice are updated synchronously. Thus, the state of the entire lattice advances in discrete time steps. It is clear that the concept of parallelism is implicit to cellular automata. In terms of computing, we need to process many elements (cells) with the same program. Each element is independent of the other elements and basically, elements cannot communicate with each other. These constraints match exactly with the GPU programming model, which is called SPMD and stands for Single Program, Multiple Data. Technically, each element can operate on a 32-bit integer or on floating-point data with a reasonably complete general-purpose instruction set. Elements can read data from a shared global memory (a "gather" operation) and with the newest GPUs, they can also write back onto arbitrary locations in shared global memory ("scatter") (Owens et al., 2008). This programming model is well-suited for programs formulated as CA, as many elements can be processed in lockstep running the exact same code. Code written in this manner is called "SIMD", for Single Instruction, Multiple Data.
The most recent published papers using CA and GPU cover simulation of physical (Zhao, 2008), biological (Gobron et al., 2007) or physiological (Alonso Atienza et al., 2005)

processes. It would not be surprising to see an increase in the popularity of CA algorithms now that powerful GPUs are widely available, at low cost, on home computers.

2. Our contribution

Our work seeks to develop simple CA algorithms to perform efficient multi-label segmentation tasks on, but not restricted to N-Dimensional medical images, and implement them on low cost graphical hardware (GPUs).
In this paper we propose a massively parallel implementation of the Ford-Bellman's shortest paths algorithm (FBA) to perform graph-based segmentation.
Two applications based on FBA are presented. The first focuses on ultra-fast computation of the watershed transform on N-D images, while the second presents an alternative and an optimized framework to perform graph-based seeded segmentation on N-D images. It is important to notice that our work was voluntarily designed as a compromise between computational efficiency and hardware compatibility. Indeed, our FBA approach was implemented using OpenGL-Cg language on low cost, non vendor-specific graphics hardware. This aspect becomes interesting in a commercial software context because it can run on recent or moderately old hardware (NVIDIA GeForce 6 or ATI 1000 graphic card families).

3. Method

The N-D image is treated as a discrete object and is seen as a graph where each pixel (voxel) is a graph node or vertex. A predefined cost-function is used to characterize the edges abutting two adjacent nodes included in the neighborhood. The cost-function is used to compute multiple shortest-path-trees (sp-tree) where the tree roots are specifically labeled vertices called seeds. The segmentation result is obtained by cutting the graph in order to separate it into two, or more, sets. The algorithm starts by computer or user defined seed groups on the image. Each seed group is characterized by a location and a specific label so that the K-labels belong to K-specific objects in the image. The segmentation algorithm iteratively evolves from these starting labels, so that at the end, the N-D image is segmented in K objects. Each region is then guaranteed to be connected to seed points with the same label.

In the following, we consider an N-D gray scale image as a graph $G=(V,E)$ with a set of V vertices, or nodes, and a set of edges $e \in E \subseteq VxV$ spanning two neighboring vertices, v_i and v_j defined by its neighborhood. The weighted graph assigns a value to each edge e_{ij} called a weight or a cost and is denoted by c_{ij} or $c(e_{ij})$ and defined by $f : E \to R$, a real-valued weight function. $\lambda(v)$ and $l(v)$ represent the value and the label assigned to a vertex respectively. S^k is a start (or source) vertex with label k or a group of vertices with the same $k-label$. For each unlabeled vertex we can compute the minimal cost to reach the source s as the sum of the weights of the edges in the path. If we compute a sp-tree for k-labeled source vertex S^k, each unlabeled vertex of the graph can be represented by a K-tuple vector which represents the K-cost to reach the K-label. As shown by equation (1), the global cost, noted C_i^k is computed as the cumulative cost of the shortest path P^k between vertices v_i and the S^k seed group.

$$C_i^k = \sum f\left(P_i^k\right) \tag{1}$$

The *i-th* vertex can then be represented by a K-tuple vector $v_i\left\{C_i^j\right\}$ with $j \in 1, K$. The final vertex labeling process, $l(v)$ in (2), is derived from these K-tuples by tagging each vertex with the nearest K-label. The label assigned to vertex v_i is the label of the minimum cost C_j.

$$l(v_i) = label\left\{\min_{j=1}^{j=K}\left(C_i^j\right)\right\} \tag{2}$$

We show that this method can be used to partition a graph in K sub-graphs by performing K-cuts. It means that the image is split in K-specific regions including the k-seed groups. In practice, computing K times the sp-tree is not an optimal approach. We show in the next section that the multiple-sp-tree can be computed in a more effective way.

3.1 Dijkstra's and Ford-Bellman shortest paths algorithm

Dijkstra's shortest paths algorithm (Dijkstra, 1959) is the most popular method to compute the shortest path between two vertices (s,t), or between a start vertex s and all other vertices v_i in a graph. Dijkstra's algorithm is a heap-based method with computational complexity in O (N log N), but this complexity can be reduced by the use of techniques like that presented in (Yatziv et al., 2006). However, generally speaking, priority queues are very difficult to parallelize and we aim to develop other implementation strategies to compute the shortest paths.
A different approach used to calculate the distance of all vertices v_i from a defined vertex s is the Ford-Bellman's algorithm (Bellman, 1956, Ford Jr, 1956) described in Algorithm 1 below. This 50-year-old algorithm gives a concise generalized expression of the cost-minimization problem. The most crucial, unique and unintuitive aspect of this algorithm is that even though the vertices can be processed in any order (even randomly), the algorithm will, in the end, produce the lowest-cost distance from every vertex to the start vertex (Even, 1979). Each vertex is simply relaxed several times until the algorithm converges to the stationary global solution. This important aspect of the Ford-Bellman's algorithm has caught our attention because it allows an efficient parallel implementation that can be achieved by cellular automata as presented in the following section.

```
Algorithm 1: The Ford-Bellman's algorithm
$\lambda(s) \leftarrow 0$ and for every $p \neq s, \lambda(p) \leftarrow \infty$
as long as there is an edge $p \to q$, such that $\lambda(p) > \lambda(q) + w_{pq}$ then
$\lambda(p) = \lambda(q) + w_{pq}$
$label(p) = label(q)$
```

This algorithm generalizes Dijkstra's algorithm for graphs having negative arc weights but without cycles of the negative weights. Unfortunately, on conventional sequential computers, the algorithm takes *O(n*^{3}*)* time to generate a complete connected *n*-vertex weighted graph. A hardware specific parallel implementation is proposed by (Nepomniaschaya, 2001). In this paper, the author introduces a natural straightforward matrix representation of the Ford-Bellman algorithm on a STAR-machine. The STAR-machine is an associative parallel system of the SIMD type with vertical processing. The goal of the study was to represent the Ford-Bellman algorithm as a corresponding STAR procedure, to justify its correctness and evaluate time complexity.

3.2 Expression of FBA as a cellular automaton

CA is a collection of cells arranged in an N-D lattice, such that each cell's state changes as a function of time according to a defined set of rules that include the states of neighbouring cells. That is, the state of a cell at a given time depends only on its own state, at the previous time step, and on the states of its neighbourhood cells at the previous time step. All cells on the lattice are updated synchronously. Thus the state of the entire lattice advances in discrete time steps. For a 2D image, the Moore or von Neumann neighbourhoods can be used, while in 3D, the natural extension of these neighbourhoods gives us 6 and 26 neighbours respectively. Following this definition, CA can easily be applied to N-D images (lattice) represented by a graph G, where cells are pixels (or voxels) of this image and vertices of the graph.

We can then write the CA rule that computes, for each time step (t), the Ford-Bellman's algorithm by the following equation :

$$\lambda^{t+1}(p) = \min\left[\lambda^{t}(p), \min_{q \in N_p}\left\{\lambda^{t}(q) + w_{pq}\right\}\right] \tag{3}$$

Where N_p is the neighbourhood of *p*.

