EURASIP Journal on Advances in Signal Processing
a SpringerOpen Journal

RESEARCH Open Access

Initial condition for efficient mapping of level set algorithms on many-core architectures

Gábor János Tornai* and György Cserey

Abstract

In this paper, we investigated the effect of adding more small curves to the initial condition which determines the required number of iterations of a fast level set (LS) evolution. As a result, we discovered two new theorems and developed a proof on the worst case of the required number of iterations. Furthermore, we found that these kinds of initial conditions fit well to many-core architectures. To show this, we have included two case studies which are presented on different platforms. One runs on a graphical processing unit (GPU) and the other is executed on a cellular nonlinear network universal machine (CNN-UM). With the new initial conditions, the steady-state solutions of the LS are reached in less than eight iterations depending on the granularity of the initial condition. These dense iterations can be calculated very quickly on many-core platforms according to the two case studies. In the case of the proposed dense initial condition on GPU, there is a significant speedup compared to the sparse initial condition in all cases since our dense initial condition together with the algorithm utilizes the properties of the underlying architecture. Therefore, greater performance gain can be achieved (up to 18 times speedup compared to the sparse initial condition on GPU). Additionally, we have validated our concept against numerically approximated LS evolution of standard flows (mean curvature, Chan-Vese, geodesic active regions). The dice indexes between the fast LS evolutions and the evolutions of the numerically approximated partial differential equations are in the range of 0.99 ± 0.003.

Introduction

The use of level-set (LS)-based curve evolution has become an interesting research topic due to its versatility and accuracy. These flows are widely used in various fields like computational geometry, fluid mechanics, image processing, computer vision, and materials science [1]. In general, the method entails that one evolves a curve, surface, or image with a partial differential equation (PDE) and obtains the result at a point in the evolution. There is a subset of problems where only the steady state of the LS evolution is of practical interest like segmentation and detection. In this article, only this subset is considered. In addition, we do not form any operator or speedfield (F) for driving the evolution of the LSs. However, two theoretical worst-case bounds of the required number of iterations are proposed to reach the steady state for a well-defined class of LS-based evolution. These bounds depend only on the initial condition. Furthermore, the bounds only allow an extremely small number of iterations if the evolution is calculated with a properly chosen initial condition. These kinds of evolutions are calculated very quickly on many-core devices.

The subject of this paper is both theoretical and practical. The theoretical side is clearly the two new theorems on the worst case of the required number of iterations of the Shi LS evolution [2]. This evolution omits the numerical solution of the underlying PDE and successfully approximates it with a rule-based evolution. It is based on the sign of the driving forces (F) normal to the curves to be changed. The first theorem gives a general bound, and the second one assumes a special kind of discrete convexity defined in subsection 'Basic definitions'. The practical side is presented through two case studies, namely, the LS evolution of Shi is mapped in a straightforward way on two completely different many-core architectures. With a lot of small curves in the initial condition, which would be infeasible on a conventional single-core processor, the proposed theorems ensure small number

*Correspondence: tornai.gabor@itk.ppke.hu
Faculty of Information Technology and Bionics, Pázmány Péter Catholic University, Práter str. 50/a, Budapest 1083, Hungary

of iterations. Additionally, with the change in the initial condition (instead of one curve, a lot of small curves are used), the computing width of the many-core platform are utilized.

The first successful method to speed up the LS evolution was introduced in [3]. A local method was proposed in [4] with better characteristics. These methods are labeled as narrow banding methods. However, we are not presenting any PDE operators and do not design any speedfield. Instead, we direct the reader to the work of Sapiro [5] who gives a detailed picture from the art of PDE operator design for a given purpose. Furthermore, there are several results [6-9] regarding region- and model-based evolutions. In [10] the authors used Sobolev norm instead of the standard inner-product-based L_2 norm and showed that this norm allows new energies to implement otherwise considered infeasible.

There are multiple results reporting successful mapping of LS evolution to many-core platforms. An interactive 3D solver for GPU is presented in [11]. This realization uses a graphics application programming interfaces (API) (OpenGL) and the rendering pipeline. Two later works [12,13] applied the computing-unified device architecture (CUDA) of NVIDIA (NVIDIA Corporation, Santa Clara, CA, USA). Both papers worked with 3D volumes. In [12] the authors mapped a sparse solver, while in [13] a higher-order scheme was used to evolve the LSs. Cellular neural networks (CNN) [14] proved to be an inspiring construct. There have been results regarding the mapping of LS-like evolutions to CNN [15-17]. Cserey et al. [15] successfully mapped the nonlinear histogram modification to CNN. A later work [17] realized an online boundary detection algorithm based on LS curve evolution to extract the volume of the right atrium. These papers and results indicate that various LS evolutions can be mapped and used on different many-core platforms. In this paper, not a new numerical variant or many core, parallel implementation of the narrow band algorithm is presented. Instead, we are focusing on the given type of evolution on many-core devices and for this evolution, we give two theorems upperbounding the required number of iterations of the evolution process.

This paper is organized as follows. The next section summarizes the theory for curve evolution realized as LS motion, then some definitions follow and the two theorems on worst-case bound of the required number of iterations with their proofs close the theoretical part. Section 'Many-core hardware platforms' describes the hardware platforms used in the two case studies. Section 'Experiments' presents the experimental results. In section 'Validation,' we validate our method against three numerically solved PDE of LS flows. It is followed by section 'Discussion' in and the last section concludes our paper.

Theory

Basic curve evolution

Let us consider a family of closed curves $\gamma(s,t)$ (or surfaces) in $\mathbf{R}^k$, $(k = 2,3)$ where s parameterizes the curve and t the (artificial) time. Our aim is to trace $\gamma(s)$, which moves normal to itself, with a given speed function F. If there are no restrictions on the sign of F, then the motion of γ can be complex. Furthermore, it is hardly trackable by curve-based (Lagrangian) methods since these methods cannot handle topological changes and become unstable near singularities. However, the initial curve $\gamma(s, t = 0)$ can be embedded as the zero LS of a higher dimensional function so the evolution of this function is linked to the propagation of the front through the time evolution of this initial value problem. This family of methods is called level set methods (LSM). The evolution equation of these methods is as follows:

$$\frac{\partial \phi}{\partial t} = F|\nabla \phi| \tag{1}$$

γ is embedded into a function that is called LS function. It is denoted by ϕ. The uniformly sampled domain of ϕ is denoted by D and a point $\mathbf{x} \in D$ is characterized by its coordinates $(\mathbf{x} = (x_1, ..x_k))$. The region enclosed by γ is referred to as the object region and is denoted by Ω. Similarly, the region outside γ is referred to as the background region and is denoted by Γ. Let Ω^* and Γ^* denote the true object region and true background region, respectively. ϕ is chosen to be negative inside Ω and positive in Γ. The zero LS is represented implicitly by ϕ. The LS evolution process of [2] is used throughout in this work. In the neighborhood of the zero LS, two sets are defined uniquely with respect to the selected discrete neighborhood:

$$L_{\text{in}} = \{\mathbf{x} | \phi(\mathbf{x}) < 0 \text{ and } \exists \mathbf{y} \in N(\mathbf{x}) \text{ that } \phi(\mathbf{y}) > 0\} \tag{2}$$

$$L_{\text{out}} = \{\mathbf{x} | \phi(\mathbf{x}) > 0 \text{ and } \exists \mathbf{y} \in N(\mathbf{x}) \text{ that } \phi(\mathbf{y}) < 0\} \tag{3}$$

where $N(\mathbf{x})$ is the selected neighborhood defined by Equation 4, but can be any discrete neighborhood known from discrete topology.

