DE GRUYTER OPEN

Print ISSN: 1312 – 2622; Online ISSN: 2367-5357
DOI: 10.2478/itc-2013-0011

Performance Analysis of Robust Image Features Detection Algorithms

I. Nikolova

Key Words: *Image processing; image features detection and description; general-purpose parallel computer platform; performance and speedup analysis.*

Abstract. *This paper deals with the challenging task of acquiring stable image features in a sequence of images of the same scene taken under different viewing positions by a digital still camera. Two popular contemporary algorithms for discrete feature detection: SIFT and SURF are regarded. The results of the timing performance analysis of their sequential implementations are presented and discussed. The performance speedup analysis and scalability tests with multi-threading and GPU-based implementations are analyzed.*

Introduction

The presented in this paper performance study of robust feature detection algorithms is a part of a research project related to one of the interesting and most challenging tasks: the acquisition of three-dimensional coordinates of objects in a scene on the basis of multiview two-dimensional images captured by a digital still camera. This problem is known as *scene depth recovery from multiview images* [8]. The basis of the solution stands in finding correspondences between images of the same scene. This is a combinatorial problem, whose computational time increases exponentially with the number of the features found in the matched images. The correspondence problem refers to the task of finding distinctive image locations (called *features, features point, points of interest* or *key points* [1,11,15]) in the images under consideration, and then match these locations based on their local appearance. The images taken from a different point of view and at different times represent the same scene with changes of the scale, rotation, and possibly changes in lighting conditions. Therefore, it is extremely important to use robust algorithms for image feature detection with abilities to repeatedly localize the distinctive features across different images invariant to image transformations. Furthermore, when the time is a limiting factor, these algorithms need to be efficient enough, so that the procedure of calculating the feature points does not require significant computational costs. The main strategy for reducing the processing time introduces the usage of a parallel form of computations. In characterizing the evolution of the computer system architectures, one can see a stable trend for utilization of parallelisms, at both instruction levels (ILP – Instruction Level Parallelism) and thread level (TLP – Thread Level Parallelism) [9]. The wider uptake of the massive parallel architectures, of MIMD class (Multiple Instruction Multiple Data) [9], GPGPU-based platforms and mobile devices with embedded Mpx cameras and computing capabilities, are a good prerequisite for searching suitable computing models for accelerating the feature detection algorithms. These models are adapted to the specifics of the physical machine models.

The paper provides a short overview of two of the most popular robust algorithmic schemes applied in 3D recovery solutions: the Scale Invariant Feature Transform (SIFT) [11] and the Speeded up Robust Features (SURF) [1]. Both algorithms perform simultaneous search for feature points and construct their descriptions that are invariant to changes in scale and rotation. This makes them particularly suitable for the purposes of 3D recovery on the base of multiview two-dimensional images. The experimental work is focused on profiling the program code and speed analysis of the feature detection and description phases. The experimental results demonstrate the effeiciency of the methods in providing a large number of features for a short time. The algorithms performance is measured by the processing time for images with different resolutions.

The rest of the paper is organized as follows. Section 2 gives a short overview of SIFT and SURF. The experimental work and the obtained results of timing performance analysis of serial implementation of the algorithms are presented and discussed in Section 3, which is followed in Section 4 by a discussion of the possibilities of applying optimizations for performance improvement of the investigated algorithms. Finally, some experimental results with two parallel versions of the serial SURF algorithm are presented that illustrate the obtained improvement in terms of the processing speed. The last section provides some final conclusions.

1. SIFT and SURF Algorithms' Description

The computational workflow of SIFT and SURF feature detector algorithms is shown in *figure 1.*

Two basic *Detection* and *Description* stages are used to identify the features of a given image and create their unique descriptions. The first stage is related to the extraction of image patterns (feature points). At first an image is loaded and initialized (Step C1). Then, the input image convolves with a set of Gaussian filters at different scales in order to construct the so-called scale space (Step C3). Such a scale-space is formed to guarantee the invariance of the features detection when the scale changes. Feature detection schemes used by SIFT and SURF differ in terms of the computational methods used for implementing the scale