Equation (3) shows that the Ford-Bellman's CA rule can be written in a very efficient and concise form. The principle of CA is then to apply this transition rule (3) synchronously for all cells and to iterate as long as any cell changes its state. The relation (3) computes the shortest path of all vertices to a set of specific initial seeds. In this case, at the end of the algorithm, all vertices are labeled with the seed label. In a more general case, the user starts to define interactively, on the image, K-group of seeds having K-labels, so that K-labels belong to K-specific objects. In this case, the seeds label must be spread out at the same time as the cell state is updated. We can then write the pseudo-code of the evolution rule of the CA (Algorithm 2) for the FBA when K-labeled seed groups are specified. At least, the user specified 2 labels, one for the object and the other for the background.

For iteration $k+1$, cell labels $label^{k+1}$, and cell states λ^{k+1}, are updated as follows:

```
Algorithm 2: CA rule for K-seeded weighted distance map
$s_l \in V$ Where $l \in [1, K]$ with $K$ the total number of label
$\lambda(s_l) \leftarrow 0$ and for all $p \neq s_l$, $\lambda(p) \leftarrow \infty$
$label(s_l) \leftarrow l$ and for all $p \neq s_l$, $label(p) \leftarrow 0$
for $\forall p \in V$
        $U^k(p) = \min_{q \in N_p} \{\lambda^t(q) + w_{pq}\}$
        $\lambda^{k+1}(p) = \min[\lambda^t(p), U^k(p)]$
        $label^{k+1}(p) = label\{\min[\lambda^t(p), U^k(p)]\}$
end for
```

3.3 Neighbourhood and Cost function

For a 2D image, the Moore or von Neumann neighbourhoods can be used, while in 3D, the relation below gives the 3D edge set connectivity E in the case of 6, 18 and 26-connected for N=1, $\sqrt{2}$ and $\sqrt{3}$ respectively.

$$E = \left\{ p, q \middle| \left\| C(p) - C(p) \right\|_2 \leq N \right\} \tag{4}$$

Where $\|\cdot\|_2$ is used to denote the standard Euclidean norm and $C(p)$ maps voxel p to its 3D coordinates.

As for many graph-based segmentation algorithms (Boykov and Funka-Lea, 2006, Chefd'hotel and Sebbane, 2007, Grady and Funka-Lea, 2004, Protiere and Sapiro, 2007), the edge weights encode image intensity changes between two neighboring graph nodes *p, q*. In a general case, the cost w_{pq} can be represented by the following relation:

$$w_{pq} = f\left(I_p - I_q\right) + h \bullet \left\| C(p) - C(q) \right\|_2 \tag{5}$$

The parameter h is used, if needed, to add a regularization term which represents the geometric distance between two vertices.

3.4 FBA to compute the Watershed transform

Watershed transform is one of the most popular methods for image segmentation. The watershed transform was originally proposed by (Digabel and Lantuejoul, 1977) and later improved in (Beucher and Lantuejoul, 1979). The watershed method can be formulated in a general framework called image labeling, where a label is associated to each pixel from a finite set. The intuitive idea underlying this method comes from geography : when a landscape or topographic relief is flooded with water, watersheds are the dividing lines of

the basins of attraction of rain falling over the region. The various formalizations, definitions, algorithms and implementations of the watershed concept can be divided into two classes.
One is based on the specification of a recursive algorithm by (Vincent and Soille, 1991) and the other one is based on distance functions by (Meyer, 1994). Moreover, watershed methods are usually based on sequential algorithms but during the last decade, serious efforts were made to find parallel implementation strategies (Eom et al., 2007, Noguet, 1997, N. Moga et al., 1998). Unfortunately, despite the use of all the techniques and architectures, there is always a stage, in the watershed transform, that remains a global operation. Therefore, only modest speedups are to be expected in the case of parallel implementation (Roerdink and Meijster, 2000). Our approach shows efficient parallel implementation of the watershed transforms based on a cellular automaton (CA) that computes Ford-Bellman's shortest paths (Kauffmann and Piche, 2008).
Our CA algorithm, presented above, does not produce watershed lines. All pixels are merged within some basin, so that the set of basins tessellates the image plane. This is a consequence of the local condition of the CA algorithm that is very advantageous for a parallel implementation of the watershed transform (Roerdink and Meijster, 2000). Unlike other parallel algorithms, our CA algorithm is deterministic and the result does not depend on the order in which the pixels are treated during the execution of the algorithm. This is a consequence of a fundamental aspect of CA, which is that all cells on the lattice are updated synchronously at each time step.

3.5 GPU implementation of FBA

The graphic card is a Single Instruction Multiple Data (SIMD) computer. This type of computer was not commonly available before its introduction into graphic processor, SIMD machines were principally dedicated to signal processing or other tasks. In the 90's there was great interest around these types of computers (for example the famous Connection Machine by Thinking Machines Corporation Inc.) but it was constrained to the research area. Huge improvements in regard to the speed of classical computers and the inherent programming difficulty of SIMD made them less attractive. The expertise needed to program this kind of devices was either lost or not developed. Nowadays SIMD are available at very low cost, so their use became widespread. However, to program a SIMD computer, one often needs to completely reformulate algorithms. Not all algorithms are well suited for GPU; ideal GPGPU applications have large data sets (in regards to the memory available on the graphic card), high parallelism, and minimal dependency between data elements. GPUs are perfect devices to execute CA code: this is due to the fact that CA algorithms only rely on local information (nearest neighbours and local states are usually sufficient). The only complication lies in that we are not aware of the sweeping order of the cells in our CA by the GPU (we do not know if a memory cell has yet to be processed or not). This means that you can have neighbours that are at time (t+1) and others at time (t). In order to update synchronously all the cell states, we need to use some kind of a buffer.
Let's describe how we have programmed these devices to implement the FBA algorithm: To contain the data and the results, we used a RGB 32 bits float graphic texture (double precision floating point will be available in near future on high end graphics cards). In the red Channel (R), we put the image data, in the green channel (G), we put the computed weighted distance map and in the Blue channel (B), we updated the label associated to each

vertex. The vertex labeling step could have also been computed from the converged distance map but it is more efficient to update the labels at each iteration because a channel is available to store this information. Since all vertexes are updated synchronously in lockstep, we needed to have some kind of a buffer. The buffering technique used is the famous Ping-pong scheme, also called 'double buffering'. It is a programming technique that uses two buffers to speed up a computer that can overlap I/O with processing. Data in one buffer is being processed while the next set of data is read by the other one. This buffer technique does not need extra video memory. The 'shader' program, that has to be executed on each vertex, is written in OpenGL Cg, which runs on almost all hardware. It is a strait forward implementation of the FBA equation, as described in algorithm 2. The CA is iterated a predefined number of times and the label information are extracted from the Blue channel. In the cases where the algorithm did not fully converge, a few iterations have been added from this non converged state to reach the expected segmentation results. The user can also add or remove seeds and run the 'shader' code for an additional number of iterations.
Our approach offers the great benefits of running on older hardware and of being compatible with new graphic cards. We could have implemented a more efficient algorithm using CUDA, but this avenue is much too restrictive in terms of hardware requirement.

4. Experimental results

Two different studies, based on the FBA-GPU approach, were conducted. In the first study we applied the algorithm to compute the watershed transform on 2D and 3D images. The computational efficiency of our GPU approach was compared to a CPU optimized version of the watershed transform.
The second study concerned the use of the FBA-GPU approach to perform seeded segmentation of organs in medical image data sets. At first, we evaluated the reproducibility and accuracy of our GPU-FBA segmentation approach applied to 3D kidney segmentation on a retrospective study totaling 20 magnetic resonance angiography (MRA) acquisitions. Then, we performed a comparison between the computing performances of our FBA-GPU approach and a CPU implementation of our algorithm. Finally, the FBA-GPU was benchmarked on available graphic hardware's in order to assess the improvement of computation time on different graphic cards.

4.1 Technical specifications

All experimental results presented in the following studies have been performed on an Intel Xeon Dual Core (3 GHz) with 2GB RAM and an ATI Radeon X1950 Pro graphic card with 512 MB of graphical memory.
It is important to notice that our work was voluntarily designed as a compromise between computational efficiency and hardware compatibility. Indeed, our FBA approach was implemented using OpenGL-Cg language on different brands of low-cost graphic cards. This aspect becomes interesting in a commercial software context because it can run on recent or moderately old hardware, even on a NVIDIA GeForce 6 or on ATI 1000 graphic card families.