$$N(\mathbf{x}) = \{\mathbf{y} \in D | \sum_{k=1}^{K} |y_k - x_k| = 1\} \ \forall \mathbf{x} \in D\} \tag{4}$$

The sets L_{in} and L_{out} are referred to as active front, and ϕ is defined as an approximated signed distance function:

$$\phi(x) = \begin{cases} -3, \text{ if } \mathbf{x} \in \Omega, \text{ and } \mathbf{x} \notin L_{\text{in}} \text{ inner points} \\ -1, \text{ if } \mathbf{x} \in L_{\text{in}} \\ \ \ 1, \text{ if } \mathbf{x} \in L_{\text{out}} \\ \ \ 3, \text{ if } \mathbf{x} \in \Gamma, \text{ and } \mathbf{x} \notin L_{\text{out}} \text{ outer points} \end{cases} \tag{5}$$

The LSM of Shi uses only the sign of the speed function F to determine the motion of the active front at a given point. The speed function itself is chosen according to the requirements and can be arbitrary (Chan-Vese,

geodesic active contour, geodesic active region (GAR) - gaussian mixture model (GMM), nonparametric mixture model, etc.). It is (re)calculated in each cycle for the active front, but its sign at every point of the active front determines the action on these points. Furthermore, from an implementation point of view, both sets of the active front are realized as two separate lists. The motion of the active front is realized by two local switch operators, one for outward and one for inward motion. By applying *switch_in()* to any points in L_{out} having positive sign F at that point, the active front is moved outward one grid point from that location. Namely, the point is moved from L_{out} to L_{in}, its exterior neighbors are put to L_{out}, and its interior neighbors are deleted from L_{in}. The *switch_out()* procedure operates similarly in the other direction.

One iteration of the evolution of the algorithm performs four scans. Firstly, it scans L_{out} for *switch_in()* then it scans L_{in} for elements that are not in the active front any more (all neighbors reside in Ω), afterwards, L_{in} is scanned for *switch_out()*. Lastly, L_{out} is scanned for elements that are not in the active front any more. Basically, the algorithm makes one step inward or outward according to the sign of the speed function. If it contains a curvature-dependent smoothing term, then sharpened gaussian filtering (shock-filtered heat diffusion) can be applied just on the active front itself in a different cycle after a given number of evolution cycles. For further details see [2,18,19].

Basic definitions

A path p between $\mathbf{x}$ and $\mathbf{y}$ is a sequence of points $\mathbf{x}_l (l = 0, 1, \ldots, L) \in D$ subject to $\mathbf{x}_l \in N(\mathbf{x}_{l+1})$ and $\mathbf{x} = \mathbf{x}_0$ and $\mathbf{y} = \mathbf{x}_L$. A set of points $\mathcal{A}$ forms a connected region if and only if there exists a path p between every $\mathbf{x}, \mathbf{y} \in \mathcal{A}$ subject to $\forall \mathbf{x}_l \in p$ is an element of $\mathcal{A}$.

The length of a path is a non-negative integer (L) and $L = |p| - 1$, where $|.|$ denotes the number of points in the path. A minimum path $p_{\min}$ is a shortest path meaning there are no shorter p' paths between $\mathbf{x}$ and $\mathbf{y}$. Minimum path is usually not unique and can depend on the chosen discrete neighborhood. The distance between $\mathbf{x}$ and $\mathbf{y}$ is a non-negative integer that is exactly the length of a minimum path between the two points. This is a real metric and is going to be referred to as d_d.

Within a connected region $\mathcal{A}$, a minimal path p between $\mathbf{x}$ and $\mathbf{y}$ is minimal if and only if $\mathcal{A} \cap p = p$ and there are no shorter p' paths within $\mathcal{A}$ between $\mathbf{x}$ and $\mathbf{y}$. Like the minimum path, the minimal path may not be unique and may depend on the chosen neighborhood. The diameter B of a connected region is the longest minimum path having at least its endpoints within the connected region. A connected region is considered as convex if all minimal paths are minimum paths at the same time. A configuration $C = \{D \times \phi\}$ is the actual state of the LS function, namely, the shape of the zero LS and the connected regions (Ω_p, Γ_q) composing the object and the background region.

Now we have all the necessary tools to establish proper worst-case bounds on the number of iterations required by the Shi LSM to converge.

Theoretical results: worst-case bounds

Theorem 1 (General bound)**.** *Let the true object region Ω^* be composed of P-connected regions Ω_p^* (where $p = 1 \ldots P$) and the true background region Γ^* be composed of q connected regions Γ_q^* (where $q = 1 \ldots Q$). Assume that $F > 0$ in Ω^* and $F < 0$ in Γ^*. At initialization, C is chosen such that $\Omega = \cup_i \Omega_i$, $\Gamma = \cup_j \Gamma_j$ and $\Omega_p^* \cap \Omega \neq \emptyset$, $\forall p = 1 \ldots P$ and $(D \setminus \Omega) \cap \Gamma_q^* \neq \emptyset$, $\forall q = 1 \ldots Q$. Then, the Shi LSM converges to Ω^* in $N_{it} \leq \max(\max_i(|\Omega_i|), \max_j(|\Gamma_j|))$ iterations, where $|.|$ denotes the number of elements in the region.*

Theorem 2 (Convex bound)**.** *If either Ω^* or Γ^* is convex, then the Shi LSM converges to Ω^* in $N_{it} \leq \max(\max_i(B_{\Omega_i}), \max_j(B_{\Gamma_j}))$ iterations, where B denotes the diameter of the given region.*

Proof of Theorem 1 on general bound. Let $\Omega^a = \Omega^* \cap \Omega = \bigcup_{p=1}^{P} \Omega_p^a$. These are fixed sets and will not change during the evolution process. Furthermore, $F(\mathbf{x_k}) > 0$, $\forall \mathbf{x_k} \in \Omega^a$ which ensures that $\Omega^a \subseteq \Omega$ as Ω evolves.