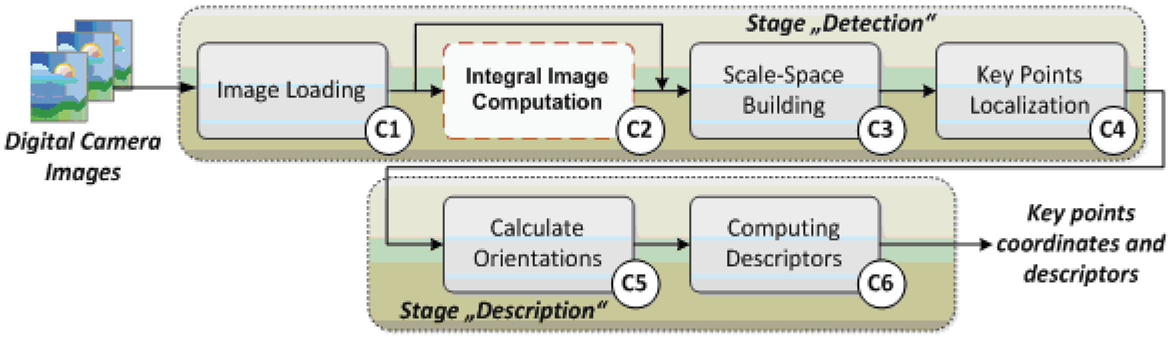


Figure 1. Computational workflow of SIFT и SURF algorithms

space. In practice, it is implemented as a pyramid. In SIFT the scale space pyramid is formed by repeatedly convolving the input image with a set of Gaussian filters. After that, the image is downscaled twice and convolved with the same set of filters. The process is repeated until a certain image size is reached. Finally, the adjacent Gaussian images are subtracted to produce a difference-of-Gaussian (DoG) image pyramid. As opposed to SIFT, in SURF the detection process relies on forming the scale space by convolving the input image using DoH (Determinant of the Hessian) filter matrix, which contains different 2D Gaussian second order derivatives. The use of Hessian ensures invariance with respect to rotation, but not in respect to a scale change. Therefore, SURF applies filters of different scales for calculating the Hessian to the input image. The image size remains unchanged, only the filter size changes. Thus, as opposite to SIFT, this decision is much more efficient with respect to the amount of memory that needs to be allocated for storing the scale space pyramid. Another improvement made by SURF's authors, is the approximation of the Gaussian filters, used in SIFT with integer weighted box shaped filters. This ensures also a better processing speed. Additionally, SURF also takes advantage of a swift convolution technique using integral images [17] (Step C2, SURF only). The value at each pixel in the integral image is the cumulative sum of intensities within the rectangular region, bounded by opposite corners that are located at the image origin and at the current coordinates. This is a technique for speeding up the convolution: the convolution response can be obtained by adding and subtracting four values in the integral image.

After building the scale space, a process of feature detection and localization via searching the scale space for extremes is started (Step C4). Each pixel of the image pyramid in SIFT is compared with its surrounding pixels within a 3×3 region in the current and adjacent scales. The pixels with a local maximum or minimum of the intensity value are considered as feature point candidates. The final number of features for a given image is obtained by removing the low contrast points introduced by noise and edge responses. In SURF, the local maxima of the DoH responses are found by the Non Maximum Suppression (NMS) search method with a 3×3×3 scanning window. The most suitable features are those with large DoH values. Features with small DoH values are regarded as noise and thus are not very distinguishable. Therefore, the SURF algorithm employs a detector threshold, which is used to reject these weak features.

The second stage of the computational workflow *Description* is associated with the formation of *N*-dimensional vector (descriptor) for each of the detected feature points. This is a two-step process: orientation assignment (Step C5) and feature vector descriptor computation (Step C6). To describe each feature, both algorithms summarize the pixel information within a local neighborhood. A gradient orientation value is computed on the base of the information around the considered feature point. In SIFT a gradient orientation histogram is computed in the neighborhood of the feature. This histogram represents the distribution of the gradient magnitude in a window, whose size is 1.5 times of the feature point scale. The peak of the histogram corresponds to the dominant orientation. In SURF the gradient orientation for each feature is calculated by convolving pixels in its neighborhood ($12\sigma \times 12\sigma$ square region positioned at each feature location) with two directional (horizontal and vertical) Haar wavelet filters of size 4σ. The last step after the orientation direction determination is the calculation of the feature vector. In SIFT a 16×16 window, centered at the feature point is used. A set of orientation histograms for each of 4×4-pixels subregions of the window is calculated. Each histogram contains 8 bins for each direction. Therefore, the length of the obtained feature descriptor is 128-dimensional ($4 \times 4 \times 8$). The original descriptor length of SURF is 64-dimensional and is formed on the base of $20\sigma \times 20\sigma$ square region around the feature, but also implementations with 128-dimensional descriptors are suggested.

