EURASIP Journal on Advances in Signal Processing
a SpringerOpen Journal

RESEARCH Open Access

High-resolution image segmentation using fully parallel mean shift

Balázs Varga* and Kristóf Karacs

Abstract

In this paper, we present a fast and effective method of image segmentation. Our design follows the bottom-up approach: first, the image is decomposed by nonparametric clustering; then, similar classes are joined by a merging algorithm that uses color, and adjacency information to obtain consistent image content. The core of the segmenter is a parallel version of the mean shift algorithm that works simultaneously on multiple feature space kernels. Our system was implemented on a many-core GPGPU platform in order to observe the performance gain of the data parallel construction. Segmentation accuracy has been evaluated on a public benchmark and has proven to perform well among other data-driven algorithms. Numerical analysis confirmed that the segmentation speed of the parallel algorithm improves as the number of utilized processors is increased, which indicates the scalability of the scheme. This improvement was also observed on real life, high-resolution images.

Keywords: High resolution imaging, Parallel processing, Image segmentation, multispectral imaging, Computer vision

1 Introduction

Thanks to the mass production of fast memory devices, state of the art semiconductor manufacturing processes, and vast user demand, most contemporary photograph sensors built into mainstream consumer cameras or even smartphones are capable of recording images of up to a dozen megapixels or more. In terms of computer vision tasks such as segmentation, image size is in most cases highly related to the running time of the algorithm. To maintain the same speed on increasingly large images, the image processing algorithms have to run on increasingly powerful processing units. However, the traditional method of raising core frequency to gain more speed–and thus computational throughput–has recently become limited due to high thermal dissipation, and the fact that semiconductor manufacturers are attacking atomic barriers in transistor design. For this reason, future trends of different types of processing elements–such as digital signal processors, field programmable gate arrays or general-purpose computing on graphics processing units (GPGPUs)–point toward the development of multi-core and many-core processors that can face the challenge of computational hunger by utilizing multiple processing units simultaneously [1].

Our interest in this paper is the task of image segmentation in the range of quad-extended and hyper-extended graphics arrays. We have designed, implemented and numerically evaluated a segmentation framework that works in a data parallel way, and which can therefore efficiently utilize many-core mass processing environments. The structure of the framework follows the bottom-up paradigm and can be divided into two main sections. During the first, clustering step, the image is decomposed into sub-clusters. The core of this step is based on the mean shift segmentation algorithm, which we embedded into a parallel environment, allowing it to run multiple kernels simultaneously. The second step is a cluster merging procedure, which joins sub-clusters that are adequately similar in terms of color and neighborhood consistency. The framework has been implemented on a GPGPU platform. We did not aim to exceed the quality of the original mean shift procedure. Rather, we have showed that our parallel implementation of the mean shift algorithm can achieve good segmentation accuracy with considerably lower running time than the serial implementation, which operates with a single kernel at a time. Numerical evaluation was run on miscellaneous GPGPUs with different numbers of

* Correspondence: varga.balazs@itk.ppke.hu
Pázmány Péter Catholic University, Faculty of Information Technology, Práter St. 50/a, Budapest 1083, Hungary

stream processors to demonstrate algorithmic scaling of the clustering step and speedup in segmentation performance.

The paper is organized as follows: in Sect. 2, we discuss the fundamentals of the mean shift algorithm, the available speedup strategies and the most important mean shift-based image segmentation methods. Section 3 discusses the basic steps of our version of the algorithm, while Sect. 4 describes the main parametric and environmental aspects of the numerical evaluation. The results are summarized in Sect. 5 and a conclusion is given in Sect. 6.

2 Related work

The first part of this section gives a brief overview of prominent papers that describe the evolution of the mean shift algorithm and also reveals the most important parts of its inner structure. The second part focuses on acceleration strategies, while the third considers state of the art algorithms that deal explicitly with high definition images and that rely partially or entirely on mean shift.

2.1 Mean shift origins

The basic principles of the mean shift algorithm were published by Fukunaga and Hostetler [2] in 1975, who showed that the mean shift iteration always steps toward the direction of the densest feature point region. Twenty years later, Cheng [3] drew renewed attention to the algorithm by pointing out that the mode seeking process of the procedure is basically a hill climbing method, for which he also proved convergence. Comaniciu and Meer [4] successfully applied the algorithm in the joint spatial-range domain for edge preserving filtering and segmentation. Furthermore, in [5] they gave a clear and extensive computational overview, proved the smooth trajectory property, studied bandwidth selection strategies and their effects on different feature spaces.

The standard mean shift algorithm is briefly summarized in the next subsection.

2.2 Mean shift basics

The mean shift technique considers its feature space as an empirical probability density function. A local maximum of this function (namely, a region over which it is highly populated) is called a mode. Mode calculation is formulated as an iterative scheme of mean calculation, which takes a certain number of feature points and calculates their weighted mean value by using a kernel function, such as the Gaussian. If we assume that the covariance matrix is an identity matrix multiplied with the variance (or with other words, the kernel is radially symmetric), the generalized form of the Gaussian is:

$$g\left(\left\|\frac{\mathbf{x}-\mathbf{x_0}}{\sigma}\right\|^2\right)=\frac{1}{\left(\sqrt{2\pi\sigma^2}\right)^d}e^{-\frac{\sum_d(\mathbf{x}-\mathbf{x_0})^2}{2\sigma^2}},\quad(1)$$

where $\mathbf{x}$-s are the considered feature point samples, $\mathbf{x_0}$ stands for the mean value, σ^2 denotes the variance and d is the number of dimensions of $\mathbf{x}$.

The algorithm can handle various different types of feature spaces, such as edge maps or texture, but in most cases of still image segmentation, a composite feature space consisting of topological (*spatial*) and color (*range*) information is used. Consequently, each feature point in this space is represented by a $\chi = (\mathbf{x_r}; \mathbf{x_s})$ 5D vector, which consists of the 2D position $\mathbf{x_s} = (x, y)$ of the corresponding pixel in the spatial lattice, and its 3D color value $\mathbf{x_r}$ in the applied color space (for instance, in the current paper, we use $\mathbf{x_r} = (Y, Cb, Cr)$ coordinates).

The iterative scheme for the calculation of a mode is as follows: let $\chi_\mathbf{i}$ and $\mathbf{z_i}$ be the 5D input and output points in the joint feature space for all $i \in [1, n]$, with n being the number of pixels in color image I. Then, for each i

1. Initialize $\chi_\mathbf{j}^{\mathbf{k=0}}$ with the original pixel value and position;

2. Compute a new weighted mean position using the iterative formula

$$\chi_\mathbf{i}^{\mathbf{k+1}}=\frac{\sum_{j=1}^{n}\chi_\mathbf{j}\, g\left(\left\|\frac{\mathbf{x_{r,j}}-\mathbf{x_{r,i}^k}}{h_r}\right\|^2\right)g\left(\left\|\frac{\mathbf{x_{s,j}}-\mathbf{x_{s,i}^k}}{h_s}\right\|^2\right)}{\sum_{j=1}^{n}g\left(\left\|\frac{\mathbf{x_{r,j}}-\mathbf{x_{r,i}^k}}{h_r}\right\|^2\right)g\left(\left\|\frac{\mathbf{x_{s,j}}-\mathbf{x_{s,i}^k}}{h_s}\right\|^2\right)},(2)$$

where g denotes the Gaussian kernel function with h_s and h_r being the spatial and range bandwidth parameters respectively, until

$$\|\chi_\mathbf{i}^{\mathbf{k+1}}-\chi_\mathbf{i}^{\mathbf{k}}\| < \text{thresh} \quad (3)$$

that is, the *shift* of the *mean* positions (effectively a vector length) falls under a given threshold (referred to as *saturation*).

3. Allocate $\mathbf{z_i} = \chi_\mathbf{i}^{\mathbf{k+1}}$.

In short, when starting the iteration from $\chi_\mathbf{i}$, output value $\mathbf{z_i}$ stores the position of the mode that is obtained after the last, $(k + 1)$ th step. Clusters are formulated in such a way that those $\mathbf{z_i}$ modes that are adequately close to each other are concatenated, and all elements in the cluster inherit the color of the contracted mode, resulting in a non-overlapping clustering of the input image. In this manner, segmentation is done in a nonparametric way: unlike in the case of some other clustering methods such as K-means, mean shift does not require the user to explicitly set the number of classes. In addition, as a result of the joint feature space, the algorithm is capable of discriminating scene objects based on their color and position, making mean shift a multipurpose, nonlinear tool for image segmentation.

Despite the listed advantages, the algorithm has a notable downside. The naive version, as described above, is initiated from each element of the feature space, which–as pointed out by Cheng [3]–comes with a computational complexity of $\mathcal{O}(n^2)$. The fact that running time is quadratically proportional to the number of pixels makes it slow, especially when working with high definition images.

Several techniques were proposed in the past to speed up the procedure, including various methods for sampling, quantization of the probability density function, parallelization and fast nearest neighbor retrievement among other alternatives. In next two subsections, we enumerate the most common and effective types of acceleration.

2.3 Acceleration strategies tested in standard definition

DeMenthon et al. [6] reached lower complexity by applying an increasing bandwidth for each mean shift iteration. Speedup was achieved by the usage of fast binary tree structures, which are efficient in retrieving feature space elements in a large neighborhood, while a segmentation hierarchy was also built.

Yang et al. [7] accelerated the process of kernel density estimation by applying an improved Gaussian transform, which boosts the summation of Gaussians. Enhanced by a recursively calculated multivariate Taylor expansion and an adaptive space subdivision algorithm, Yang's method reached linear running time for the mean shift. In another paper [8], they used a quasi-Newton method. In this case, the speedup is achieved by incorporating the curvature information of the density function. Higher convergence rate was achieved at the cost of additional memory and a few extra computations.

Comaniciu [9] proposed a dynamical bandwidth selection theorem, which reduced the number of iterations till convergence, although it requires some *a priori* knowledge.