At initialization for each Ω_i two cases are possible. First case: $\Omega_i \cap \Omega^* = \emptyset$. Therefore, $\Omega_i \subseteq \Gamma^*$ so, $F(\mathbf{x}) < 0$. On the boundary of Ω_i, $L_{\text{in},i}$, a *switch_out* operation is applied so the diameter of Ω_i becomes smaller with two in every iteration. Second case: $\Omega_i \cap \Omega^* \neq \emptyset$. Therefore, the longest possible path in Ω_i gives the upperbound of the number of iterations that is obviously upperbounded by the number of points in Ω_i. Following similar arguments, we can also show this for Γ_j. Taking the maximum of the upperbounds completes the proof of Theorem 1. □

Proof of Theorem 2 on convex bound. Obviously, the first case of the proof of Theorem 1 obeys the desired bound. The second case is as follows. Since Ω^* is convex, the length of the longest path is bounded by the diameter of Ω_i. In worst case, $\Omega_i \cap \Omega^*$ is one of the endpoints of the diameter. Following similar arguments, we can also show this for Γ_j. Taking the maximum of the diameters in each initial and background region completes the proof of Theorem 2. □

Theorem 1 gives a general upperbound on N_{it}, and the iteration cycle checking the stopping condition is not necessary if the number of iteration has reached this upper bound. This worst-case bound is approached if Ω^* or Γ^* are degenerated in some sense (see Figure 1D and Table 1 for example). However, in many cases the stricter bound can be applied. We shall emphasize that

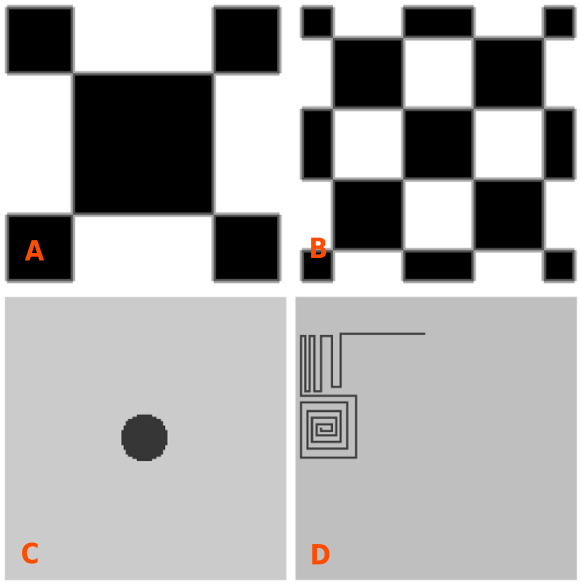

Figure 1 Example of initial condition and test objects. (A and **B)** Initial conditions belonging to $n = 1, 2$. This chessboard-like initial pattern partitions the domain to several convex pieces. It ensures the theoretical bounds to decrease as n increases. **(C)** Convex test object. **(D)** Concave test object. Required number of iterations are presented for **(C)** and **(D)** in Table 1.

the worst-case bound is a quantity derived from the initial condition.

The possibility of choosing the initial shape of the regions Ω_i and Γ_j is essential to minimize the required number of iterations. It shall be noted that according to the Shi LSM, all calculations are done in the active front that have direct connection with the initial shape of the aforementioned regions. Making both Ω_i and Γ_q smaller, the smaller the worst-case bounds can be. This statement leads us to section 'Experiments', namely, how to construct initial conditions that are minimal in the sense of worst-case bounds and can be mapped and processed efficiently on a many-core architecture. It should be noted that the presentation above does not depend on the dimensionality of the data so the theorems are general from this point of view and the dimension can be arbitrary.

Many-core hardware platforms

CNN universal machine

The CNN universal machine (CNN-UM) [20] is a virtual machine construct like the Turing machine. It is Turing complete and it is universal in the sense that a CNN-UM can present all the behaviors that a predefined CNN dynamics can show. A CNN consists of nonlinear dynamical systems called cells. Each cell has a state, an input, and an output that is a nonlinear function of the state. The cells are arranged in a rectangular grid and each cell is connected to its (nearest) neighbors within a given range (N_r). The connections are weighted and these weights are called templates. The templates define the operator that is applied on the input and the state, hence, they define the output dynamics.

$$\frac{d}{dt}x_{ij}(t) = -x_{ij}(t) + \sum_{kl \in N_r} A_{kl,ij}y_{kl}(t) + \sum_{kl \in N_r} B_{kl,ij}u_{kl}(t) + z_{ij} \tag{6}$$

Here x_{ij}, u_{ij}, y_{ij} stand for the state, input, and output of the cell ij, respectively; A, B, z are the template parameters: A for feedback, B for input, and z_{ij} for bias. The CNN-UM, in addition to the standard CNN, contains memories and control units to allow performing series of template operations and branching.

The experiments were done on an Eye-RIS v1.3 vision system (VS) (Anafocus Ltd., Seville, Spain). It consists of a Q-Eye, Altera NIOS-II 32-b RISC microprocessor and on chip RAM. The Q-Eye is a quarter common intermediate format (QCIF) monochrome image sensor processor with 7- to 8-b *de facto* accuracy. It is a fine-grain CNN-UM implementation with nearest neighborhood capable operations. The microprocessor handles the memory, the I/O ports and organizes the execution. The consumption of the complete VS is below 750 mW.

GPU

Recent GPUs are feasible for nongraphic operations as well and programmable through general purpose APIs like C for CUDA [21] or OpenCL [22]. In this paper, OpenCL nomenclature is used. The description below is a brief

Table 1 Experimental validation of the theorems

	Values							
Number of circles (n)	1	2^2	4^2	8^2	16^2	24^2	32^2	64^2
Bound according to Theorem 1	64^2	32^2	256	64	16	9	4	1
Bound according to Theorem 2	127	63	31	15	7	5	3	1
Circle N_{it}	26	16	9	6	4	3	3	1
Degenerate N_{it}	145	68	18	7	6	3	3	1

The image was 128^2 pixels. Configuration C was set as squares arranged into n rows and n columns in a chessboard-like pattern (see Figure 1A,B). Two different objects were tested: a circle in the center with radius 11 pixels and a snake-like degenerate object. Configuration and objects are presented in Figure 1.

overview of GPUs and, in addition to the basics, gives only those details that have great influence on the LS evolution.

A function that can be executed on the GPU is called a kernel. Any call to a kernel must specify an NDRange for that call. This defines not only the number of work-items to be launched, but also the arrangement of groups of work-items to work-groups and work-groups to the NDRange. The dimensionality of a work-group can be one, two, or three.