4.2 Parallel Watershed transform

In order to illustrate CA-Watershed results, the algorithm 2 was applied to different images by using the two following valued functions.

$$f_A = \left|\nabla(I)\right| \quad and \quad f_B = \left|\nabla(G_\sigma * I)\right| \tag{6}$$

where *G* is a Gaussian smoothing function. The watershed results are represented by mosaic images where each labeled region is filled by the mean value of the pixels inside this region. The starting seeds, s_i , are defined automatically as the local minima of the input image *I*, such as:

$$I(s_i) < \min_{q \in N_{si}} (I(q)) \tag{7}$$

Where N_s is defined as the neighborhood of *s*.

The first 2D application represents a magnetic resonance image (MRI) of a kidney along a sagittal plane (figure 1) while the second one is the popular image of Lena (figure 2). These examples show that the watershed regions give a regular partitioning of the image and a coherent and smooth representation of boundaries. On both images we applied the watershed transform by increasing the size of the Gaussian filtering kernel, with the smallest kernel represented in (a) and the largest one in (c). As the kernel size increases, the number of local minima decreases which results in greater watershed regions, as illustrated by figures 1 and 2.

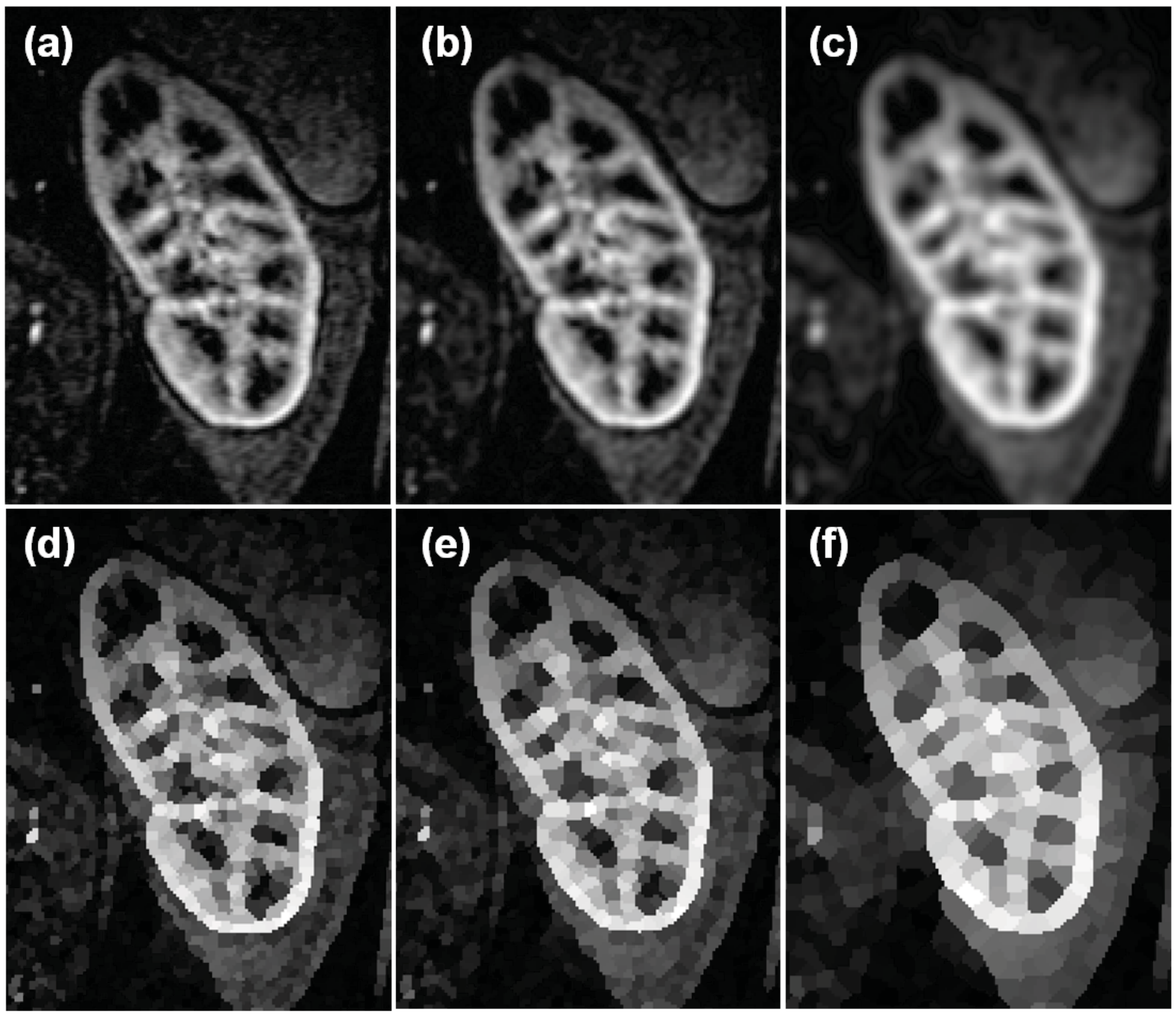

Fig. 1. MRI image of a kidney along the sagittal plane (a), kidney image filtered by a Gaussian (b, c), and respective watershed mosaic images (d, e, f).

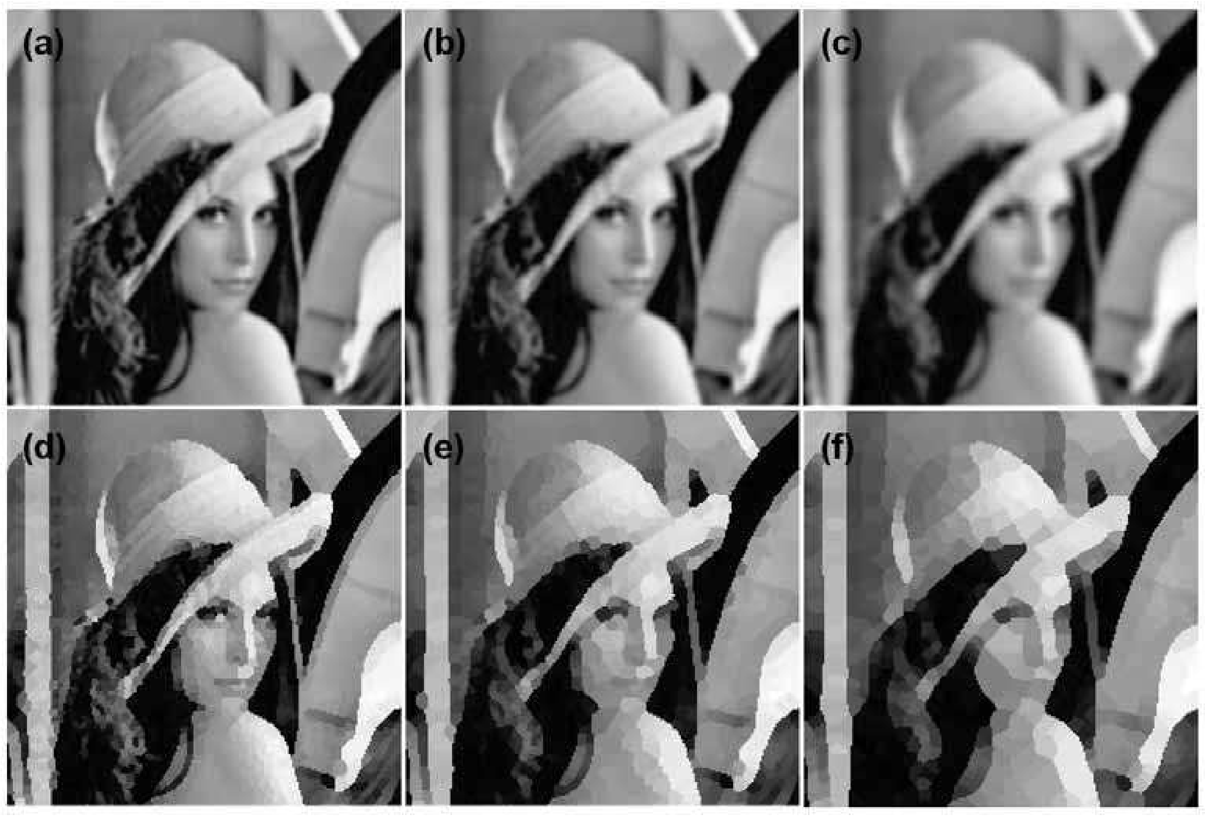

Fig. 2. Image of Lena (a) and smoothed versions (d, c) by a Gaussian function. Watershed mosaic (d, e, f) computed on gradient of respective images (a, b, c).