Georgescu et al. [10] speed up the nearest neighbor search via locality sensitive hashing, which approximates the adjacent feature space elements around the mean. As the number of neighboring feature space elements is retrieved, the enhanced algorithm can adaptively select the kernel bandwidth, which enables the system to provide a detailed result in dense feature space regions. The performance of the algorithm was evaluated by performing a texture segmentation task as well as the segmentation of a 50 dimensional hypercube.

Usage of anisotropic kernels by Wang et al. [11] was aimed at improving quality. The benefit over simple adaptive solutions is that such kernels adapt to the structure of the input data; therefore, they are less sensitive to the initial kernel bandwidth selection. However, the improvement in robustness is accompanied by an additional cost of complexity. The algorithm was tested on both images and video, where the 5D feature space was enhanced with a temporal axis.

Several other techniques were proposed by Carreira-Perpiñán [12] to achieve speedups: he applied variations of spatial discretisation, neighborhood subsets, and an EM algorithm [13], from which spatial discretisation turned out to be the fastest. He also analyzed the suitability of the Newton method and later on proposed an alternative version of the mean shift using Gaussian blurring [14], which accelerates the convergence rate.

Guo et al. [15] aimed at reducing the complexity by using resampling: the feature space is divided into local subsets with equal size, and a modified mean shift iteration strategy is performed on each subset. The cluster centers are updated on a dynamically selected sample set, which is similar to the effect of having kernels with iteratively increasing bandwidth parameter; therefore, it speeds up convergence.

Another acceleration technique proposed by Wang et al. [16] utilized a dual-tree methodology. During the procedure, a query tree and a reference tree is built, and in an iteration a pair of nodes chosen from the query tree and the reference tree is compared. If they are similar to each other, a mean value is linearly approximated for all points in the considered node of the reference tree, while also an error bound is calculated. Otherwise the traversal is recursively called for all other possible node pairs until it finds a similar node pair (subject to the error boundary) or reaches the leafs. The result of the comparison is memory-efficient cache of the mean shift values for all query points speeding up the mean shift calculation. Due to the applied error boundary, the system works accurately, however the query tree has to be iteratively remade in each mean shift iteration at the cost of additional computational overhead.

Zhou et al. [17] employed the mean shift procedure for volume segmentation. In this case the feature space was tessellated with kernels resulting in a sampling of initial seed points. All mean shift kernels were iterated in parallel and as soon as the position of two means overlapped, they were concatenated subject to the assumption that their subsequent trajectory will be identical. Consequently, complexity was reduced in each iteration giving a further boost to the parallel inner scheme. Sampling on the other hand was performed using a static grid which may result in loss of information in the case when there are many small details on the image.

Jia et al. [18] also utilized feature space sampling along the nodes of a static-sized grid pattern. Next, 3-8 iterations of the *k*-means algorithm was run in order to preclassify the feature space. Finally the mean shift segmentation was initialized from the seed positions into which the *k*-means converged into. The framework was implemented in a GPGPU environment in which the authors managed to reach close to real-time processing for VGA-sized grayscale images.

Zhang et al. [19] approached the problem of complexity from the aspect of simplifying the mixture model behind the density function, which is done using function approximation. As the first step, similar elements are clustered together, and clustering is then refined by utilizing an intra-cluster quantization error measure. Simplification of the original model is then performed using an error bound being permanently monitored. Thus the mean shift run on the simplified model gives results comparable in quality to the variable bandwidth mean shift utilized on the original model, but at a much lower complexity and hence with a lower computational demand.

Although the performance, scaling and feasibility of the above approaches have not been tested on high definition images, they are discussed here due to their valuable contribution to the theory and the applications of mean shift. As the final step before entering the high definition image domain, the most prominent recent segmentation methods are briefly considered, which do not employ mean shift, but are mentioned here because of their real-time, or outstanding volumetric segmentation capability achieved via the utilized parallel scheme.

Hussein et al. [20] and Vineet et al. [21] proposed a parallel version of graph cuts, Sharma et al. [22] and Roberts et al. [23] both introduced a version of a parallel level-set algorithm, Kauffmann et al. [24] implemented a cellular automaton segmenter on GPGPUs, while Laborda et al. [25] presented a real-time GPGPU-based segmenter using Gaussian mixture models.

Finally, Abramov et al. [26] used the Potts model, a generalized version of the Ising superparamagnetic model for segmentation. In this system pixels are represented in the form of granular ferromagnets having a finite number of states. Equilibrium is found through two successive stages. As the first step, preliminary object boundaries are returned using a twenty-iteration Metropolis-Hastings algorithm, and the resulting objects of the binary image mask are then labeled. In the second step, segment labels are finalized in a five Metropolis iterations. To avoid false minima that may cause domain fragmentation, annealing iterations are performed slowly, which has an additional time demand, but still the system runs with 30 FPS at a resolution of 320 × 256, making it suitable for online video processing.

2.4 Acceleration strategies tested in high definition

In this section we discuss recently published, mean shift-related papers, all of them explicitly providing segmentation performance in the megapixel range.

Paris and Durand [27] employed a hierarchical segmentation scheme based on the usage of Morse-Smale complexes. They used explicit sampling to build the coarse grid representation of the density function. Clusters are then formulated using a smart labeling solution with simple local rules. The algorithm does not label pixels in the region of cluster boundaries; this is done by an accelerated version of the mean shift method. Additional speedup was obtained by reducing the dimensionality of the feature space via principal component analysis.

Freedman and Kisilev [28,29] applied sampling on the density function, forming a compact version of the kernel density estimate (KDE). The mean shift algorithm is then initialized from every sample of the compact KDE, finally each element of the original data set is mapped backwards to the closest mode obtained with the mean shift iteration.

Xiao and Liu [30] also proposed an alternative scheme for the reduction of the feature space. The key element of this technique is based on the usage of kd-trees. The first step of the method is the construction of a Gaussian kd-tree. This is a recursive procedure that considers the feature space as a d-dimensional hypercube, and in each iteration splits it along the upcoming axis in a circular manner until a stopping criterion is met, providing a binary tree. In the second step of this algorithm, the mean shift procedure is initialized from only these representative leaf elements resulting in modes. Finally, the content of the original feature space is mapped back to these modes. The consequence of this sampling scheme is decreased complexity, which, along with the utilization of a GPGPU, boosted the segmentation performance remarkably.

3 Computational method

Our framework is devoted to accelerate the segmentation speed of the mean shift algorithm with a major focus on its performance on high resolution images. The acceleration strategies used are summarized below:

1. Reduce the computational complexity by sampling the feature space.
2. Gain speedup through the parallel inner structure of the segmentation.
3. Reduce the number of mean shift iterations by decreasing the number of saturated kernels required for termination (referred to as abridging).

Figure 1 reveals the flowchart of the segmentation framework.

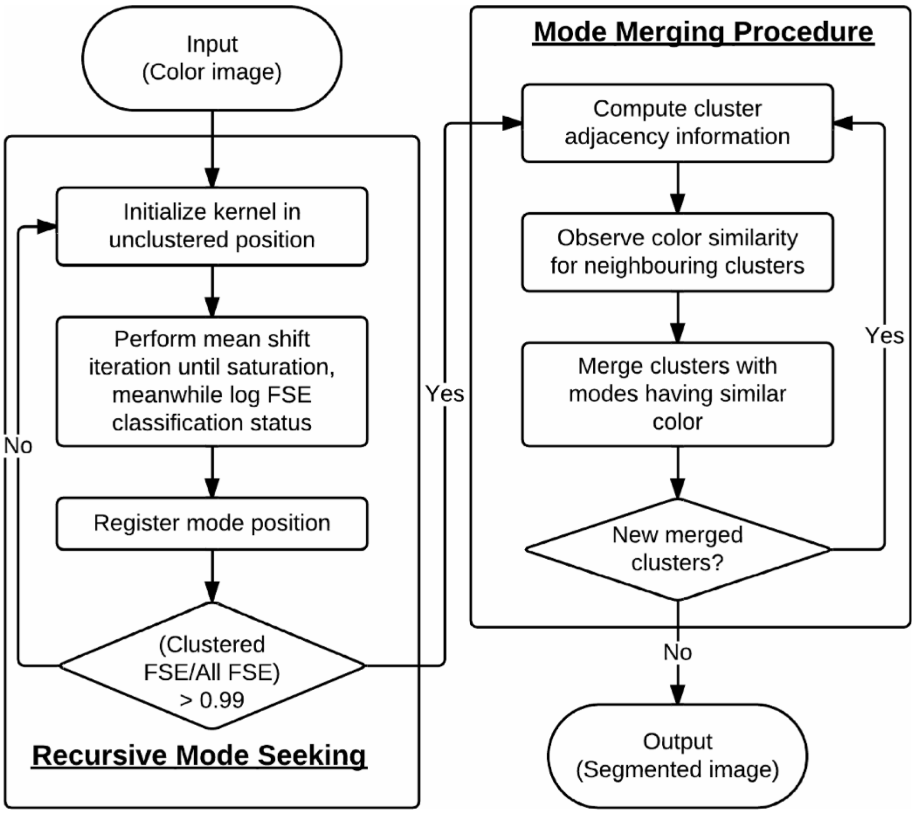

Input
(Color image)
Initialize kernel in unclustered position
Perform mean shift iteration until saturation, meanwhile log FSE classification status
Register mode position
(Clustered FSE/All FSE) > 0.99
No
Yes
Recursive Mode Seeking
Mode Merging Procedure
Compute cluster adjacency information
Observe color similarity for neighbouring clusters
Merge clusters with modes having similar color
New merged clusters?
Yes
No
Output
(Segmented image)

Figure 1 Flowchart of the segmentation framework. The result of the recursive mode seeking procedure is a clustered output that is an over-segmented version of the input image. The step of mode seeking is therefore succeeded by the merging step that concatenates similar clusters such that a merged output is obtained. The term "FSE" refers to feature space element.

3.1 Sampling scheme

The motivation behind sampling is straightforward: it reduces the computational demand, which is a cardinal aspect in the million-element feature space domain. The basic idea is that instead of using all *n* feature points, the segmentation is run on *n*' <<*n* initial elements. The mean shift iteration is then started from these seed points, and the other elements of the feature space are assigned to the so-obtained modes by using certain local rules [12,15,17,18,27,29].