Physically, the elementary computing element is the computing element. A few computing element together with a given amount of SDRAM, scheduling unit, and special function unit forms a computing unit (CU). A device consists of several CUs and a global memory (off-chip).

The experiments were done on an NVIDIA 780 GTX GPU. It has 12 CUs, 192 computing elements, and 48KB shared memory in each CU, and 3 GB global memory. The hosting PC runs on Intel core i7-2600 CPU @3.4 GHz with 8 GB system memory, the operating system is Debian with Linux kernel the GPU driver version is 325.15.

Experiments

Theorems 1 and 2 give upperbounds on the required number of iterations (N_{it}). A practical proposal of this paper is to construct configurations that have as low worst-case bounds on N_{it} as feasible and can be computed efficiently on many-core architectures. This scenario is presented through two case studies. The first one is on an Eye-RIS v1.3 VS that is a hardware implementation of the CNN-UM and the second one is on a GPU.

The whole image is covered with many-many small active fronts, and as a consequence, the intersection condition ($\Omega_p^* \cap \Omega \neq \emptyset$) is automatically fulfilled. During the case studies, the speedfield F has been very simple, $+1$ for the object region and -1 for the background regions.

A case study on CNN-UM

In subsection 'CNN universal machine,' a short overview was given on the CNN-UM. Now the details of the mapped algorithm are described. The perspective in this scenario is the precedence of locality which becomes increasingly important as the technology feature size decreases, and the delay together with power consumption of global communication increases.

The mapped algorithm is based on the set theoretic description of the LS function. In addition to L_{in} and L_{out} two other sets are defined.

$$F_{\text{in}} = \{\mathbf{x} \in D | \, \phi(\mathbf{x}) < 0 \wedge \mathbf{x} \notin L_{\text{in}}\} \tag{7}$$

$$F_{\text{out}} = \{\mathbf{x} \in D | \, \phi(\mathbf{x}) > 0 \wedge \mathbf{x} \notin L_{\text{out}}\} \tag{8}$$

In other words, the neighbors of each point in these sets are in the same region as the point itself. Figure 2 shows the universal machine on flows (UMF) diagram of the algorithm. It is a special flowchart description of the algorithms executable on CNN-UM so that it is complete and unique [20].

Templates AND, OR denote elementary logic, AND-NOT performs logic subtraction ($Op1 \wedge \neg Op2$), DIL4 and ERODE4 are the four connected dilatation and erosion (spatial logic). All templates are of the nearest neighbor kind. In the 'Update L_{out}' phase, *foutmask* is computed first. It contains the points that are going to move outward. *foutmask* is used in three different ways. It is subtracted (ANDNOT) from L_{out}, added (OR) to L_{in} and dilated (DIL4, ANDNOT, AND) to generate its own outer neighbors. This is the new stepped L_{out} part and the unchanged parts are added with an OR operation. The resulting set is finalized as the new L_{out} (black rectangle in Figure 2). From the old F_{out} the new L_{out} is subtracted (ANDNOT) to get the new F_{out} (again, black rectangle in Update L_{out} phase). Finally, the modified L_{in} is added to F_{in}. In the 'Clean L_{in}' phase, the merged *foutmask*, L_{in}, and F_{in} is the only input. The new L_{in} is the outer pixel layer of this merged input. The new F_{in} is obtained by a simple four connected erosion while L_{in} is the result of a subtraction. 'Update L_{in}' and 'Clean L_{out}' are nearly identical, only the input of the operations are switched, and another mask is used (*finmask*).

The algorithm is implemented on an Eye-RIS 1.3 VSoC. One step of the algorithm is performed in 400 to 440 μs on a QCIF image. It must be noted that the actual computing is finished within 60 to 70 μs and the remaining time (340 to 370 μs) is required for the data movement from the main memory of the Eye-RIS (on the Altea NIOS-II microprocessor) to the Q-Eye chip memory.

A case study on GPU

The evolution process is divided into two steps. The first one is the planner step and the second is the evolution step. The planner creates the so-called plan. It contains the position offsets of the 16×16 tiles that are calculated actually in the iteration step. The planner works on the indicator image. The indicator is a tiny image and each pixel of the indicator is *true* if the corresponding tile on the input image shall be processed in this iteration and *false* otherwise. The size of the plan is calculated by local prefix-sum work-group wise, and global atomic addition is used to correctly determine the offset of the work-group within the plan. The source of the planner kernel is provided as Additional file 1.

The evolution kernel processes only those tiles of the LS function that are inserted in the plan. The evolution kernel makes a step either inward or outward direction depending on the sign of the force field on the LS function. This is done simultaneously unlike in the sequential algorithm. Each work-group processes a 16×16 tile provided