As we can see in the second study, the number of iterations needed so that the FBA algorithm converged, depends on the Euclidian length of the longest geodesic path between k-labelled vertices. We showed intuitively that computing a watershed transform can be an ideal case for the FBA application because the distance between two k-labelled seeds (two local minima in a specific neighborhood) is generally small, so that only few iterations are needed to reach a converged result. This is especially true in the context of noisy images, such as in medical image datasets, where the local minima are regularly spaced over the whole image dataset. Filtering the image will increase the distance between the local minima, so that more iterations are needed to ensure the convergence of the algorithm. A consequence of this shows that the FBA method is more computationally efficient when the local minima are regularly distributed over the image space and when the distance between the minima is small.

The computing efficiency of our GPU-CA-Watershed was evaluated by comparing its running time to the running time of a C++ implementation of the Tobogganing algorithm described in (Fairfield, 1990, Lin et al., 2006). The initial 3D images used for testing was based on an isotropic CT-scan acquisition of size 512 by 512 by 512 pixels. This dataset was downsampled, using the nearest neighbor method, In order to obtain isotropic datasets sized as a multiple of 64 by 64 by 64 pixels. The CPU and GPU watershed algorithms were applied to all 3D datasets and the respective computing times were recorded.

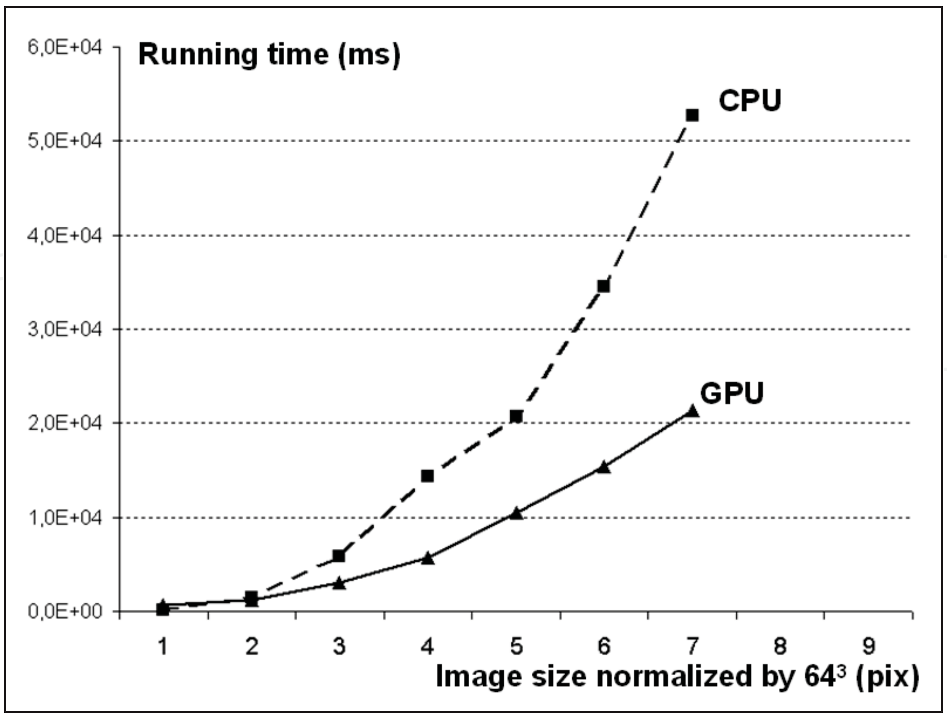

Fig. 3. Comparison between CPU and GPU computing time (ms) of watershed transform on box sized image datasets.

The results of our experiment, illustrated on figure 3, show that the GPU CA-Watershed performs 2.5 times faster than the C++ optimized version of the Tobogganing watershed. Moreover, this speedup factor in favour of GPU can easily be increased by using more recent low cost graphic hardware, as shown in Table 3 in the following section.

4.3 FBA for seeded segmentation

The second study seeked to evaluate the reproducibility and accuracy of the FBA method applied to the segmentation of renal volumes on a retrospective study totaling 20 magnetic resonance angiography (MRA) acquisitions. The computational performances of our FBA-GPU approach were compared with a CPU implementation of the Dijkstra's shortest paths algorithm. Finally, the FBA-GPU implementation was benchmarked on available graphic hardware in order to assess the improvement of computation time on different graphic cards.

The proposed CA-GPU segmentation technique was validated experimentally by measuring the renal volumes on MRA acquisitions of twenty patients affected by symptomatic renovascular disease. The context of validation was to apply our method to perform an automatic 3D segmentation of the kidneys based on user defined labelled seeds. From these segmentation results, the Parenchymal renal volumes were computed.

Study setup : All MRA were performed on a 1.5 T Magnetom Vision unit (Siemens, Erlangen, Germany) using a phased array body coil. The sequence used was a coronal 3D gradient echo centered on the aorta and the kidneys during dynamic IV administration of a contrast agent. The typical image matrix size (XYZ) was 512 by 512 by 150 pixels, with an associated pixel resolution of 0.82, 0.82 and 1.25 mm. A representative sample of our MRA database can be shown in figure 3. In these images, the kidneys are represented in a coronal section where two major structures can be observed: the superficial part is the renal cortex (enhanced signal) and the deep part is the renal medulla (appearing in grey or black). The kidney is a bean-shaped structure which presents concave and convex surfaces. The goal was to segment the 3D kidney Parenchyma (cortex and medulla). There were three main challenges : firstly, the kidney borders are not well defined in the concave surface, because it is the point at which the renal artery enters the organ;, secondly, renovascular disease directly affects the quality of MRA acquisitions which results in poor contrast between the kidney and the background as shown in figure 1b. Finally, the segmentation method had to be able to deal with cortical cysts (appearing, for example, as a black hole on the right kidney of figure 1d) that had to be excluded from the segmentation.

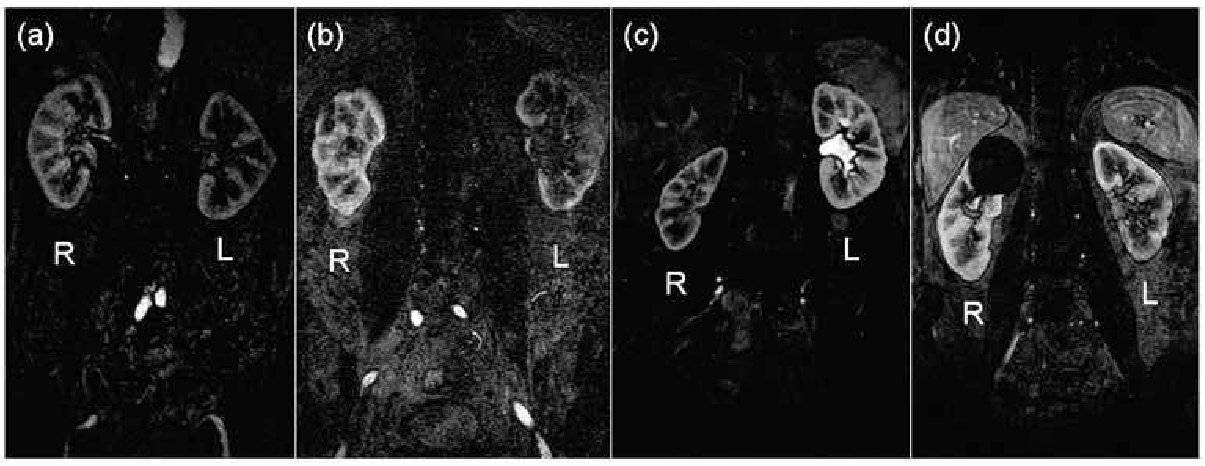

Fig. 3. MRA acquisitions under a coronal 3D gradient echo sequence centered on the aorta and the kidneys during dynamic IV administration of a contrast agent. The kidney is seen as a Bean-Shape structure where two major structures can be observed: the superficial part is the renal cortex (enhanced signal) and deep part is the renal medulla (appearing in grey or black). Significant morphologic and pathologic differences can be observed on the image sample which challenges the segmentation. Poor contrast between the cortex and the background can be observed on the left kidney (L) on figure 1b and 1a. Cortical cysts (appearing, for example, as a black hole on the right kidney of figure 1d) were excluded from the segmentation.