There are however two major things one has to take into account in the case of sampling: undersampling the feature space can highly decrease segmentation quality, while oversampling leads to computational and memory-related overheads.

To address the above concerns, we have designed a recursive sampling scheme which works as follows:

1. Initialize a mean shift kernel in a yet unclustered element *i* of the feature space and repeat the mode seeking iteration until termination.

2. At this point $\chi_{\mathbf{j}}$ feature space element is assigned to a $\mathbf{z_i} = \chi_{\mathbf{i}}^{\mathbf{k+1}} = (\mathbf{x}_{\mathbf{r,i}}^{\mathbf{k+1}}; \mathbf{x}_{\mathbf{s,i}}^{\mathbf{k+1}})$ mode that is obtained from $\chi_{\mathbf{i}}$ sampled initial mean shift centroid if, and only if

$$\| \mathbf{x_{s,j}} - \mathbf{x}_{\mathbf{s,i}}^{\mathbf{l}} \| < h_s, \tag{4}$$

and

$$\| \mathbf{x_{r,j}} - \mathbf{x}_{\mathbf{r,i}}^{\mathbf{l}} \| < h_r, \tag{5}$$

where $l \in [0, k+1]$ denotes the mean shift iterations. In case a pixel is covered by more than one kernel, it is associated to the one with the most similar color.

3. If unclustered pixels remain after the pixel-cluster assignment, resampling is done in the joint feature space, and new mean shift kernels are initialized in those regions, in which most unclustered elements reside.

3.2 Dynamic kernel initialization

Since resampling is driven by the progress of the clustering of the feature space, both the *number* and the *position* of mean shift kernels is selected in proportion to the content of the image. Note that in the case of real life images, the image usually contains high frequency shading and gleams due to inconsistent lighting conditions. These phenomena appear in the feature space as outliers. For this reason, we applied a similar solution to Meer and Comaniciu's *"M threshold"* [4], but instead of removing small classes from the fully clustered image in a post-processing step, resampling is terminated when the number of clustered elements in the feature space reaches 99%, at which point all unclustered elements are assigned to the closest mode.

3.3 Cluster merging

After the iterative clustering procedure finished, cluster merging was performed. We used two simple rules for concatenation: cluster *i* and *j* are joined if they satisfy the following two criteria:

C1. The two clusters have a common border in terms of eight-neighbor connectivity.

C2.

$$\|\mathbf{x_{r,i}} - \mathbf{x_{r,j}}\| < h_r \tag{6}$$

where $\mathbf{x_{r,i}}$ and $\mathbf{x_{r,j}}$ are the range components of the corresponding cluster modes.

If both criteria hold for a pair of observed classes, the position of the mode in the feature space is recalculated as follows: let N_i and N_j denote the number of pixels in clusters *i* and *j* respectively, and let us suppose that the two are merged into cluster *k*. Then the spatial and range information carried by mode $\mathbf{z_k}$ of the newly formed cluster becomes a weighted average of the conjoined duet:

$$\mathbf{z_k} = \frac{N_i * \mathbf{z_i} + N_j * \mathbf{z_j}}{N_i + N_j}. \tag{7}$$

3.4 Parallel extension

Constructing a parallel inner structure for the framework is motivated by the following considerations:

1. The task is computationally intensive;

2. The task is highly data parallel [31].

The kernels of the mean shift segmenter perform the same, iterative procedure [5] on the data corpus, which allows for their efficient implementation on a parallel processor array.

The recursive serial framework described in Sect. 3.2 was extended to work in a parallel way:

1. Initialize a given number of mean shift kernels on the joint feature space.

2.a Perform the iterative mode seeking procedure (Equation 2) of the concurrent kernels simultaneously until termination.

2.b Perform pixel-cluster assignment according to Equations 4 and 5 respectively and save the position of the obtained modes.

3. Observe the topology of unclustered elements:

- If the feature space requires additional clustering, go to step 1.

- If the feature does not require additional clustering, proceed to cluster merging.

Merging is performed after the clustering is finished, and it is also a recursive procedure:

1. Compute pairwise neighboring information of the clusters (i.e. isolate clusters for which *C*1 is true).

2. Observe criterion *C*2 for adjacent clusters:

- If *C*2 does not hold for any cluster pair, terminate the merging procedure.

- Otherwise, continue with step 3.

3.a Concatenate clusters for which both *C*1 and *C*2 hold by recalculating the feature space position of the class-defining mode using Equation 7.

3.b Return to step 1.

While the theoretical advantages of parallel systems are widely known, the parallel implementation of the mean shift algorithm results in a few drawbacks that do not occur in the serial version.

The most important aspect of the parallel implementation is the memory intensive behavior. The position of a given mean is calculated using Equation 2 on the elements residing in the kernel's region of interest (ROI). However, the feature space elements grouped by the different ROI windows are stored in non-consecutive places in the device memory. This pattern does not favor coalescent memory access directly, which slows down the simultaneous mode seeking procedure. In order to accelerate these ROI operations, the ROI windows of a given mode seeking step are "cut" from the feature space and stored in a continuous structure.

The implementation induces another important change in the mean shift scheme. When running the mode seeking process given by Equation 2 on multiple autonomous kernels at once, it is not feasible to isolate saturated modes and replace them with new kernels in a "hot swap" way, due to the characteristics of block processing. Although such a switching solution is theoretically possible, it involves a lot of additional memory operations, which have a negative influence on the speed of the segmentation procedure. For this reason, a new mean position is calculated for each of the kernels utilized by the current sampling operation, until Equation 3 is met by every single one of them. Since this property is not present in the sequential mean shift, two important remarks should be made here.

1. This property does not result in corruption concerning image content retrieval. Kernels for which the shift of the mean value is below the threshold (Equation 3) will continue stepping toward the steepest ascent [3].

2. This property results in an overhead in terms of computational complexity.

3.5 The abridging method

In order to suppress the number of redundant iterations (in other words, the number of additional steps of the kernels that are beyond saturation), we introduced a so-called *abridging method*.

The method uses a single constant called the *abridging parameter* $A \in [0,1]$ that specifies the minimum proportion of kernels that is required to saturate. At the time instant this value is met, the ongoing mode seeking procedure is terminated and the next resampling iteration is initialized.

The usage of abridging is demonstrated through the following example (see Figure 2): consider a parallel mode seeking procedure, which is performed on n kernels simultaneously, and let us say that it takes m mean shift iterations until all kernels saturate. The ratio of kernel saturation follows an exponential pattern; such that a remarkable fraction of the kernels saturates in the first few shifting steps, so that in their case, each additional iteration is superfluous. The abridging parameter gives us a simple tool to terminate the mode seeking procedure after a reasonable amount of steps, when the number of saturated kernels is satisfactory.

The practical effect of the abridging parameter was studied by running the segmentation on 100 images provided by the "test set" of the Berkeley Segmentation Dataset and Benchmark (BSDS) [32] using various bandwidths and

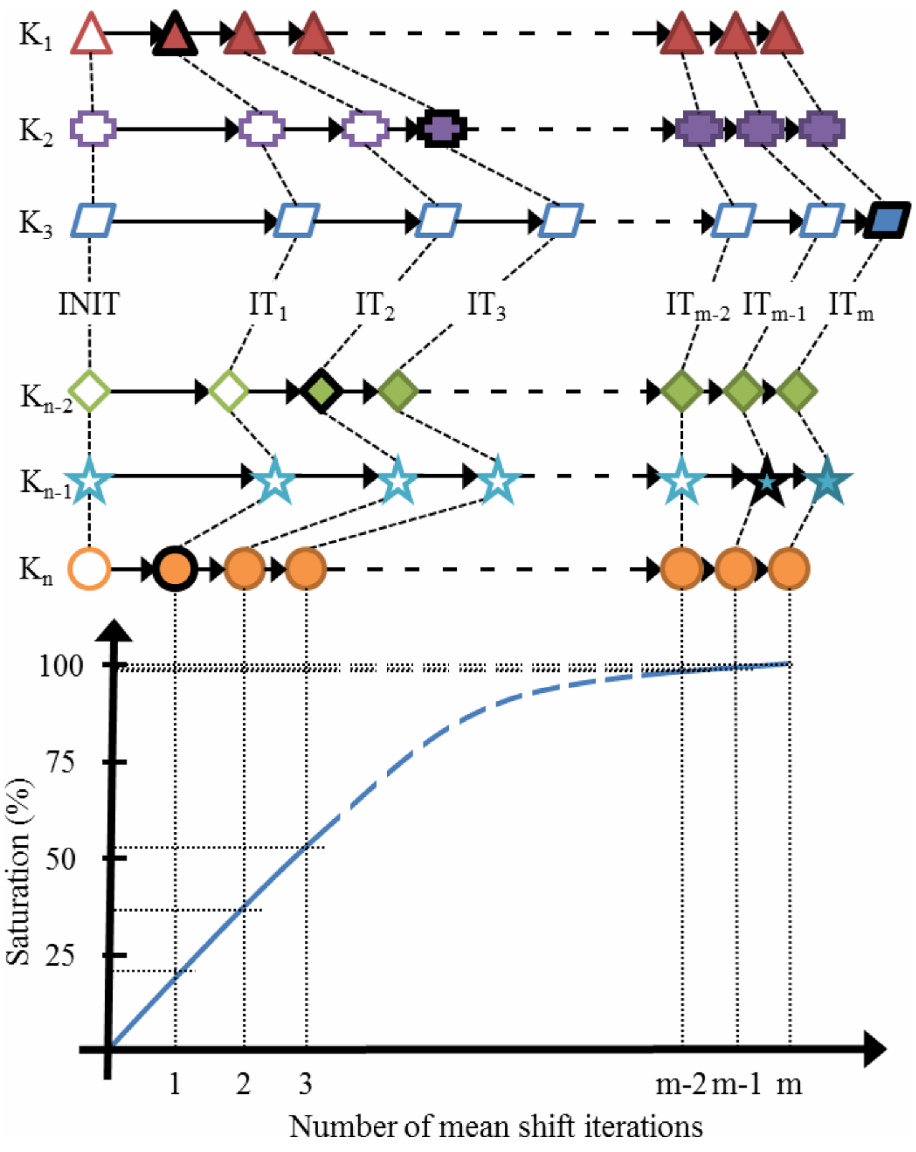


Figure 2 Demonstration of the kernel saturation tendency. The upper half of the figure gives an example for a mode seeking procedure with *n* simultaneously iterated kernel windows. Consecutive iterations are marked with "IT", and the length of *the arrows* are proportional to the length of the shift of the given kernel's mean. Saturated kernels are filled, with a *thick black* silhouette highlighting the first such iteration. Each consecutive iteration for that kernel is redundant. The bottom half illustrates the number of mean shift steps versus the percentage of saturated kernels.

abridging parameters (see Sect. 4.4). The main conclusions are discussed briefly in the following, whereas a more detailed description is given in the sections as noted. The following aspects were analyzed:

1. *Impact on the number of mean shift iterations.* The main motivation for using the abridging method is its strong reduction of the number of mean shift iterations. Compared to a setting of $A = 1$, a framework with $A = 0.6$ requires 3.1 times less mean shift iterations on average. In this case, the fact that in 95% of the cases the reduction was at least 2.04 times and standard deviation value of 0.79 underlines that the speedup is stable and present at a broad selection of bandwidths.