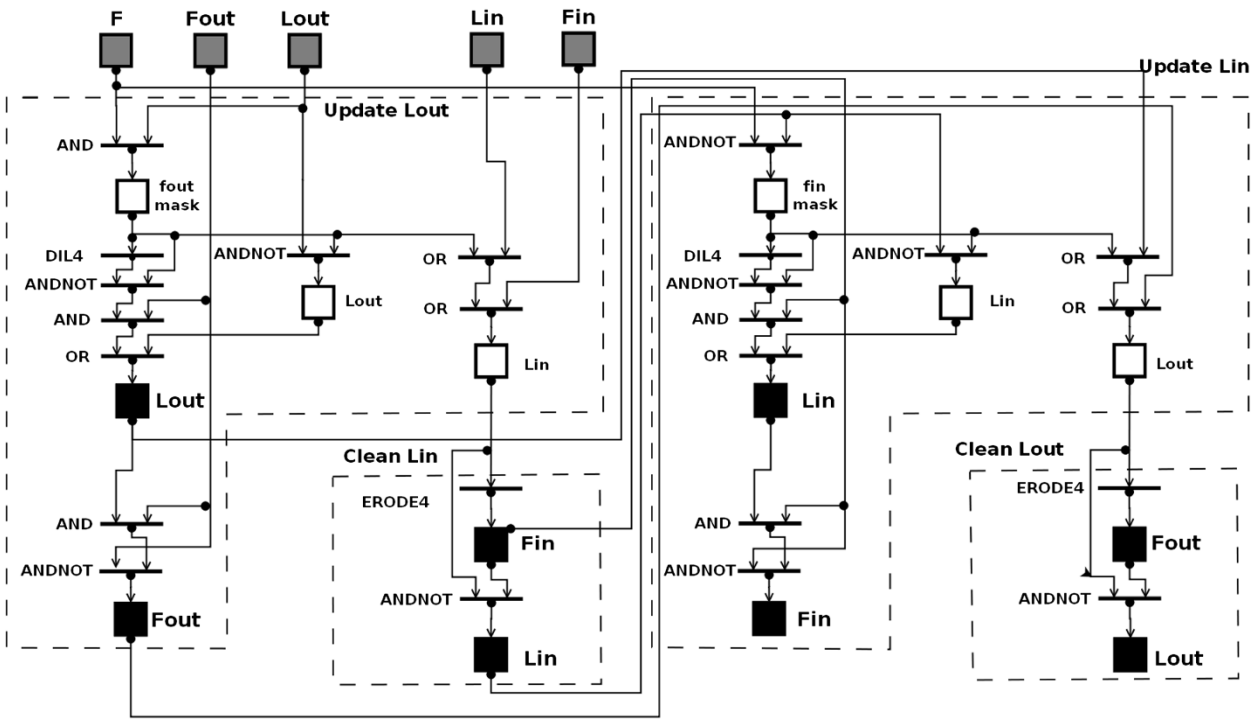

Figure 2 UMF diagram of LS evolution on CNN. Rectangles denote memories, bold short horizontal lines with capital operator names on the left denote template operations. Dashed lines indicate the phases corresponding to the four cycles of the Shi LSM. Black rectangles denote final forms of sets in that phase. Thin lines ending with arrows denote dataflow from memory to an operation, from an operation to an operation, or from an operation to a memory.

in the plan and writes the complete tile back to the global memory. First, each work-item calculates the new value of the pixel of the LS function. Then the neighbors of each pixel are updated as the *switch_out()* and *switch_in()* operations require, and the active front is cleaned to maintain the two pixel width. If there is no activity inside the tile, then set the corresponding pixel of the indicator image to *false*. The boundary of the tile requires special care, namely, to properly update the corresponding neighboring pixels of the LS function and the indicator. The source of the evolution kernel is provided as Additional file 2.

Table 2 shows execution time measurements of the work-efficient parallel algorithm on NVIDIA 780 GTX GPU compared to a baseline single-threaded implementation on Intel core i7-2600 CPU. The execution time was measured by the `gettimeofday()` C-function which has microsecond resolution. The table specifies the image resolution, the initial condition configuration, and presents the mean of the execution time of an iteration on GPU, on CPU, and the speedup. The iteration time on the GPU contains the execution time of both kernel functions (planner, iteration). The two kernels evenly share the execution time in the case of conventional, sparse initial condition; however, in the case of dense iteration step, the ratio of the evolution kernel can shift to 30:1 with respect to the planner.

Number of iterations

In the experiments more initial configurations were tested. In each configuration, regions of Ω and Γ were placed in a chessboard like pattern as it is showed in Figure 1A,B. Two sample objects are presented in Figure 1C,D that shall be detected. Additionally, the two objects represent the two object families: the degenerate and convex ones having worst-case bounds stated in the theorem 1 and 2.

Iteration results are presented in Table 1 together with the two different bounds of the given configuration. The number of iterations (N_{it}) was measured on the original sequential algorithm of Shi and these values are presented in the table. It is below or equal to the worst-case bounds in every cases.

In the case of CNN-UM, N_{it} coincides with the values presented in the table, while in the case of GPU implementation, N_{it} is consistently higher with one iteration. This means that it exceeded the bounds in the case of $n = 32$ and $n = 64$. However, the reason is as follows: the boundary pixels of the subregion have one iteration delay in the

Table 2 Time measurements on NVIDIA GTX 780 GPU compared to Intel core i7 CPU

Data size	Initial condition	$\bar{T}_{\text{iteration}}$ on GPU (μs)	$\bar{T}_{\text{iteration}}$ on CPU (μs)	N_{it}	Speedup
256 × 256	1 × 1	129	1,610	32	12.5
256 × 256	2 × 2	126	2,242	59	17
256 × 256	8 × 8	140	3,164	20	22
256 × 256	32 × 32	143	8,874	8	62
512 × 512	1 × 1	317	3,190	64	10
512 × 512	4 × 4	167	8,724	40	52
512 × 512	16 × 16	157	12,897	25	82
512 × 512	64 × 64	123	16,246	18	132
1,024 × 1,024	1 × 1	534	6,431	129	12
1,024 × 1,024	8 × 8	548	27,461	55	50
1,024 × 1,024	32 × 32	590	43,739	32	74
1,024 × 1,024	128 × 128	490	84,078	12	171
2,048 × 2,048	1 × 1	560	14,972	210	26
2,048 × 2,048	16 × 16	703	79,920	79	113
2,048 × 2,048	64 × 64	830	198,980	28	239
2,048 × 2,048	256 × 256	684	327,541	7	478

Presented results are the mean value of 100 runs.

cleaning process. This causes the additional iteration so it is not a violation of the theorems.

Validation

In this section, we compare the result of the exact numerical implementation and the Shi LSM for three different speed functions: mean curvature motion, Chan-Vese, and GAR. The quantitative comparison is made by the dice coefficient. Given the state of the two LS functions Ω_1 and Ω_2 of the two different methods, the coefficient is defined as

$$d(\Omega_1, \Omega_2) = \frac{2\text{Area}(\Omega_1 \cap \Omega_2)}{\text{Area}(\Omega_1) + \text{Area}(\Omega_2)} \tag{9}$$

Its value is in the range of 0 and 1; 0 means complete difference and 1 means complete agreement.

Mean curvature flow

In this case, the speed function is defined as

$$F = -\kappa \tag{10}$$

where κ is the (euclidean) curvature of the LS. It is the norm of the second derivative of γ with respect to the (euclidean) arc length ($\kappa = \|\gamma_{ss}(s)\|$, s is the arc length parametrization of the curve). Another possible, precise, and more easy way to calculate the curvature of an LS from ϕ is as follows:

$$\kappa = \text{div}\,\text{grad}\frac{\nabla\phi}{\|\nabla\phi\|} \tag{11}$$

This force term appears in almost every LS flow as a smoothing and regularizing term. The steady-state solution is a circle with infinitesimal diameter. In practice, the object region vanishes. In this case, not only the steady state but the evolution itself is also investigated. This is an autonomous motion and does not have any control term from an external image.

The details of the numerical approximation are as follows. The LS function ϕ is a signed distance function. It was recalculated after every 30 iterations. The time (T_{maximum}) run to 800 units. The time step (Δt) size has been set to 0.4. The curvature has been calculated from the LS function from Equation 11.

In the case of the fast LS evolution, the curvature was calculated according to the work Merriman, Bence, and Osher (MBO) [23,24], namely, by $G \otimes \phi$, where G is a 2D gaussian of a given variance.

Figure 3 shows the test initial condition for mean curvature motion and the state of the evolution after 20, 40, 60, and 80 iterations of the fast LS evolution.

Figure 4 shows the dice coefficient between the first 80 steps of the fast LS evolution and the corresponding state of the numerical approximation.