Two readers used the FBA-GPU software tool in a blinded manner to segment the 40 kidneys in MRA images. The automatic segmentation was started from user defined labelled seeds. In order to standardize the segmentation protocol between the readers, the two following steps were established. During the first step, the reader navigated in the 3D MRA dataset through an orthogonal multiplanar reconstruction (MPR) using a mouse-driven synchronized 3D cursor. As illustrated in figure 4, the user moved the cursor near the centre of the kidney so that the renal artery appeared in the axial view.

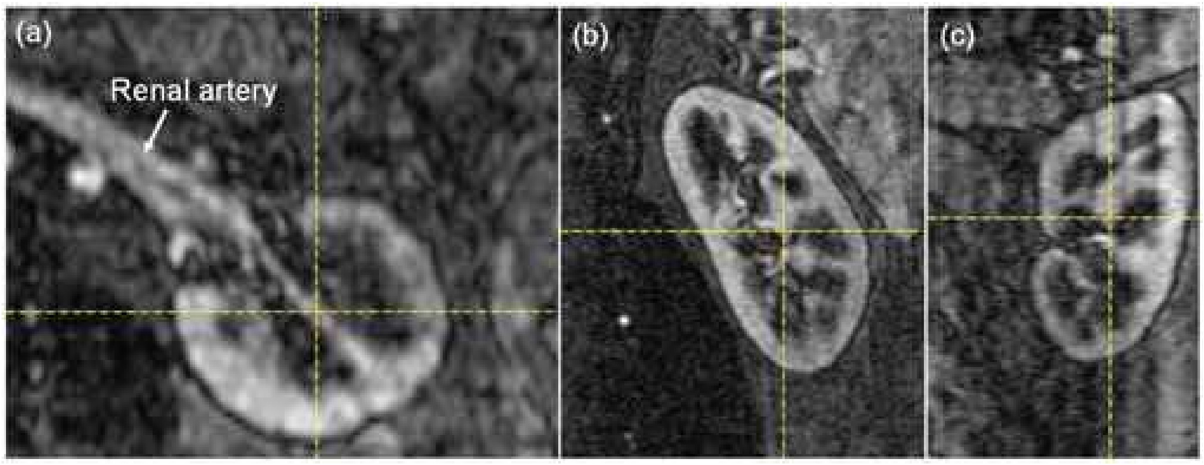

Fig. 4. Multiplanar reformatted images of the kidney as an axial plane (a), a sagittal plane (b) and a coronal plane (c). A synchronized cursor (yellow dashed lines) was used to navigate in the 3D data set. The reader moved the cursor near the centre of the kidney so that the renal artery appeared in the axial view.

In a second step, based on these three selected planes, the user roughly drew two groups of labelled seeds using a painting tool as shown in figure 5. The red seeds needed to be defined inside the kidney while the blue ones were drawn outside the kidney (Fig. 5). The FBA-GPU process was then started and automatically stopped after a fixed number of iterations. The corresponding 3D kidney segmentation result is illustrated in figure 6. The number of iterations was set to 100 (conservative value) for all kidney segmentations in order to ensure the convergence of the algorithm, as will be discussed further.

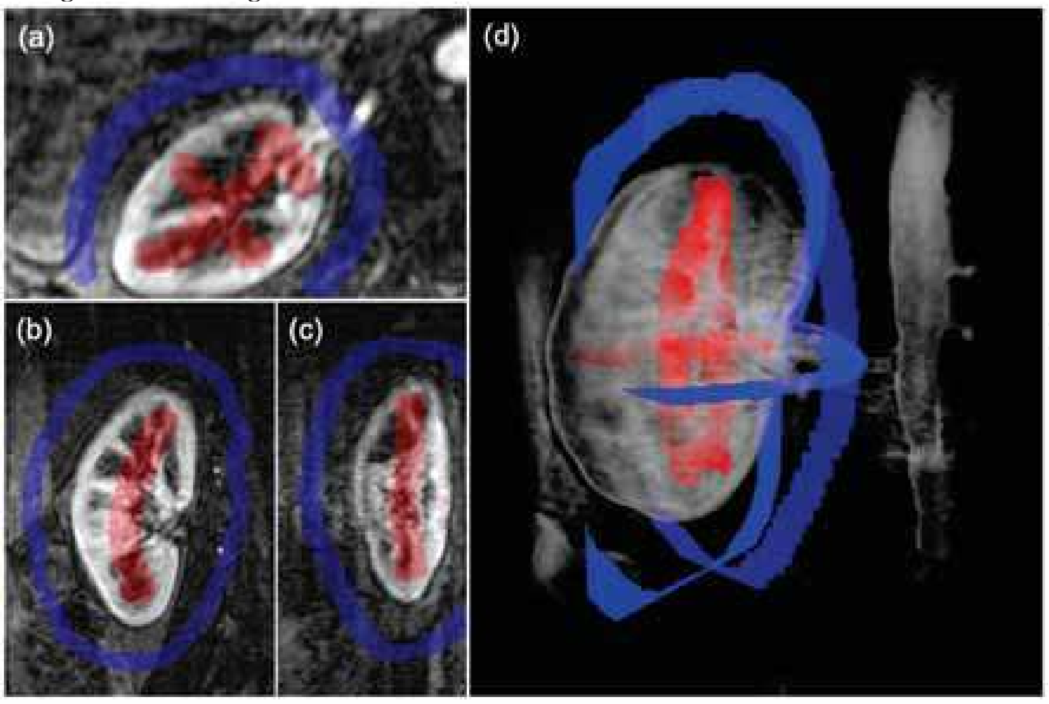

Fig. 5. Initialization of the segmentation algorithm on axial (a), coronal (b) and sagittal (c) planes. The user roughly drew two groups of labeled seeds using a painting tool; the red

seeds were defined inside the kidney while the blue ones were drawn outside the kidney. 3D view of seeds positioning is illustrated on (d).

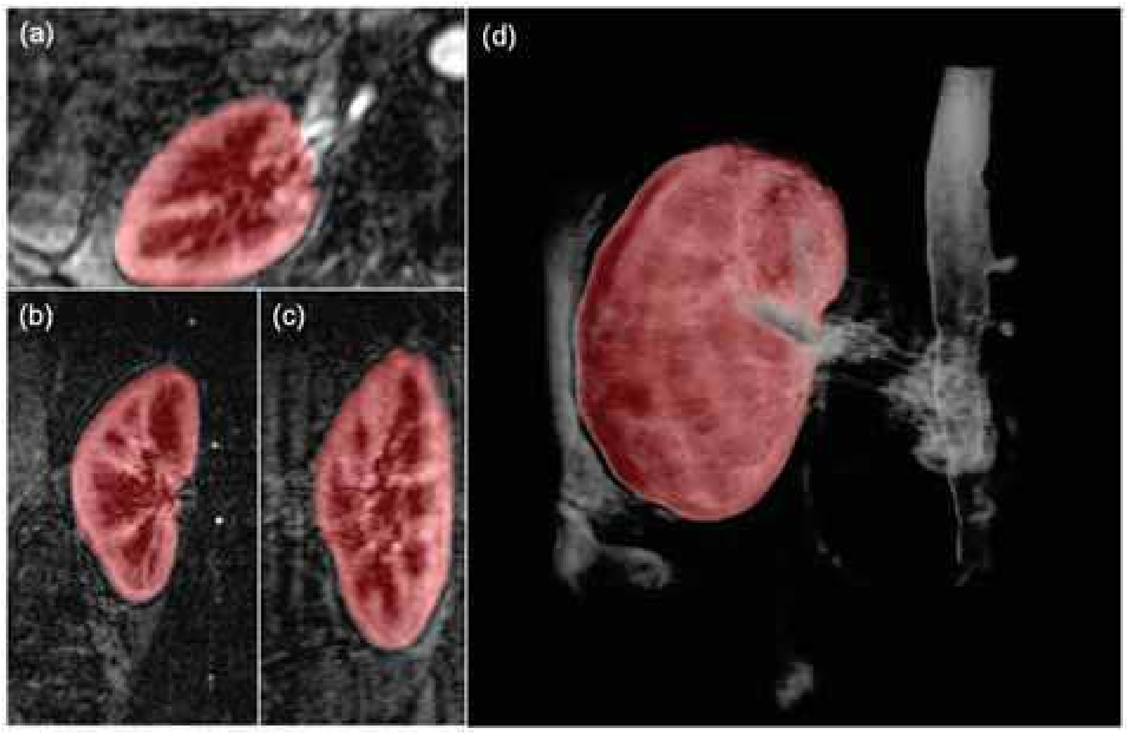

Fig. 6. Example of a segmented kidney. The red overlay represents our automatic segmentation results obtained by our FBAGPU approach after 100 iterations. Kidney VOI is shown in red on axial (a), coronal (b) and sagittal (c) planes, and in 3D view (d).