2. *Impact on the number of resampling iterations.* Abridging increases the number of resampling iterations, but has a small and strictly monotonically decreasing effect, which is inversely proportional to the bandwidth parameters. The number of resampling iterations showed an increase of 1-15% on average in a system with an abridging parameter of 0.6, which corresponds to 0.02-1.77 additional resamplings depending on the selected bandwidth parameters.

3. *Impact on segmentation speed.* The usage of the abridging parameter reduces the time demand of the mode seeking procedure, because although it may increase the number of resampling operations, it drastically cuts back the number of required mean shift iterations. Sect. 5.2 gives a complete overview.

4. *Impact on output quality.* The position of the mean values of kernels that did not saturate at the instant the abridging parameter caused termination are not situated at the local maxima of the underlying probability density map. Due to our pixel-cluster assignment scheme, this only implies the formulation of clusters that have more localized color information, and in practice, appears in the form of a slight over-segmentation. See Sect. 5.1 for a complete numerical evaluation.

5. *The actual number of saturated kernels.* The ratio of kernels saturated at termination generally exceeds the prescribed threshold ratio by 15-28% on average.

4 Experimental design

One of the most important jobs within a data parallel environment is controlling the simultaneous data access. In contrast to a simple threaded serial system, in which processing consists of consecutive–and therefore mutually exclusive–read and write memory accesses, a parallel environment requires additional buffering steps to properly handle simultaneous memory operations, and additional memory space to feed the processors.

Another issue with data parallel programming is that the host to device memory transfers (and vice versa) are slow, compared to accesses to local memory on the device. For this reason, fitting the data representation into device memory is a key element in context of speed, and also gives us a basic guideline during the selection of the number of (re)sampled kernel windows.

Lastly, limitations in the size of quickly accessible device memory calls for compact data representation, which again costs memory operations, and therefore time.

For the above reasons, parallelization of a given algorithm can only be considered effective if the speedup can be achieved in spite of all the enumerated constraints, and without sacrificing accuracy.

Our research prototype implementation of the segmentation framework was analyzed concerning three different aspects: the quality of the output, the scaling on different devices with various number of processors and the time demand of the algorithm on images with different size. Quality analysis was done with a broad selection of parameters in an exhaustive search-like scheme, which presents two notable benefits:

1. We obtained a broad overview about the robustness of the framework's output quality.

2. We obtained optimal parametrizations both in terms of speed and quality, which were used during the timing measurements as the two alternative settings to fully evaluate.

4.1 Hardware specifications

Our choice for the parallel hardware architecture was the GPGPU platform offered by NVIDIA. The measurements were performed on five GPGPUs with various characteristics. As a reference, the framework was also tested on a single PC equipped with 4GB RAM and an Intel Core i7-920 processor clocked at 2.66GHz, running Debian Linux. The technical specifications of the hardware are summarized in Table 1. Note that in the case of the NVIDIA S1070, only a single GPU was utilized (for this reason it is referred later on as S1070SG).

Compute capability numbers consist of two values: a major revision number that is indicating fundamental changes in chip design and capabilities, and a minor revision number referring to incremental changes in the device core architecture.

4.2 Measurement specifications

In the case of the scaling and timing experiments, the measurements were made on five different image sizes. The naming conventions and corresponding resolutions are summarized in Table 2.

4.3 Environmental specifications

The measurements were performed in the 5D joint feature space consisting of each pixel's *Y*, *Cb* and *Cr* color coordinates, and *(x,y)* spatial position. All channels were normalized into the [0,1] interval, but the luminance

Table 1 Parameters of the used GPGPU devices

Device name	No. of stream processors	Clock frequency (MHz)	Device memory (MB)	Compute capability
8800GT	112	1,500	1,024	1.1
GTX280	240	1,296	1,024	1.3
S1070SG	240	1,440	4,096	1.3
C2050	448	1,500	3,072	2.0
GTX580	512	1,544	1,536	2.0

channel was given an additional multiplier of 0.5 in order to somewhat suppress the influence of gradients that are often caused by the natural lighting conditions. Furthermore, the spatial representation of the pixels was equidistant in both dimensions, which basically means that in the case of a rectangular image, the channel representing the dimension with more pixels reached value 1, while the other channel's maximum was proportional to the dimension's aspect ratio. This way we ensured the anamorph property of the kernel, and the central symmetry suggested by Meer and Comaniciu [5].

The kernel window was selected to be the Gaussian, with distinct h_s and h_r parameters for the spatial and range domains respectively. In order to speed up the segmentation, the spatial weight kernel was calculated only once at the beginning of the segmentation, and was shifted to the position of the corresponding mode in each iteration. Furthermore, since the support of the Gaussian kernel is infinite, we only considered it within a radius, in which its value is above 0.1.

4.4 Quality measurement design

For output quality analysis, we used the Berkeley Segmentation Dataset and Benchmark in order to provide comparable quantitative results. The "test" set consisting of 100 pictures was segmented multiple times using the same parametrization for each image in a run. Three parameters were alternated among two consecutive runs: h_r taking values between 0.02 and 0.05, h_s with values in the interval of 0.02 and 0.05, both utilizing a 0.01 stepsize, and the abridging parameter ranging from 0.4 to 1.0 with a stepsize of 0.2. In each case, the segmenter was started with 100 initial kernels, and in every resampling iteration 100 additional kernels were utilized.

Note that since the BSDS benchmark evaluates quality based on boundary information, we generated soft boundary maps in the following way: the luminance channel of the segmentation framework's output was subject to morphological dilation using a 3 × 3 cross-shaped structuring element. The difference of the original and the dilated channel resulted in an intensity boundary map.

To enable easy comparison with other works, we have chosen the *F-measure* as the main quality indicator:

$$F = 2\frac{PR}{P+R}, \tag{8}$$

where $F \in [0,1]$ is the *F*-measure value, P stands for precision and R denotes recall. Precision is the ratio of the retrieved true boundary pixels and the retrieved elements, therefore it characterizes exactness. Recall is the quotient of the retrieved true boundary pixels and all true boundary pixels, hence it is a measure of completeness. Taking the harmonic mean of the two measures ensures that the *F*-measure stays well-balanced.

4.5 Timing measurement design

Timing measurements aimed at registering the running time of the algorithm on high resolution real life images. We formulated an image corpus consisting of 15 high quality images that were segmented in five different resolutions, using the parameter settings "speed" and "quality", obtained during the quality measurements (see Sect. 5.1). In each case, the segmenter was started with 10 initial kernels, and in every resampling iteration, 10 additional kernels were utilized.

4.6 Scaling measurement design

The mean shift iteration given in Equation 2 was timed individually on the different devices (and as a reference, on the CPU) to observe the scaling of the data parallel scheme. In order to give a complete overview, all linear

Table 2 Naming convention and resolution data of the images used for the timing and scaling measurements

Name of extended graphics array	Abbreviation	Resolution	Resolution in megapixels (MP)
Wide quad	WQXGA	2, 560 × 1, 600	4.1
Wide quad super	WQSXGA	3, 200 × 2, 048	6.6
Wide quad ultra	WQUXGA	3, 840 × 2,400	9.2
Hexadecatuple	HXGA	4, 096 × 3, 072	12.6
Wide hexadecatuple	WHXGA	5, 120 × 3, 200	16.4

combinations of spatial bandwidth parameters ranging from 0.02 to 0.05 with a stepsize of 0.01, and kernel numbers of 1, 10 and 20 were measured. Each displayed value represents a result that was obtained as the average value of 100 measurements.

5 Results

5.1 Quality results

As a result of alternating h_r, h_s and the abridging parameter, the framework was run with 64 different parametric configurations for each image of the 100 image BSDS test corpus.

The obtained average *F*-measure values for the different bandwidths and abridging parameters are displayed in Figure 3.

The highest *F*-measure value was 0.5816 for parameters h_r = 0.03 and h_s = 0.02 without any abridging, which fits in well among purely data-driven solutions [33]. It can be observed on Figure 3 that the output quality remained fairly consistent when relatively small bandwidths were selected. The system is more robust to changes made to the spatial bandwidth, while the effect of a high range bandwidth parameter decreases output quality. As one may expect, abridging has a negative effect on quality, but it can be seen that for certain parameter selection (namely, for $h_s \in$ [0.02, 0.03] and $h_r \in$ [0.03, 0.04]) even an abridge level of 0.6 results in acceptable quality. An interesting observation is that when both bandwidth parameters are set high, smaller abridging parameter values increase quality. The explanation for this is the following: as described in Sect. 3.5, abridging induces over-segmentation, and in this context, has an effect similar to having a smaller bandwidth parameter. This way, additional edges appear in the soft boundary map that is calculated for the benchmark. Among these edges, many are coincident with the ground truth reference of the benchmark, because the formulation of the extra clusters was data driven.