Chan-Vese flow

This method was proposed in [6] and its speed term is defined as

$$F = \mu\kappa - \lambda_1(c_1 - I)^2 + \lambda_2(c_2 - I)^2 \tag{12}$$

The parameters are set as follows: $\mu = 1, \lambda_1 = 0.8, \lambda_2 = 0.8$. I represents the input image intensities, the constants

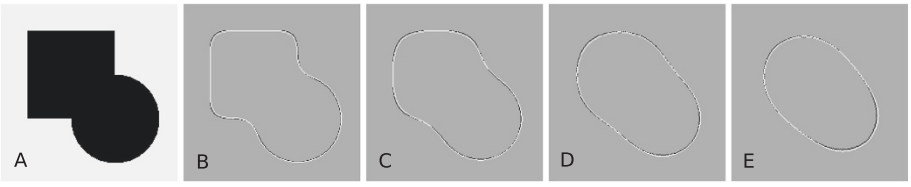

Figure 3 Comparison of mean curvature evolution of PDE approximation and fast LS evolution. This shows the initial condition and evolution of fast LS (white line) and numerical PDE approximation (black line). **(A)** Test initial condition for validation of mean curvature motion fast LS evolution against numerical PDE approximation. The test region contains positive, negative, and zero curvature regions and singularities as well. **(B)** State of evolution fast LF at $N_{it} = 20$ and PDE approximation at $T = 56.8$. **(C)** State of evolution fast LF at $N_{it} = 40$ and PDE approximation at $T = 190.8$. **(D)** State of evolution fast LF at $N_{it} = 60$ and PDE approximation at $T = 405.6$. **(E)** State of evolution fast LF at $N_{it} = 80$ and PDE approximation at $T = 706.8$.

$c_1 = 0.5$ and $c_2 = 0$ are simply the means of pixel intensities inside and outside the zero LS. The artificial time parameter runs to 180 units, the time step is 0.5 units. The total number of iterations are 360. The initial condition is 25 circles arranged uniformly in five rows and five columns each with diameter 27 pixels. The LS function (signed distance) is recalculated in each 30 iterations for the numerical solution. The initial condition is 5×5 circles each with diameter 27 pixels. The steady states of the two Cahn-Vese evolutions are shown in Figure 5A. The dice index of the two states is 0.998.

Geodesic active regions flow

This method was proposed in [7]. This method combines boundary and region-based information to segment an image. In this method, the pixel intensities are modeled with a GMM. The speedfield is as follows:

$$F = -\alpha \log\left(\frac{P(I|R_1)}{P(I|R_2)}\right) + (1-\alpha)\left(b\kappa + \nabla b \frac{\nabla\phi}{|\nabla\phi|}\right) \quad (13)$$

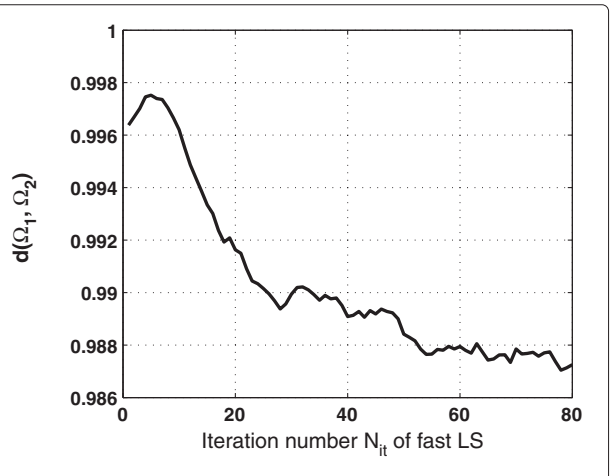

Figure 4 Dice index of mean curvature evolution. Ω_1 is the state of the fast LS evolution, and Ω_2 is the state of the numerical solution. The similarity between the two states is very high.

where R_1 and R_2 are the regions to be separated, b is a strictly decreasing function of boundary probability, and α is a balancing constant. In our case $\alpha = 0.3$, and b is defined as follows:

$$b = \frac{1}{1 + \|\nabla G \otimes I\|} \quad (14)$$

Here G is a 2D gaussian with $\sigma = 3$. The GMM parameters are calculated from the image histogram with a recursive expectation maximization algorithm. The artificial time runs to 6 units, the time step is 0.02 units. The total number of iterations are 300. The LS function (signed distance) is recalculated in each 30 iterations for the numerical solution. The initial condition is the same as in the case of Chan-Vese evolution, 5×5 circles each with diameter 27 pixels. Steady states are shown in Figure 5B. The dice index of the two states is 0.998.

Discussion

In this paper, given our investigation of the initial condition and the required number of iterations as a function of it, we presented two bounds on the required number of iterations of an LS evolution. The bounds were proven theoretically and validated experimentally with the original algorithm and also with two different mappings of the algorithm on many-core machines (GPU, CNN-UM). The bounds depend only on the initial configuration of the LS function. The many-core realizations required not only a very small number of iterations less than or equal to the bounds, but the execution of an iteration was also fast (see Table 2 for detailed measurement data).

In addition to the drastic decrease of the required number of iterations, the total execution time decreases as well if dense initial condition is used for the evolution. The total execution time on CPU with sparse initial condition is comparable to the total execution time with dense initial condition. For the smaller images, the dense initial condition was less effective by 30% to 15%; but in the case of the biggest image, the dense iteration was the faster by 35%. In the case of the dense initial condition on GPU, there is a

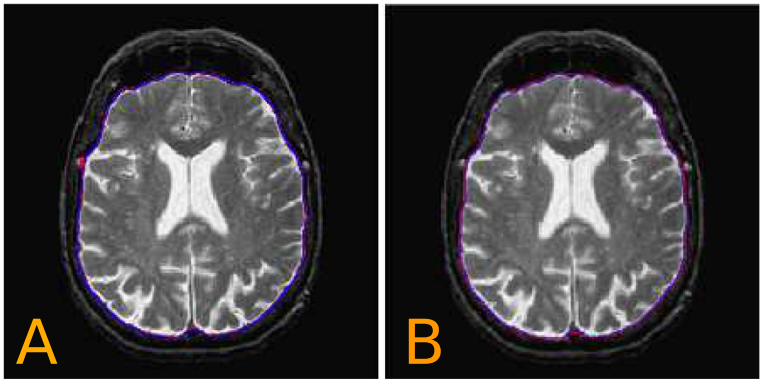

Figure 5 Validation of fast LS evolution. (A) CV and **(B)** GAR flow. Red corresponds to the numerical PDE solution while blue corresponds to the fast LSM. The two curves are nearly the same and the dice index is 0.998 in both cases.

significant speedup compared to the sparse initial condition in all cases since our proposed dense initial condition together with the algorithm utilizes the properties of the underlying architecture. Therefore, greater performance gain can be achieved on GPU if dense initial condition is used.