Reliability: The two readers performed the 3D segmentation of 40 kidneys (from the 20 MRA) using the FBA-GPU approach.The mean and standard deviation of the kidney volumes computed by each reader are summarized in Table 1.

Reader	Volume (mL) : Mean ± SD
Radiologist 1	140.6 ± 37.23
Radiologist 2	140.907 ± 37.222

Table 1. Descriptive statistics of 40 segmented kidney volumes

The inter-observer analysis between reader #1 and #2 is summarized on Table 2 below. These results indicate that inter-observer reproducibility for kidney volume measurement is excellent, ICC=0.998 (0.997-0.999) and show an absolute volume difference (in %of mean kidney volumes) of 1.16% ± 0.86. No statistically significant differences were observed (p=0.43) between reader #1 and #2 for the volume computation by our method.

Method	Comparison between reader #1 and #2			
	ICC (95%) CI (two-sided)	Volume diff (%) Mean ± SD	Abs volume diff (%) Mean ± SD	P value for Student T-test
3D CA-GPU	0.998 (0.997-0.999)	-0.21 ± 1.44	1.16 ± 0.86	0.43

Table 2. Statistical results for 40 renal volumes measurements

Measurement time: The whole segmentation time of the FBA-GPU method was subdivided as the initialization time needed to seeds positioning (1 min on average) and as the GPU computation time (3.6 s ± 0.9). This gave us a total estimated segmentation time lower then 1min 04s, on average, per kidney. For complicated MRA cases, more time was needed because it was necessary to add other seeds to ensure good segmentation results. However, the total time never exceeded 3 min.

Computational efficiency: In this section, we validated the hypothesis that our FBA-GPU algorithm was more efficient, in this segmentation context, than a CPU implementation of the method. To perform this comparative analysis we implemented the Dijkstra's all pair shortest path (APSP) algorithm in C++, using a binary heap. The algorithm was adapted to propagate the seed labels needed by the multi-label segmentation approach.

All CPU segmentation tests were performed automatically by using the previously saved labelled seeds, defined by the users during the first validation study.

The fundamental difference between the Dijkstra algorithm (DA) and FBA approach is that the DA implementation uses Priority Queue and defines a sorting function on the nodes. The algorithm converged to an optimal solution, since the search proceeded by expanding the lowest-cost vertices first, and optimal wave fronts, that worked their way out through the search space were generated ; an optimal decision at each step produced a globally optimal solution. In other words, the valid solution was only available once all vertices had been visited. In the FBA approach, each vertex was simply relaxed several times until the algorithm converged to the stationary global solution. This presents the advantage that, at each iteration, the solution is available and becomes closer to the optimal solution until the convergence is reached. The number of iterations needed to converge depends on the length of the longest geodesic path between a k-labelled vertex and all other vertices of the image or between two different k-labelled vertices. This means that the number of iterations needed to converge depends on the geometric complexity of the object to segment. To illustrate this we can say that the FBA approach needs many more iterations to segment an object like a maze than for an organ such as the kidney. Based on different segmentation contexts of human organs (liver, kidney, bones, aortic aneurysm …) we showed that the number of iterations, D, can be estimated by the equations (8) as a constant C that multiplies the maximal shortest Euclidian distance between the seed families defined inside (s) and outside (t) the organ respectively.

$$d_i = \min_{j=1}^{N}\left(\left\| s_i - t_j \right\|\right) \text{ and } D = C \max_{i=1}^{M}\left(d_i\right) \tag{8}$$

Where M and N represent the number of seeds having label s and t respectively.
For our segmentation application, a conservative value for D is given by C > 4.

GPU vs. CPU: The average of the DA-CPU computing time was 38.5s ± 8 while the corresponding FBA-GPU time for 100 iterations was 3.6s ± 0.9. The convergence rate of the FBA-GPU approach was evaluated by comparing the segmentation results obtained after a fixed number of iterations to the fully converged result given by the DA-CPU method. We show in figure 7 that after 10 iterations the difference between the intermediate FBA result and the ideal volume was 30.8% while after 50 iterations, this difference was drastically reduced to 0.12%. We confirmed that for all 40 kidney segmentations, no differences were observed after 60 iterations. This justifies that we fixed the number of iterations at 100 (conservative value) for all segmentation experiments presented in this paper. It should be noted that in the cases where the algorithm did not fully converge, a few iterations were added from this non converged state to reach the expected segmentation results.
GPU Benchmark: We were also interested in the evaluation of the performances of our FBA-GPU approach on different graphics hardware available in our imaging labs. To do this, the same version of the software was installed on seven PCs where we recorded GPU and CPU computation times needed to run the identical kidney segmentation experiment. Table 3 summarizes the measured computing times to run FBA on different graphics hardware and PCs. The GPU time is given separately as the setup time and as the FBA time. The setup-time is the time allocated for data transfer from/to the video memory and the FBA-time represents the time needed to perform 100 iterations of the Ford-Bellman algorithm.

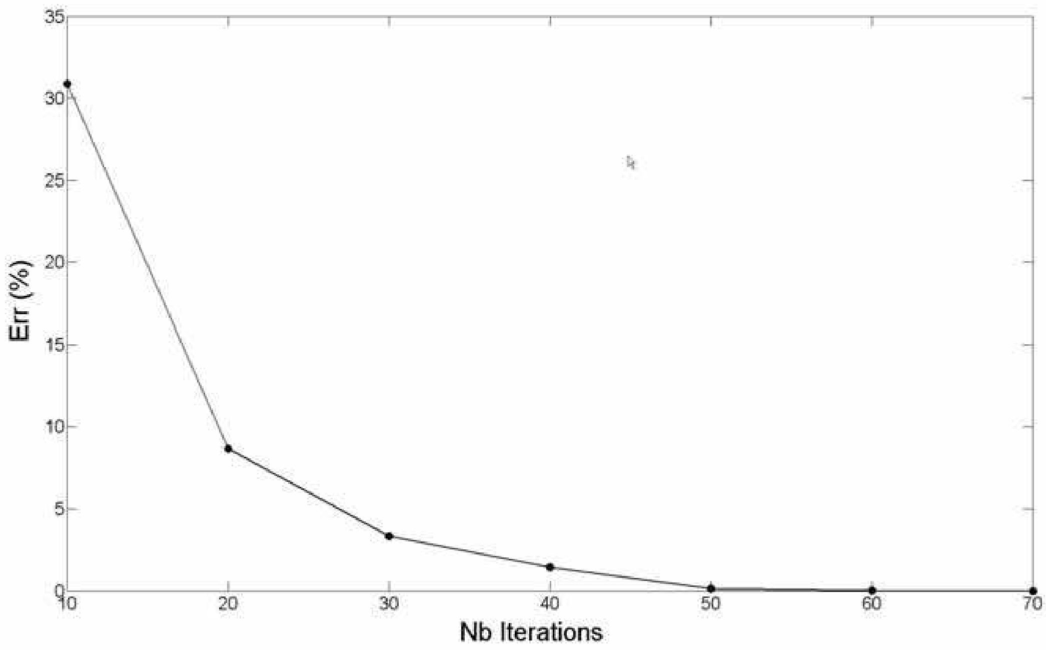

Fig. 7. Difference (Err) between the FBA segmented kidney volume and the converged solution at each 10 iterations. Err is represented in % of the converged volume.