Table 3 displays the *F*-measure values for the different parametric constellations given as the percentage of the best result.

Boldface numbers indicate the cases when quality loss is less than 3% compared to the best result. Based on these results, we selected two parametric settings for the timing measurements:

1. the **Quality setting** was selected to be h_r, h_s, A = (0.03, 0.02, 1), while

2. the **Speed setting** was selected to be h_r, h_s, A = (0.04, 0.03, 6).

In the case of the quality setting the only guideline was to result in the best quality, while in the case of the speed

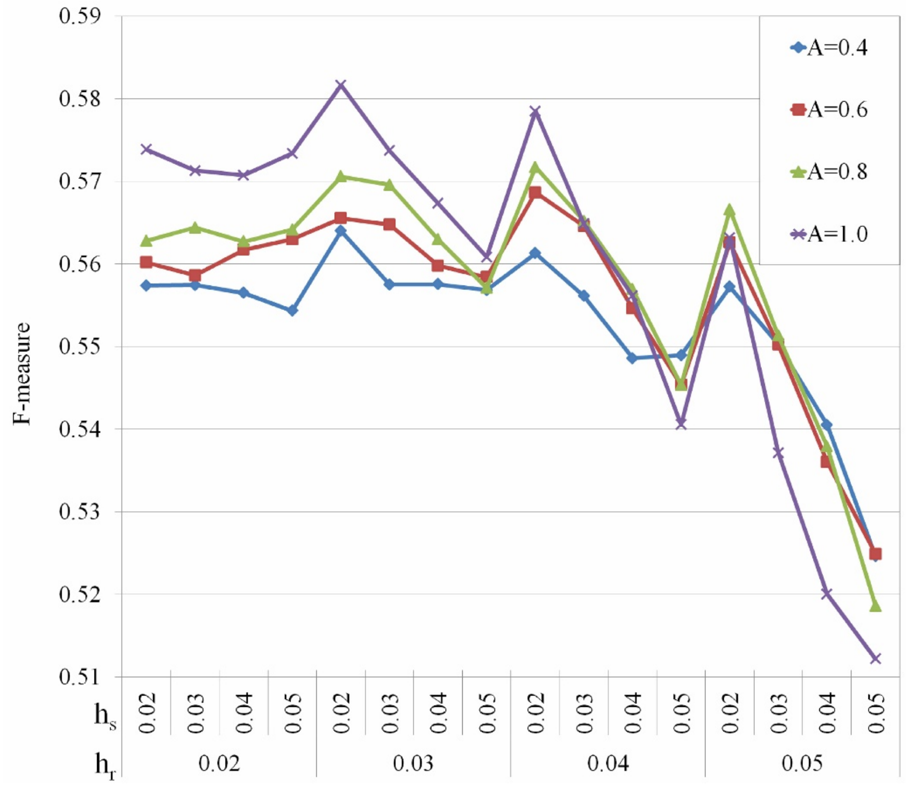


Figure 3 ***F*-measure values obtained for the different parametrizations of the segmentation framework**. h_s and h_r denote the spatial and range kernel bandwidths respectively, *A* values stand for different abridging constants.

Table 3 ***F*-measure values obtained with different abridging and bandwidth parametrization given as the percentage of the best result**

h_r	h_s	A = 0.4 (%)	A = 0.6 (%)	A = 0.8 (%)	A = 1.0 (%)
0.02	0.02	95.83	96.31	96.76	**98.66**
	0.03	95.84	96.04	**97.04**	**98.22**
	0.04	95.67	96.58	96.75	**98.12**
	0.05	95.31	96.80	**97.00**	**98.58**
0.03	0.02	96.97	**97.23**	**98.10**	***100***
	0.03	95.85	**97.10**	**97.92**	**98.64**
	0.04	95.86	96.24	96.79	**97.54**
	0.05	95.73	96.01	95.78	96.42
0.04	0.02	96.50	**97.77**	**98.30**	**99.46**
	0.03	95.62	***97.07***	**97.18**	**97.12**
	0.04	94.32	95.35	95.75	95.61
	0.05	94.38	93.76	93.76	92.94
0.05	0.02	95.81	96.73	**97.41**	96.83
	0.03	94.61	94.60	94.79	92.35
	0.04	92.93	92.16	92.48	89.41
	0.05	90.20	90.24	89.16	88.06

Boldface numbers indicate the cases when quality loss is less than 3% compared to the best result. The two settings selected for performance evaluation are highlighted with italic numbers

setting the preferences in order of precedence were the quality (should be better than 97%), the value of the abridging constant (smaller is faster) finally the size of the bandwidth parameters (bigger is faster due to the data parallel structure and the formulation of the pixel-cluster assignment scheme).

Figure 4 shows a few example images from the 100 image BSDS “test” segmentation corpus among with the segmented output and the obtained *F*-measure for both settings of quality and speed.

5.2 Running time results

The average running times measured on the 15 image corpus are summarized in Figure 5 using the speed setting and the quality setting.

In the case of the measurements made on the GPGPUs, the displayed values include all operations and memory accesses that have been performed in order to obtain the merged output image. The clock was started before the host to device data transfer carrying the input image started, and was stopped after the device to host data transfer carrying the merged output was completed, such that the output was retrieved into host memory. The same rule applies to the CPU measurements, but in this case obviously neither host to device transfers nor device to host transfers were necessary. When using either the GTX580 or the C2050, the average time demand for segmenting a 16 megapixel image was just above 18 seconds in the case of the speed setting, and a bit more than 33 seconds on the GTX580 using the quality setting. Compared to the running times of the CPU using the same, 16 megapixel setup as the GTX580, this means an acceleration of 16.93 times in the case of the quality setting, and an acceleration of 14.34 times in the case of the speed setting. Figure 6 displays the time spent on average to cluster a million pixels on the different platforms. It can be seen that by sacrificing 3% of the quality, a speedup of two can be achieved in most of the cases.

Figure 7 shows a few examples of the high quality input images from the 15 image segmentation corpus and the segmented output before and after the merging procedure.

5.3 Scaling results

As a result of the different parametrizations, the mean shift iteration was timed in 60 different constellations on the 5 GPGPU devices (plus the CPU) with each measurement indicating an average value recorded on 100 iterations (see Sect. 4.6).

The first aspect of evaluation was the scaling of performance on the different GPGPU device generations. Figure 8 displays the obtained running time in milliseconds for a single kernel measured using different resolutions with bandwidth parameters h_s = 0.02 and h_s = 0.05. The running times show a clear tendency: as a result of improved device characteristics (such as the number of stream processors, memory handling, caching and in some cases, operating frequency), the performance of newer device generations is superlinear compared to the older ones.

The second aspect of evaluation was the robustness of the operating time demand related to the number of kernels. In order to obtain expectations for a linear running time demand, the running time results measured when utilizing a single kernel were multiplied with 10 and 20 respectively. These expected values were then subtracted from the measured running time results and the outcome was evaluated for each device and the CPU. Table 4 displays the obtained results. On this table it can be seen that in the case when using 20 kernels, the maximum difference is negative for all GPGPUs. This means that the measured running time performance is always better than the expected one. In this context however there are exceptions, when 10 kernels were used. But in this case the average difference is negative for all of the devices, which indicates that on average, the running time benefit is present. The closer this value to zero, the more robust the running time on the used device subject to the alternation of the number of kernels is.

Finally, we investigated the running time of calculating the mean shift iteration on the different devices in proportion to the running time of the same task measured on the CPU. Figure 9 displays an overview of the