A great property of the results is the scalability. This is true for the performance as a function of cores and for the number of iterations as a function of size of the disjoint active fronts. Considering the chessboard-like initial configuration with increasingly finer regions, the general bound is proportional to the area of the regions and the convex bound is proportional to the half perimeter of the regions. This is changed in three dimensions to the volume of region in the case of general bound and half of the longest perimeter of the volume in the case of a convex bound.

The assumption on F is stronger in Theorem 1 than the one that was given in the convergence analysis in [2]. In the examples presented there, our stronger assumption stands for at least one of the regions Ω^*, Γ^*. However, there may be cases when for a short period of iterations the sign of F changes. Typically, this is the case when inside the true object region, the actual state of the LS function contains a concave background region with high negative curvature. In these cases, the curvature-based term can be greater than the region term (the pixel-intensity-based terms), but this is a temporary effect. As soon as the local concavity is vanished, the region term becomes again greater and the sign of F changes back. Furthermore, as it was declared in the introduction, the construction of the speedfield and its components is out of the scope of this paper. Additionally, the validations indicate that the method converges *de facto* to the same state as the exact numerical solutions.

The fact that the active front of the initial condition covers the whole image has a special consequence, namely, separate, disjoint regions of the same object or multiple target objects can be found automatically without user interaction. For example, the grey matter of the brain on an MRI slice is disconnected and may be composed of 8 to 20 disjointed regions on the given slice. The problem of detecting all regions is greatly simplified with our dense

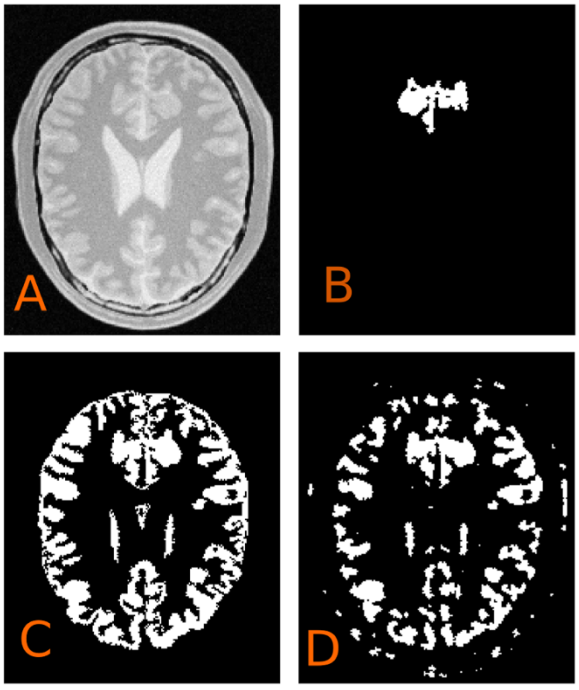

Figure 6 Initial condition dependence of evolution. (A) The original image to be segmented (grey matter of the brain). (Figure 6A is reproduced from [25]). **(B)** The reached solution of evolution started from a single circle initial condition. **(C)** The reached solution with our proposed initial condition (32 × 24 curves with diameter 3 pixels). **(D)** The reached solution of evolution with slightly modified parameters compared to the evolution shown on Figure 6C.

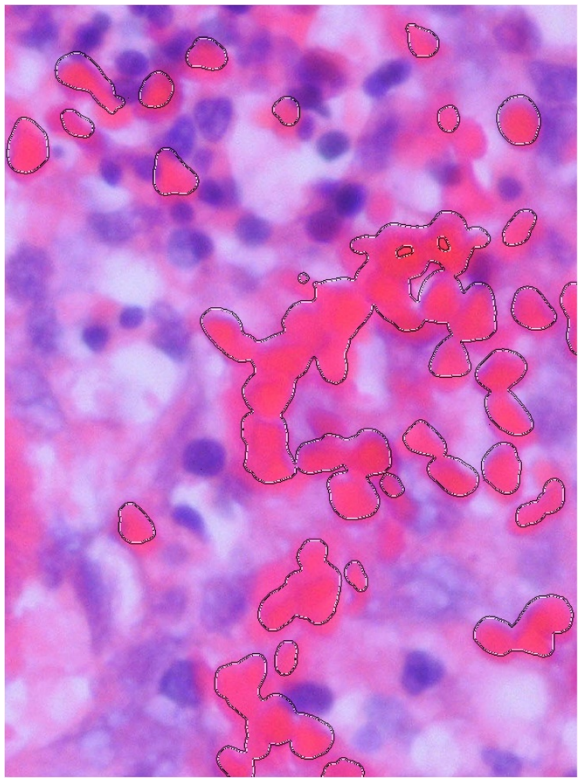

Figure 7 Histology image from the skin.

initial condition. Similarly, the selected group of cells on a histology image shows this property as well. Additionally, histology images can be extremely large (2 to 30 Mpixel), and the performance gain of our proposed method (initial condition together with the parallel algorithm) becomes more expressed on larger images. A conventional sparse initialization can easily fail this task, with wrongly chosen initial condition, see for instance the initialization and evolution of a gold standard LS implementation of [25], which is a widely used framework for medical image segmentation and analysis.

Figures 6 and 7 show an example. The evolution from a single-circle initial condition is presented on Figure 6B, while our result is presented in Figure 6C,D. It demonstrates its potential and it may be an initial condition for fine-tuning the segmentation with another method. Of course, the dense iteration may have the drawback of increased false-positive rate, for example see Figure 6D where the evolution runs with slightly different parameters, but this could be handled with more sophisticated speed functions or building *a priori* information into the initial condition. Figure 7 shows a histology image from the skin, where one class of cells is to be segmented.

It must be emphasized that the case studies presented here are not necessarily optimal mappings of the Shi LS evolution by any means. The purpose of presenting them is twofold: (1) to highlight the advantage of the proposed initial condition concept especially on those machines and (2) to give a proof of concept mapping of this fast evolution on two totally differently organized (virtual and physical) many-core machines.

Conclusions

To automatically detect segment object on an image or on a region of it, the LS-based algorithms are feasible tools. In this paper, it was shown theoretically and experimentally through two case studies that the initial condition plays an essential role in decreasing the execution time. It must be emphasized that this is only validated on many-core architectures where the computations can be distributed among the cores. Furthermore, based on the initial condition configuration, two worst-case bounds were given on the required number of iterations depending on the convexity of the object to be found. The bounds are proven theoretically and validated experimentally. Additionally, the execution time of one iteration was measured on both architectures. It was below $70+370$ µs on the Eye-RIS system handling a QCIF image (where 70 µs is the processing time and 370 µs is the outer memory delay). The timing results of the GPU is presented in Table 2 in details. In the case of the proposed dense initial condition on GPU, there is a significant speedup compared to the sparse initial condition in all cases since our dense initial condition together with the algorithm utilizes the properties of the underlying architecture. Therefore, greater performance gain can be achieved (up to 18 times speedup compared to the sparse initial condition on GPU).