	GPU time (ms)		
Graphic Card family	Setup-time	FBA-time	Speedup factor
ATI Radeon HD 4870	885	563	9,7
Nvidia GeForce GTX 260	812	922	5,9

ATI Radeon HD 3970	n/a	1211	4,5
NVIDIA GeForce 9800 GT	1032	1843	3,0
9800 gtx nvidia laptop	1061	1950	2,8
Radeon X1950 Pro	906	3047	1,8
NVIDIA GeForce 7950 GT	1419	5444	1,0

Table 3. Benchmarking of FBA running times (ms) on different graphics hardware. The speed up factor is given as the ratio of the FBA-time to the slowest FBA-time.

5. Discussion

The purpose of our work was to demonstrate that simple cellular automata algorithms implemented on low cost graphics hardware can perform efficient image processing tasks on N-D medical datasets. The implementation of the Ford-Bellman's algorithm was in a first time applied to perform the watershed transform, and in a second time, to perform seeded organ segmentation.

5.1 Watershed transform

A comparison between our GPU watershed implementation and an efficient CPU implementation of the watershed transform showed that a GPU approach based on a massively parallel implementation of the Ford-Bellman algorithm can outperform efficient CPU implementation of the watershed algorithm. Computing a watershed transform can be an ideal application case for the FBA-GPU approach because the algorithm is initialized by a large number of seeds which are defined as local minima in a specific neighborhood. As the local minimum are defined on the whole image dataset, only few iterations are needed to reach a converged result. However, the number of iterations must be increased in the case of ideal images such these having large plateaus filled with the same gray level, or with images smoothed by a large Gaussian kernel. The reason is that the local minimum are more distant from each other and more iterations are then needed to find the converged watershed lines. We showed that the FBA-GPU approach is particularly efficient over CPU implementation when it is applied to real noisy large images such as medical datasets. In other cases, an hybrid GPU-CPU approach can be considered.

5.2 Seeded segmentation

The second purpose of this study was to demonstrate that our FBA-GPU approach can efficiently be used to perform automatic organ segmentation in 3D medical datasets with a high degree of reliability and accuracy. The high reproducibility of renal volumes segmentation (inter-observer correlation ICC=0.998) combined to its accuracy (mean absolute error < 1.5%) makes this method valid for clinical use. The low difference between renal volumes segmented by two independent readers also indicates that our FBA method is robust to manual seed selection. The maximal absolute error between the two readers was 3.12%, which indicates that the precision of renal volume measurements is weakly affected by the variability of image parameters, such as anisotropic resolution of MRI scans and poor contrast between kidney parenchyma and background in the presence of severe

renovascular disease as illustrated on figure 1b. Moreover, the short time required by the whole segmentation process (< 2 min) shows that our method can be used in clinical routine.
In a second time we conduct a study to evaluate the computational efficiency of our GPU Ford-Bellman approach compared to a CPU implementation of the Dijkstra's algorithm. For all kidney segmentation (100 iterations) we see a speedup of 10 between the GPU and CPU running time. This speedup factor can easily be increased to 50 by using a more actual graphic hardware as presented in the next GPU benchmark section. Our GPU-based version of the FBA algorithm is simple to implement and presents itself as an alternative and optimized approach to perform graph-based segmentation in regards to other proposed approaches (Vineet and Narayanan, 2008, Bolz et al., 2003, Fung and Mann, 2008, Gernot et al., 2007, Koutis, 2008).
Our work was voluntarily designed as a compromise between computational efficiency and hardware compatibility. Indeed, our FBA approach was implemented using OpenGL Cg language on different brands of low cost graphics cards.
We also benchmarked the FBA on seven different graphics hardware (Table 3), available on standard PCs in our medical imaging department, in order to highlight (non exhaustive) increasing performances and speedups of GPUs between past and newer low cost hardware generations. On one side, by comparing the FBA-time on different GPUs, a speedup factor of 10 was found between the slowest and fastest tested graphics hardware and a speedup greater than 5 between the GPU used in this study and the ATI Radeon HD 4870., The setup-time was overall in the same range for all graphics hardware. On the other side, the CPU computing times were slightly the same for all PCs available in our labs.

5.3 Are GPUs better than CPUs ?

The GPU's rapid increase in both programmability and capability has spawned a research community that has successfully mapped a broad range of computationally demanding, complex problems to the GPU (Owens et al., 2008). While GPUs are a compelling alternative to traditional microprocessors in high-performance computer systems, they can also be seen as a complementary solution to CPU approaches. In our case, using the FBA-GPU approach to compute the shortest path inside a long and tortuous organ, such as vessels or the colon, gives poor performances compared to the DA-CPU approach. Yet, we show that in our study the FBA method is more efficient to perform multi label segmentation of the kidney. For this reason, we suggest that performances comparison between GPU and CPU approaches must be carefully regarded from a global point of view including the application context and the complexity of the algorithms to be implemented.
However, there are some limitations on the use of GPUs. The first limitation concerns the setup-time which represents the minimal time needed for the data transfer from/to the video memory. Since the video memory is accessed through the PCI Express lane it is much slower than the RAM access. It seems clear that the best use case is achieved by loading once the data to the graphics card's memory and to compute as much as possible there. For this reason it is not efficient to test the convergence of the FBA at each iteration or after a low number of iterations. A second limitation concerns the restriction on the type of data that can be used. Indeed, there is no native double precision float in GPU (this could change in a near future) and this limitation can be so important in some computational problems that straightforward usage of GPU is excluded.

6. Conclusion

We have presented a GPU-based Cellular Automaton to perform efficient image processing tasks on large image datasets. Our work is based on the Ford-Bellman's shortest paths algorithm which is first applied to compute the watershed transform and secondly to perform automatic multi label segmentation of organs in N-D medical images with minimal user interaction for initialization. Validation of this method on MRA examinations showed high inter-observer reproducibility and accuracy that allows the method to be used in clinical routine. Our implementation of the FBA in the form of a CA is simple, efficient and straightforward, and can be implemented in low cost vendor-independent graphics cards. Our work was strongly motivated by the fact that the processing power has clearly shifted from the CPU to the GPU. The experimental testing performed on MRA datasets confirms the expected gain in performance with GPU implementation. To our knowledge, we are the first to propose a GPU implementation of FBA as a cellular automaton to perform the watershed transform and seeded segmentation on ND images.

7. References

Aharon, S., Grady, L. & Schiwietz, T. (2005) GPU accelerated isoperimetric algorithm for image segmentation, digital photo and video editing. Google Patents.

Alonso Atienza, F., Requena Carrión, J., García Alberola, A., Rojo Álvarez, J. L., SÁnchez Muñoz, J. J., Martínez SÁnchez, J. & Valdés Chávarri, M. (2005) A Probabilistic Model of Cardiac Electrical Activity Based on a Cellular Automata System. *Revista Española de Cardiología (Internet),* **58,** 41-47.

Bai, X. & Sapiro, G. (2007) A Geodesic Framework for Fast Interactive Image and Video Segmentation and Matting. pp. 1-8.

Bellman, R. (1956) *ON A ROUTING PROBLEM,* Defense Technical Information Center.

Beucher, S. & Lantuejoul, C. (1979) Use of watersheds in contour detection. In: *International Workshop on Image Processing*. Vol. 17, pp. 2.1–2.12.

Bolz, J., Farmer, I., Grinspun, E. & Schröoder, P. (2003) Sparse matrix solvers on the GPU: conjugate gradients and multigrid. pp. 917-924. ACM New York, NY, USA.

Boykov, Y. & Funka-Lea, G. (2006) Graph Cuts and Efficient NDImage Segmentation. *International Journal of Computer Vision,* **70,** 109-131.

Boykov, Y. & Jolly, M. P. (2000) Interactive Organ Segmentation Using Graph Cuts. *LECTURE NOTES IN COMPUTER SCIENCE*, 276-286.