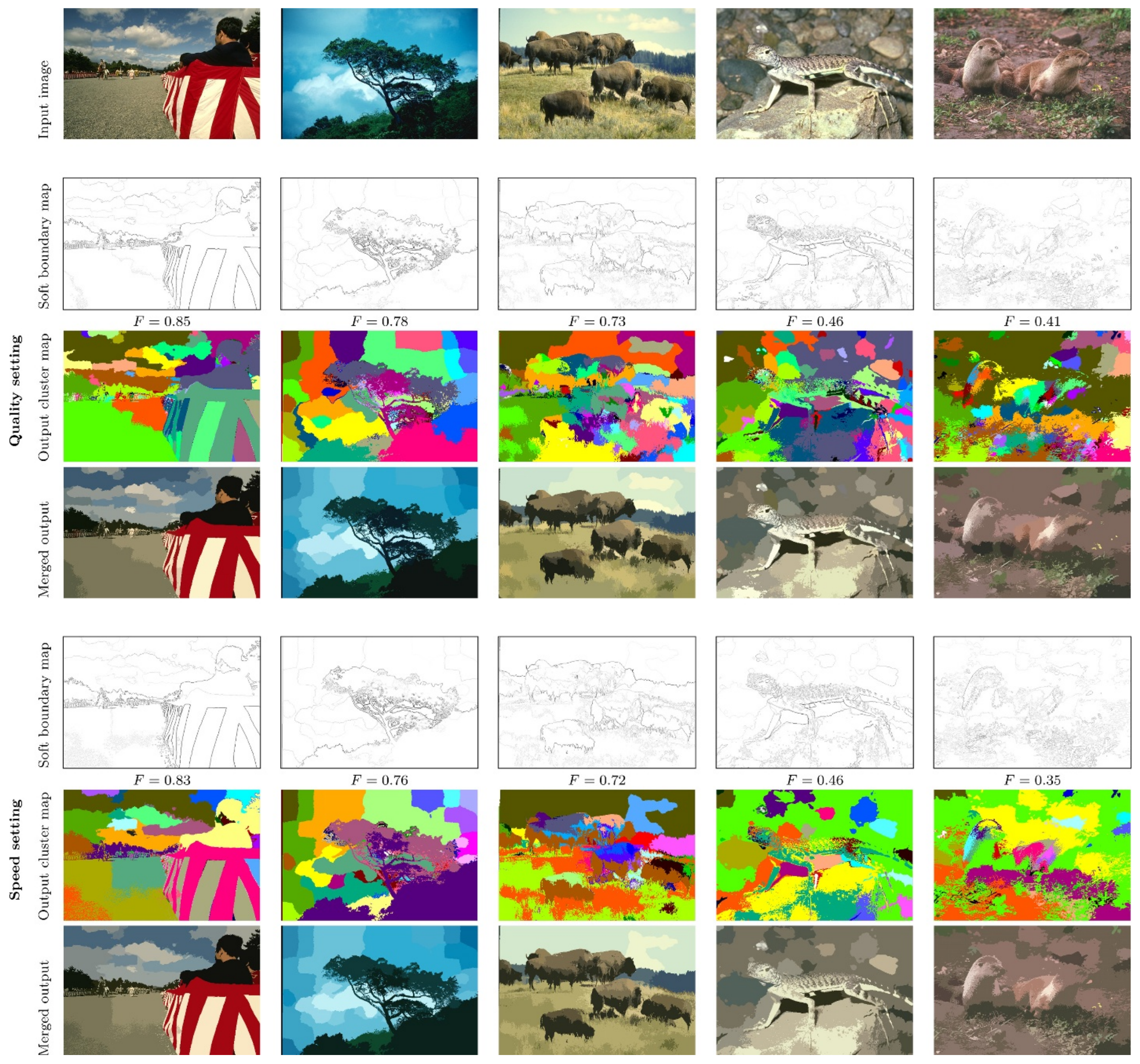


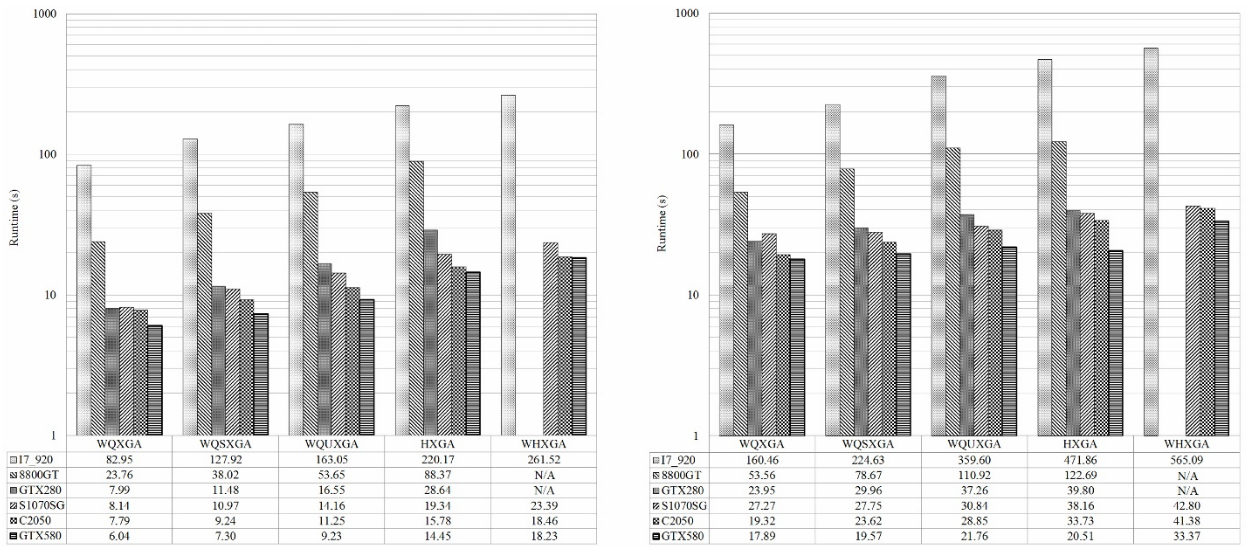
Figure 4 Five segmentation examples from the 100 "test" image corpus of the BSDS. Results for both the quality and the speed setting are shown. Quality evaluation was run using the soft boundary map generated from the merged output. For the sake of better visibility of the extent of the clusters is displayed in the form of a cluster map as well. *F* values denote the obtained *F*-measures.

speedup that is obtained by taking into account all of the different parametrizations of $h_s \in [0.02, 0.05]$ and the number of kernels being 1, 10 and 20.

As one may expect, the fastest performance was observed on the GTX580: compared to the CPU, the speed increase was greater than 28 for all parameter settings, with an average speedup of around 120. One may ask why the speedup of the mean shift iteration differs from the overall speedup of the framework. The answer to this question is that in the case of the former, only arithmetic operations are involved, so that these results represent more closely the speed of the GPGPU processing units. In contrast, the overall speedup–with all the data transfers, memory read and write operations that are involved–represent the integrated performance of the device.

Three factors affect the observed speedup of the mean shift iteration, these are: the size of the image, the kernel bandwidth and finally the number of kernels. In order to clarify their individual effect, Figure 10 displays the influence of varying these parameters on the observed speedup.

Figure 10 shows a clear trend: the parameter with the most influence on raising the speedup is the number of



	WQXGA	WQSXGA	WQUXGA	HXGA	WHXGA
▨ I7_920	82.95	127.92	163.05	220.17	261.52
▨ 8800GT	23.76	38.02	53.65	88.37	N/A
▨ GTX280	7.99	11.48	16.55	28.64	N/A
▨ S1070SG	8.14	10.97	14.16	19.34	23.39
▨ C2050	7.79	9.24	11.25	15.78	18.46
▨ GTX580	6.04	7.30	9.23	14.45	18.23

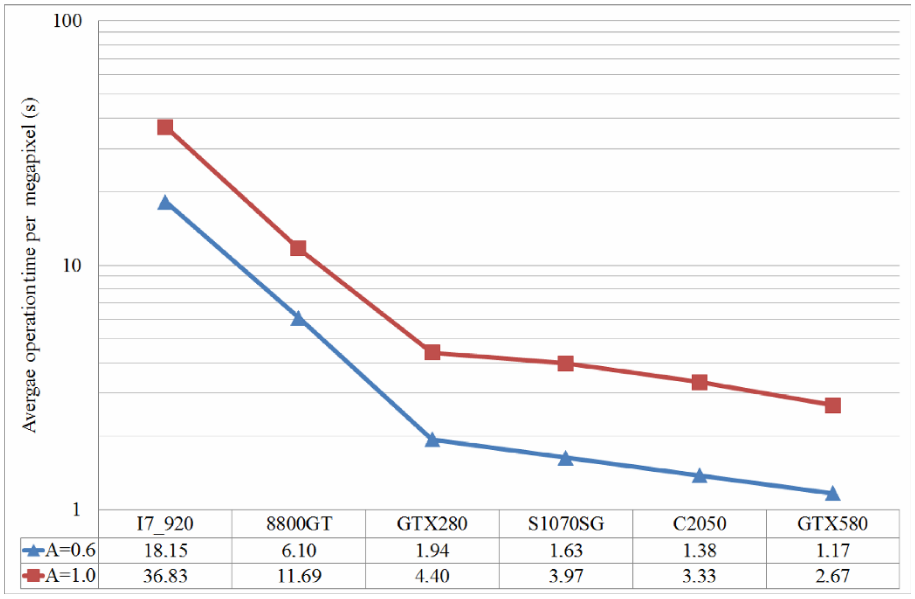
	WQXGA	WQSXGA	WQUXGA	HXGA	WHXGA
▨ I7_920	160.46	224.63	359.60	471.86	565.09
▨ 8800GT	53.56	78.67	110.92	122.69	N/A
▨ GTX280	23.95	29.96	37.26	39.80	N/A
▨ S1070SG	27.27	27.75	30.84	38.16	42.80
▨ C2050	19.32	23.62	28.85	33.73	41.38
▨ GTX580	17.89	19.57	21.76	20.51	33.37

Figure 5 Running time values of the algorithm run on images with different sizes using five different GPGPUs and the CPU as the reference. *The left* side displays running time obtained using the speed setting, *the right* side displays running times obtained using the quality setting. Each measurement displays an average value obtained from running the algorithm on 15 images. In the case of “N/A” values the onboard device memory sometimes became a bottleneck, which resulted in frequent caching operations that seriously slowed down the segmentation performance.

kernels. This is resulted by the data parallel nature of the task.

6 Conclusion

The details and design of an image segmentation framework have been presented in this paper. The core of the system is given by the parallel extension of the mean shift algorithm, which we accelerated by utilizing an abridging technique that can also be used in existing parallel mean shift techniques, such as [17,18,30], and a recursive sampling scheme that can narrow the complexity of the feature space, and is applicable in other solutions [18,19] as well. The framework was implemented on a many-core computation platform. A common



	I7_920	8800GT	GTX280	S1070SG	C2050	GTX580
▲ A=0.6	18.15	6.10	1.94	1.63	1.38	1.17
■ A=1.0	36.83	11.69	4.40	3.97	3.33	2.67

Figure 6 Average running time of clustering one megapixel on the different devices (and on the CPU) as a function of the abridging parameter. Difference in quality between the two settings is 3%.