The results and tools presented in this paper provide a method to efficiently map LS algorithms on many-core architectures and ensure bounds on the execution time through the two theorems.

Additional files

Additional file 1: The OpenCL source of the planner kernel.
Additional file 2: The OpenCL source of the evolution kernel.

Competing interests

The authors declare that they have no competing interests.

Acknowledgements

This research was supported by the European Union and the State of Hungary, cofinanced by the European Social Fund in the framework of TÁMOP 4.2.4.A/1-11-1-2012-0001 (National Excellence Program). The support grants TÁMOP-4.2.1.B-11/2/KMR-2011-0002 and TÁMOP-4.2.2/B-10/1-2010-0014 are also gratefully acknowledged. The authors would like to thank Ádám Rák for his help and suggestions.

Received: 17 September 2013 Accepted: 20 February 2014
Published: 10 March 2014

References

1. JA Sethian, *Level Set Methods and Fast Marching Methods: Evolving Interfaces in Computational Geometry, Fluid Mechanics, Computer Vision, and Materials Science*. (Cambridge University, New York, 2000)
2. Y Shi, W Karl, A real-time algorithm for the approximation of level-set-based curve evolution. IEEE Trans. Image Process. **17**(5), 645–656 (2008) [http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4480128]
3. D Adalsteinsson, JA Sethian, A fast level set method for propagating interfaces. J. Comput. Phys. **118**(2), 269–277 (1995) [http://www.sciencedirect.com/science/article/pii/S0021999185710984]
4. D Peng, B Merriman, S Osher, H Zhao, M Kang, A PDE-based fast local level set method. J. Comput. Phys. **155**(2), 410–438 (1999) [http://www.sciencedirect.com/science/article/pii/S0021999199963453]
5. G Sapiro, *Geometric Partial Differential Equations and Image Analysis*. (Cambridge University, New York, 2001)
6. T Chan, L Vese, Active contours without edges. Image Process. IEEE Trans. **10**(2), 266–277 (2001)
7. N Paragios, R Deriche, Geodesic active regions: a new framework to deal with frame partition problems in computer vision. J. Vis. Commun. Imag. Rep. **13**(1–2), 249–268 (2002) [http://www.sciencedirect.com/science/article/pii/S1047320301904754]
8. N Joshi, M Brady, Non-parametric mixture model based evolution of level sets and application to medical images. Int. J. Comput. Vis. **88**, 52–68 (2010) [http://dx.doi.org/10.1007/s11263-009-0290-5]
9. L Bertelli, S Chandrasekaran, F Gibou, BS Manjunath, On the length and area regularization for multiphase level set segmentation. Int. J. Comput. Vis. **90**(3), 267–282 (2010) [http://www.springerlink.com/index/10.1007/s11263-010-0348-4]
10. G Sundaramoorthi, A Yezzi, A Mennucci, G Sapiro, New possibilities with Sobolev active contours. Int. J. Comput. Vis. **84**, 113–129 (2009) [http://dx.doi.org/10.1007/s11263-008-0133-9]
11. AE Lefohn, JE Cates, RT Whitaker, *Interactive, GPU-based level sets for 3D segmentation*. ed. by Ellis RE, Peters TM, Medical Image Computing and Computer-Assisted Intervention - MICCAI 2003, Lecture Notes in Computer Science. vol. 2878 (Springer, Berlin Heidelberg, 2003) pp. 564–572, [http://dx.doi.org/10.1007/978-3-540-39899-8_70]
12. M Roberts, J Packer, MC Sousa, JR Mitchell, in *Proceedings of the Conference on High Performance Graphics*. A work-efficient GPU algorithm for level set segmentation (ACM, New York, 2010), pp. 123–132
13. O Sharma, Q Zhang, Q Anton, C Bajaj, in *2010 IEEE Conference on, Computer Vision and Pattern Recognition (CVPR)*. Multi-domain, higher order level set scheme for 3D image segmentation on the GPU (San Francisco, 13–18 June 2010), pp. 2211–2216
14. LO Chua, L Yang, Cellular neural networks: applications. Circuits Syst. IEEE Trans. **35**(10), 1273–1290 (1988)
15. G Cserey, C Rekeczky, P Földesy, PDE based histogram modification with embedded morphological processing of the level-sets. J. Circuits, Syst Comput. **12**(04), 519–538 (2003)
16. C Rekeczky, T Roska, in *Proceedings of the European Conference on Circuit Theory and Design, Volume 2*. Calculating local and global PDEs by analogic diffusion and wave algorithms (Helsinky University of Technology, Espoo, 2001), pp. 17–20
17. D Hillier, Z Czeilinger, A Vobornik, C Rekeczky, Online 3-D reconstruction of the right atrium from echocardiography data via a topographic cellular contour extraction algorithm. Biomed. Eng. IEEE Trans. **57**(2), 384–396 (2010)
18. Y Shi, WC Karl, in *Proceedings on IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP'05)*, vol. 2. A fast level set method without solving PDEs (Philadelphia, 18–23 March 2005, 2005), pp. 97–100
19. Y Shi, Object based dynamic imaging with level set methods. PhD Thesis, Boston University College of Engineering 2005
20. LO Chua, T Roska, PL Venetianer, The CNN is universal as the Turing machine. Circuits Syst. I: Fundam. Theory Appl. IEEE Trans. **40**(4), 289–291 (1993)
21. NVIDIA: *CUDA C Programming Guide* 2011. [https://developer.nvidia.com/cuda-toolkit-archive]. Accessed 23 January 2012
22. *OpenCL specification 1.1* 2011. [http://www.khronos.org/opencl/]. Accessed 1 April 2012
23. B Merriman, J Bene, S Osher, in *Computational Crystal Growers Workshop*. Diffusion generated motion by mean curvature. Edited by Taylor J, (Providence, RI 1992) pp.73–83.
24. B Merriman, JK Bence, SJ Osher, Motion of multiple junctions: a level set approach. J. Comput. Phys. **112**(2), 334–363 (1994). [http://www.sciencedirect.com/science/article/pii/S0021999184711053]
25. The insight toolkit 2012. [www.itk.org]. Accessed 20 November 2012

doi:10.1186/1687-6180-2014-30
Cite this article as: Tornai and Cserey: **Initial condition for efficient mapping of level set algorithms on many-core architectures.** *EURASIP Journal on Advances in Signal Processing* 2014 **2014**:30.