Chefd'hotel, C. & Sebbane, A. (2007) Random Walk and Front Propagation on Watershed Adjacency Graphs for Multilabel Image Segmentation. pp. 1-7.

Digabel, H. & Lantuejoul, C. (1977) Iterative algorithms. JL Chermant Eds. In: *Actes du Second Symposium Europeen d'Analyse Quantitative des Microstructures en Sciences des Materiaux, Biologie et Medecine.* Riederer Verlag, Caen, France.

Dijkstra, E. W. (1959) A note on two problems in connexion with graphs. *Numerische Mathematik,* **1,** 269-271.

Dixit, N., Keriven, R., Paragios, N. & Graphique, P. (2005) GPU-Cuts: Combinatorial Optimisation, Graphic Processing Units and Adaptive Object Extraction GPU-Cuts: Segmentation d'Objects.

Eom, S., Shin, V. & Ahn, B. (2007) Cellular Watersheds: A Parallel Implementation of the Watershed Transform on the CNN Universal Machine. *IEICE TRANSACTIONS on Information and Systems,* **90,** 791-794.

Even, S. (1979) Graph Algorithms. Rockville. *MD: Computer Science Press,* **249**.

Fairfield, J. (1990) Toboggan contrast enhancement for contrast segmentation. Vol. 1.

Falcão, A. X., Stolfi, J. & de Alencar Lotufo, R. (2004) The Image Foresting Transform: Theory, Algorithms, and Applications. *IEEE TRANSACTIONS ON PATTERN ANALYSIS AND MACHINE INTELLIGENCE*, 19-29.

Ford Jr, L. R. (1956) NETWORK FLOW THEORY.

Fung, J. & Mann, S. (2008) Using graphics devices in reverse: GPU-based Image Processing and Computer Vision. pp. 9-12.

Ganguly, N., Sikdar, B. K., Deutsch, A., Canright, G. & Chaudhuri, P. P. (2003) A survey on cellular automata. *Dresden University of Technology, Technical Report Centre for High Performance Computing*.

Gardner, M. (1970) Mathematical Games: The Fantastic Combinations of John Conway's New Solitaire Game 'Life'. *Scientific American,* **223,** 120-123.

Gernot, Z., Christian, T., Ivo, I., Marcus, M., Art, T. & Hans-Peter, S. (2007) GPU-based light wavefront simulation for real-time refractive object rendering. In: *ACM SIGGRAPH 2007 sketches.* ACM, San Diego, California.

Gobron, S., Devillard, F. & Heit, B. (2007) Retina simulation using cellular automata and GPU programming. *Machine Vision and Applications,* **18,** 331-342.

Grady, L. & Funka-Lea, G. (2004) Multi-label Image Segmentation for Medical Applications Based on Graph-Theoretic Electrical Potentials. *LECTURE NOTES IN COMPUTER SCIENCE*, 230-245.

Kauffmann, C. & Piche, N. (2008) Cellular automaton for ultra-fast watershed transform on GPU. In: *ICPR 2008.* pp. 1-4. Tampa bay, FL, USA.

Konushin, V., Vezhnevets, V. & Moscow, R. (2006) Interactive Image Colorization and Recoloring based on Coupled Map Lattices. pp. 231–234.

Koutis, I. (2008) Faster Algebraic Algorithms for Path and Packing Problems. In: *Proceedings of the 35th international colloquium on Automata, Languages and Programming, Part I.* Springer-Verlag, Reykjavik, Iceland.

Lin, Y. C., Tsai, Y. P., Hung, Y. P. & Shih, Z. C. (2006) Comparison between immersion-based and toboggan-based watershed image segmentation. *IEEE TRANSACTIONS ON IMAGE PROCESSING,* 15, 632-640.

Meyer, F. (1994) Topographic distance and watershed lines. *Signal Processing,* **38,** 113-125.

Mortensen, E. N. & Barrett, W. A. (1998) Interactive Segmentation with Intelligent Scissors. *Graphical Models and Image Processing,* **60,** 349-384.

N. Moga, A., Cramariuc, B. & Gabbouj, M. (1998) Parallel watershed transformation algorithms for image segmentation. *Parallel Computing,* **24,** 1981-2001.

Nepomniaschaya, A. S. (2001) An Associative Version of the Bellman-Ford Algorithm for Finding the Shortest Paths in Directed Graphs. *LECTURE NOTES IN COMPUTER SCIENCE*, 285-292.

Noguet, D. (1997) A massively parallel implementation of the watershed based on cellular automata. In: *IEEE International Conference on Application-Specific Systems, Architectures and Processors.* pp. 791-794.

Owens, J. D., Houston, M., Luebke, D., Green, S., Stone, J. E. & Phillips, J. C. (2008) GPU Computing. In: *Proceedings of the IEEE, 96(5), May.*

Owens, J. D., Luebke, D., Govindaraju, N., Harris, M., Kruger, J., Lefohn, A. E. & Purcell, T. J. (2007) A survey of general-purpose computation on graphics hardware. Vol. 26, pp. 80-113. Blackwell Publishing Ltd.

Protiere, A. & Sapiro, G. (2007) Interactive Image Segmentation via Adaptive Weighted Distances. *IEEE TRANSACTIONS ON IMAGE PROCESSING,* **16,** 1046.

Qu, Y., Wong, T. T. & Heng, P. A. (2006) Manga colorization. *Proceedings of ACM SIGGRAPH 2006,* **25,** 1214-1220.

Roerdink, J. & Meijster, A. (2000) The watershed transform: Definitions, algorithms and parallelization strategies. *Mathematical Morphology,* **41,** 187-S128.

Rother, C., Kolmogorov, V. & Blake, A. (2004) " GrabCut": interactive foreground extraction using iterated graph cuts. *ACM Transactions on Graphics (TOG),* **23,** 309-314.

Sinop, A. K. & Grady, L. (2007) A Seeded Image Segmentation Framework Unifying Graph Cuts And Random Walker Which Yields A New Algorithm. pp. 1-8.

Vezhnevets, V., Konouchine, V. & Moscow, R. (2005) “GrowCut”-Interactive Multi-Label NDImage Segmentation By Cellular Automata. pp. 150–156.

Vincent, L. & Soille, P. (1991) Watersheds in digital spaces: an efficient algorithm based onimmersion simulations. *IEEE TRANSACTIONS ON PATTERN ANALYSIS AND MACHINE INTELLIGENCE,* **13,** 583-598.

Vineet, V. & Narayanan, P. J. (2008) CUDA cuts: Fast graph cuts on the GPU. pp. 1-8.

Volkov, V. & Demmel, J. Using GPUs to Accelerate the Bisection Algorithm for Finding Eigenvalues of Symmetric Tridiagonal Matrices.

Von Neumann, J. & Burks, A. W. (1966) *Theory of self-reproducing automata,* University of Illinois Press Urbana.

Wolfram, S. (2002) A new kind of science. . *Champaign, IL: Wolfram Media.*

Xu, N., Ahuja, N. & Bansal, R. (2007) Object segmentation using graph cuts based active contours. *Computer Vision and Image Understanding,* **107,** 210-224.

Yatziv, L., Bartesaghi, A. & Sapiro, G. (2006) O (N) implementation of the fast marching algorithm. *Journal of Computational Physics,* **212,** 393-399.

Yin, L., Jian, S., Chi-Keung, T. & Heung-Yeung, S. (2004) Lazy snapping. In: *ACM SIGGRAPH 2004 Papers.* ACM, Los Angeles, California.

Zhao, Y. (2008) Lattice Boltzmann based PDE solver on the GPU. *The Visual Computer,* **24,** 323-333.

Pattern Recognition
Edited by Peng-Yeng Yin

ISBN 978-953-307-014-8
Hard cover, 568 pages
Publisher InTech
Published online 01, October, 2009
Published in print edition October, 2009

For more than 40 years, pattern recognition approaches are continuingly improving and have been used in an increasing number of areas with great success. This book discloses recent advances and new ideas in approaches and applications for pattern recognition. The 30 chapters selected in this book cover the major topics in pattern recognition. These chapters propose state-of-the-art approaches and cutting-edge research results. I could not thank enough to the contributions of the authors. This book would not have been possible without their support.