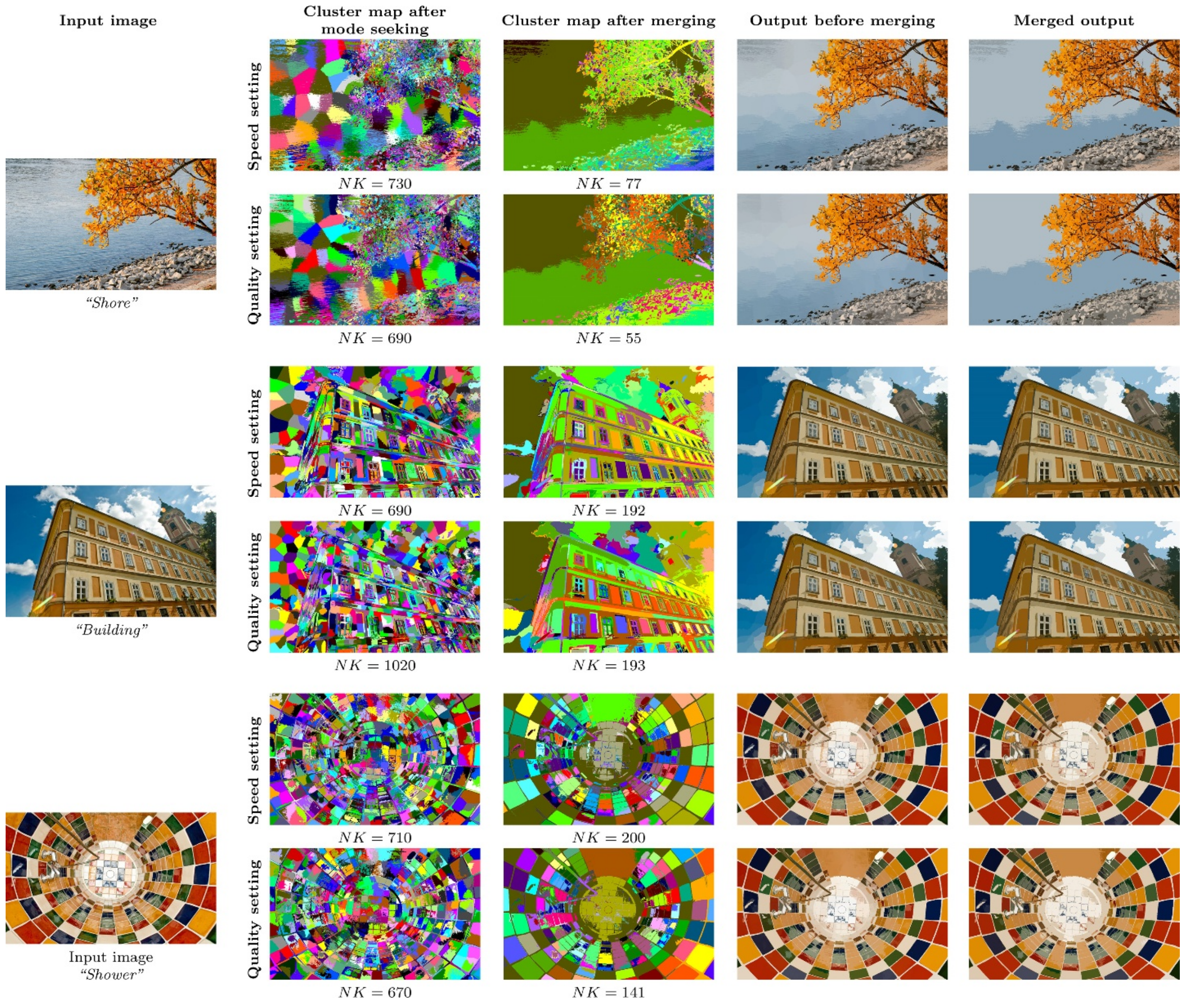


Figure 7 Three segmentation examples from the 15 image corpus. For the sake of better visibility, the extent of the clusters is also displayed in the form of cluster maps before and after the merging procedure. *NK* refers to the number of clusters.

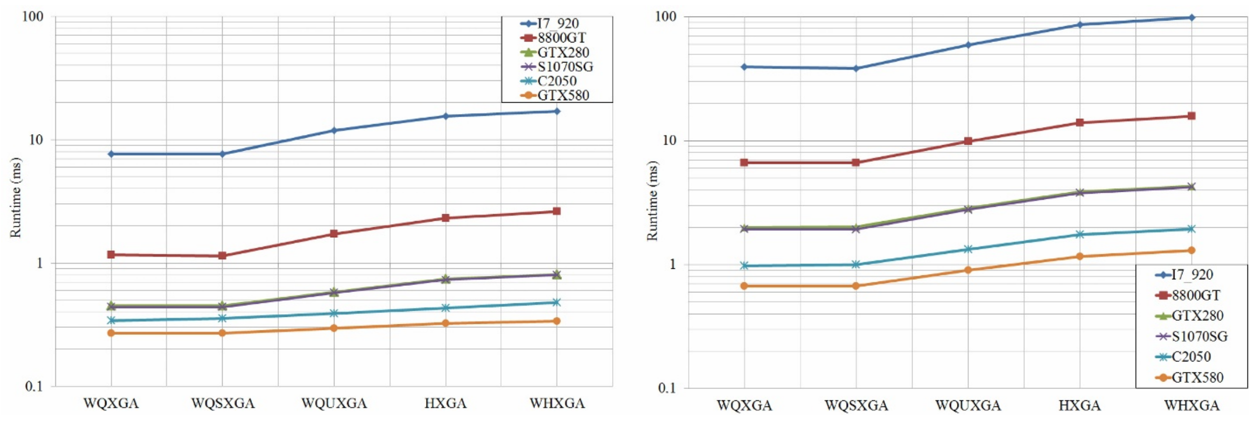


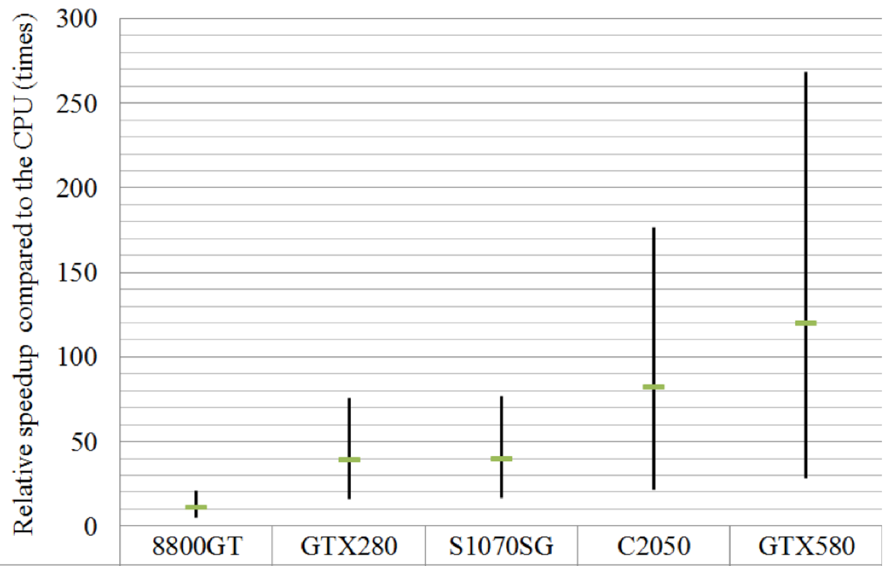
Figure 8 Running time tendencies of a mean shift iteration for single kernel on the different devices (and the CPU) using different resolutions. Spatial bandwidth selection was 0.02 for *the left* side and 0.05 for *the right*. Other bandwidths within this domain follow the same running time pattern.

Table 4 The robustness of the scaling on the different devices and the CPU.

Device Type	Relative standard deviation (%)	Minimum difference (ms)	Maximum difference (ms)	Average difference (ms)
(a) Results obtained using 10 kernels				
I7_920	90.57	3.9080	476.1333	162.7331
8800GT	162.39	-12.4617	2.4017	-2.3816
GTX280	70.80	-0.7733	1.1183	-0.5876
S1070SG	60.86	-0.7353	0.8477	-0.5672
C2050	19.24	-1.2275	-0.6879	-0.8152
GTX580	17.05	-0.8916	-0.4804	-0.7040
(b) Results obtained using 20 kernels				
I7_920	77.17	140.1300	3,436.8667	1,210.0043
8800GT	95.69	-62.3233	-4.4693	-18.4375
GTX280	70.48	-14.2183	-1.3773	-5.7447
S1070SG	65.22	-14.0697	-2.2727	-5.8767
C2050	45.28	-7.7653	-2.4625	-4.2816
GTX580	34.21	-5.7427	-2.1987	-3.3567

The statistics display the values obtained by comparing the expected running time values (derived by a multiplying the running time measured when using a single kernel) and the corresponding measured values. The relative standard deviation is the quotient of the standard deviation and the average

segmentation benchmark was used to evaluate the output quality and to demonstrate its robustness concerning parameter selection. Segmentation performance was analyzed on numerous high resolution real life images, using five different GPGPUs with miscellaneous specifications. The running time of a parallel mean shift iteration was measured on the different devices in order to observe the scaling of the data parallel scheme. The



	8800GT	GTX280	S1070SG	C2050	GTX580
Maximum speedup	20.90	75.71	77.14	176.61	268.59
Minimim speedup	5.09	16.24	16.46	21.54	28.17
▬ Average speedup	11.35	38.88	40.00	82.34	120.06

Figure 9 Speedup results obtained for different devices by pairwise comparison to the CPU. The basis of comparison were the running time values representing the time demand of calculating new position(s) of mean(s) with all combinations of $h_s \in$ [0.02,0.05] with number of kernels being 1, 10 and 20.

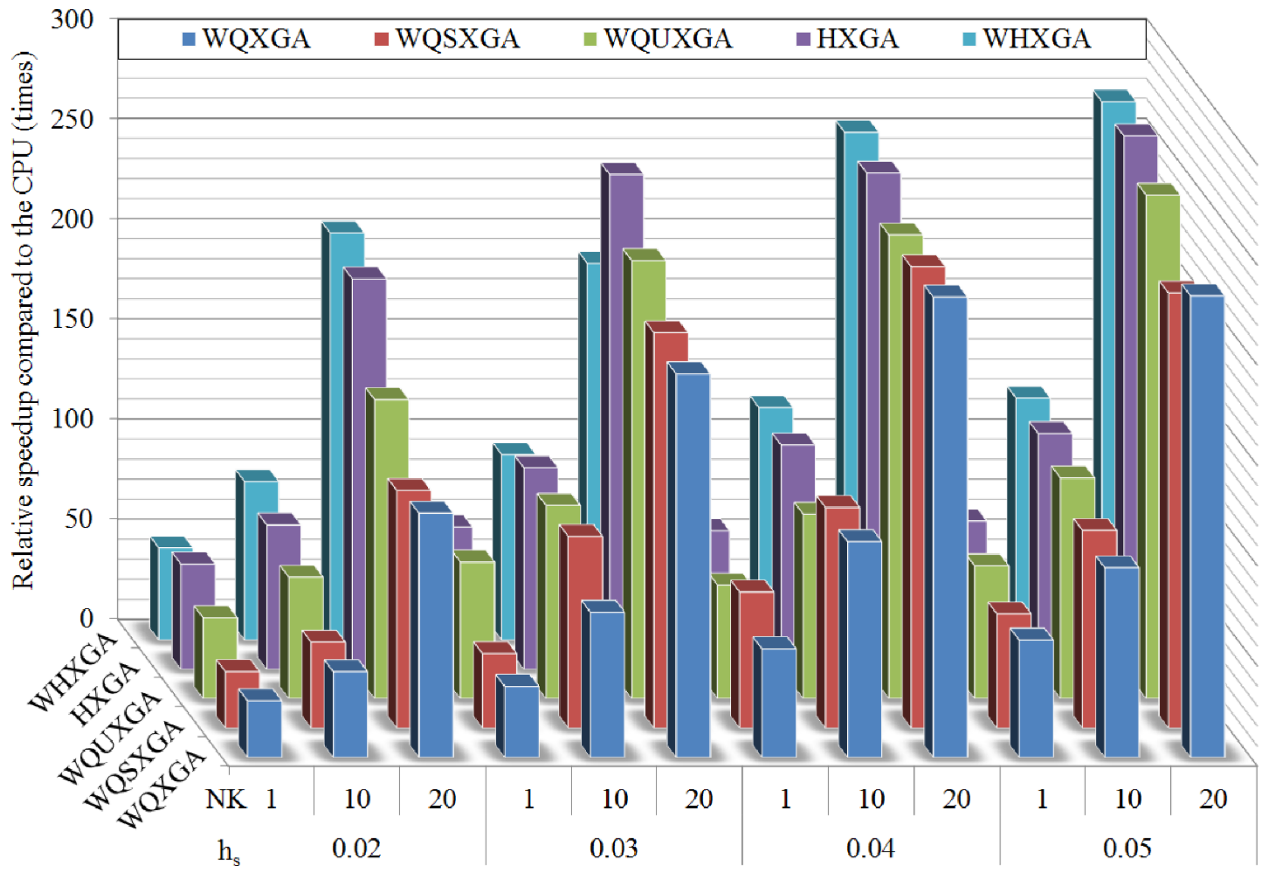


Figure 10 Speedup results of the GTX580 as a pairwise comparison to the CPU using different parameter settings. The bases of comparison were the running time values representing the time demands of calculating new mean positions. All combinations of $h_s \in [0.02,0.05]$ with number of kernels *(NK)* being 1, 10 and 20 were tested.

algorithm has proven to work fast and to provide good quality outputs.

Acknowledgements
The support of the Swiss Contribution, the Bolyai János Research Scholarship, the Tateyama Laboratory Hungary Ltd., the Hiteles Ember Foundation, the NVIDIA Professor Partnership Program and György Cserey are gratefully acknowledged.

Competing interests
The authors declare that they have no competing interests.

Received: 1 April 2011 Accepted: 23 November 2011
Published: 23 November 2011

References

1. H El-Rewini, M Abd-El-Barr, Advanced Computer Architecture and Parallel Processing (Wiley Series on Parallel and Distributed Computing), (Wiley, New York, 2005) ISBN-10:0471467405, ISBN-13:978-0471467403
2. K Fukunaga, L Hostetler, The estimation of the gradient of a density function, with applications in pattern recognition. IEEE Trans Inf Theory. **21**(1), 32–40 (1975). doi:10.1109/TIT.1975.1055330
3. Y Cheng, Mean shift, mode seeking, and clustering. IEEE Trans Pattern Anal Mach Intell. **17**(8), 790–799 (1995). doi:10.1109/34.400568
4. D Comaniciu, P Meer, Mean shift analysis and applications, in *Proceedings of the 7th IEEE International Computer Vision Conference*. **2**, 1197–1203 (1999)
5. D Comaniciu, P Meer, Mean shift: a robust approach toward feature space analysis. IEEE Trans Pattern Anal Mach Intell. **24**(5), 603–619 (2002). doi:10.1109/34.1000236
6. D DeMenthon, Spatio-temporal segmentation of video by hierarchical mean shift analysis. Center for Automation Research, University of Maryland, College Park (2002)
7. C Yang, R Duraiswami, NA Gumerov, L Davis, Improved fast gauss transform and efficient kernel density estimation. in *Proceedings of the Ninth IEEE International Computer Vision Conference*, 664–671 (2003)
8. C Yang, R Duraiswami, D Dementhon, L Davis, Mean-shift analysis using quasi-Newton methods, in *Proceedings of the International Conference on Image Processing*. **3**, 447–450 (2003)
9. D Comaniciu, An algorithm for data-driven bandwidth selection. IEEE Trans Pattern Anal Mach Intell. **25**(2), 281–288 (2003). doi:10.1109/TPAMI.2003.1177159
10. B Georgescu, I Shimshoni, P Meer, Mean shift based clustering in high dimensions: a texture classification example. in *Proceedings of the Ninth IEEE International Computer Vision Conference* 456–463 (2003)
11. J Wang, B Thiesson, Y Xu, M Cohen, Image and video segmentation by anisotropic kernel mean shift, in *Proceedings ECCV Series Lecture Notes in Computer Science*, vol. 3022, ed. by Pajdla T, Matas J (Springer, Berlin, 2004), pp. 238–249
12. MA Carreira-Perpiñán, Acceleration strategies for Gaussian mean-shift image segmentation. in *Proceedings of the IEEE Computer Society Conference Computer Vision and Pattern Recognition*. **1**, 1160–1167 (2006)
13. MA Carreira-Perpiñán, Gaussian mean-shift is an EM algorithm. IEEE Trans Pattern Anal Mach Intell. **29**(5), 767–776 (2007)

14. MA Carreira-Perpiñán, Fast nonparametric clustering with Gaussian blurring mean-shift, in *Proceedings of the 23rd International Conference on Machine Learning*, Series ICML '06, (ACM, New York, 2006), pp. 153–160
15. H Guo, P Guo, H Lu, A fast mean shift procedure with new iteration strategy and re-sampling. in *Proceedings of the IEEE International Conference Systems, Man and Cybernetics SMC '06*. **3**, 2385–2389 (2006)
16. P Wang, D Lee, AG Gray, JM Rehg, Fast mean shift with accurate and stable convergence. J Mach Learn Res Proc Track. **2**, 604–611 (2007)
17. F Zhou, Y Zhao, K-L Ma, Parallel mean shift for interactive volume segmentation, in *Machine Learning in Medical Imaging*, vol. 6357, ed. by Wang F, Yan P, Suzuki K, Shen D (Springer, 2010), pp. 67–75. Series Lecture Notes in Computer Science. doi:10.1007/978-3-642-15948-0_9
18. C Jia, W Xiaojun, C Rong, Parallel processing for accelerated mean shift algorithm. J Comput Aided Des Comput Graph , 3: 461–466 (2010)
19. K Zhang, J Kwok, Simplifying mixture models through function approximation. Neural Netw IEEE Trans. **21**(4), 644–658 (2010)
20. M Hussein, A Varshney, L Davis, On implementing graph cuts on CUDA, in *First Workshop on General Purpose Processing on Graphics Processing Units*, (Citeseer, 2007)
21. V Vineet, PJ Narayanan, CUDA cuts: fast graph cuts on the GPU. in *Computer Vision on GPU* 1–8 (2008)
22. O Sharma, Q Zhang, F Anton, CL Bajaj, Multi-domain, higher order level set scheme for 3D image segmentation on the GPU, CVPR, (IEEE, 2010), pp. 2211–2216
23. M Roberts, J Packer, M Sousa, J Mitchell, A work-efficient GPU algorithm for level set segmentation, in *Proceedings of the Conference on High Performance Graphics*, Eurographics Association, pp. 123–132 (2010)
24. C Kauffmann, N Piché, Seeded ND medical image segmentation by cellular automaton on GPU. Int J Comput Assist Radiol Surg. **5**(3), 251–262 (2010). doi:10.1007/s11548-009-0392-0
25. M Montañés Laborda, E Torres Moreno, J Martínez del Rincón, J Herrero Jaraba, Real-time GPU color-based segmentation of football players. J Real Time Imag Process. **6**, 1–13 (2011). doi:10.1007/s11554-011-0192-y
26. A Abramov, T Kulvicius, F Wörgötter, B Dellen, Real-time image segmentation on a GPU. *Facing the Multicore-Challenge* 131–142 (2011)
27. S Paris, F Durand, A topological approach to hierarchical segmentation using mean shift. in *Proceedings of the IEEE Conference Computer Vision and Pattern Recognition CVPR '07* 1–8 (2007)
28. D Freedman, P Kisilev, Fast mean shift by compact density representation. in *Proceedings of the IEEE Conference Computer Vision and Pattern Recognition CVPR 2009* 1818–1825 (2009)
29. D Freedman, P Kisilev, KDE paring and a faster mean shift algorithm. SIAM J Imag Sci. **3**(4), 878–903 (2010). doi:10.1137/090765158
30. C Xiao, M Liu, Efficient mean-shift clustering using gaussian KD-tree. Comput Graph Forum. **29**(7), 2065–2073 (2010). doi:10.1111/j.1467-8659.2010.01793.x
31. SH Roosta, *Parallel Processing and Parallel Algorithms: Theory and Computation*, (Springer, 1999) ISBN-10:0387987169, ISBN-13:978-0387987163
32. DR Martin, CC Fowlkes, D Tal, J Malik, A database of human segmented natural images and its application to evaluating segmentation algorithms and measuring ecological statistics, in *Proceedings of the 8th International Conference Computer Vision*. **2**, 416–423 (2001)
33. DR Martin, CC Fowlkes, J Malik, Learning to detect natural image boundaries using local brightness, color, and texture cues. IEEE Trans Pattern Anal Mach Intell. **6**(5), 530–549 (2004)

doi:10.1186/1687-6180-2011-111
Cite this article as: Varga and Karacs: **High-resolution image segmentation using fully parallel mean shift.** *EURASIP Journal on Advances in Signal Processing* 2011 **2011**:111.