2. Metodology and Evaluations of Serial SIFT and SURF Implementations

In this study the runtime performance over a sequence of images with different resolutions was measured. A series

Table 1. Coefficients of the relative time increase of the Detection and Description stages

	Num. of pixels*	**Growth factor**	**Processing time: Growth factor**			
			***Detection* stage**		***Description* stage**	
			SIFT++	**OpenSURF**	**SIFT++**	**OpenSURF**
Base	307 200					
2	480 000	1,56	1,57	1,53	1,02	2,34
3	786 432	2,56	2,59	2,59	1,60	3,34
4	1 228 800	4,00	4,08	3,83	2,42	4,97
5	1 470 000	4,79	4,90	4,54	2,64	4,73
6	1 920 000	6,25	6,36	5,92	3,31	6,02
7	2 073 600	6,75	6,82	6,37	3,16	5,68
8	3 145 728	10,24	10,67	10,13	4,78	8,73
9	4 915 200	16,00	16,71	15,03	7,07	12,18
10	7 680 000	25,00	25,75	23,47	10,92	18,56
11	12 192 768	39,69	44,06	37,47	19,37	26,25
	***Legend.**	Base: 640x480 (307 200 pixels) 2: 800x600; 3: 1024x768; 4: 1280x960; 5: 1400x1050; 6: 1600x1200; 7: 1920x1080; 8: 2048x1536; 9: 2560x1920; 10: 3200x2400; 11: 4032x3024				

of images of a static interior scene with many different objects were used for the evaluations, which enables the detection of a large number of features. The mages were captured with a digital still camera Olympus E-P2 at different resolutions. Three of the images were taken at resolutions maintained by the camera itself (1280Ч960, 3200Ч2400 and 4032Ч3024 pixels), while the other nine are downscaled versions of the 4032Ч3024 resolution image. The range of image resolutions between 320Ч240 and 4032Ч3024 pixels was chosen in such a way as to cover the basic formats of the mass digital still and video cameras. Thus, an image dataset „Lab1311-MR“ (MR – Multiple Resolutions) of 12 resolutions was created.

The experimental tests are based on two open source implementations of SIFT and SURF detector and descriptor algorithms widely used for research purposes, SIFT++ [16] and OpenSURF [4]. Both algorithms were tested on a general-purpose hardware platform with the following configuration: Intel(R) Xeon(R) CPU E5-2609 @ 2.40GHz, Dual CPU, 4 cores per socket, 32GB RAM. The source codes in C/C ++ were compiled by GNU g++ with the command-line flag „-O3“ under Linux, Ubuntu. The timing data were gathered using a *gprof* profiling tool [7,13] and time functions from standard C/C++ libraries.

Figure 2 shows the result of the feature detector for one of the tested images of resolution 1024Ч768 pixels. The features detected are marked by colored circles. The diameter of the circle corresponds to the scale, at which the feature point is localized, and the radius shows the dominant gradient orientation.

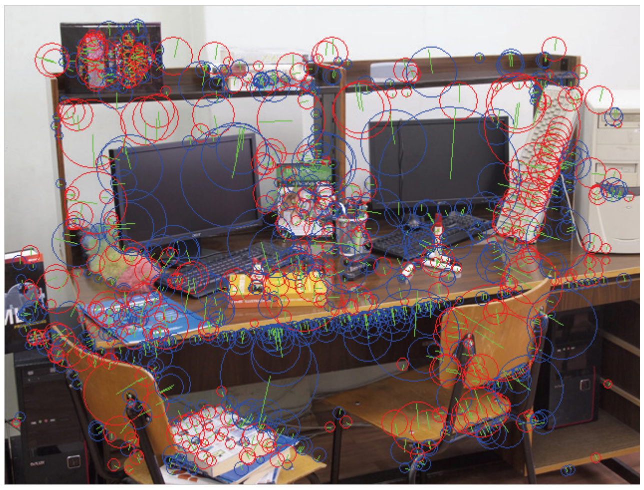
2.1. Experimental Results for the Timing Performance

To compare the execution times of SIFT and SURF algorithms, their control parameters were experimentally tuned, so that the number of the features detected for the same image is approximately equal for both implementations. The difference in the number of the detected features is less than 1% for almost all tested images.

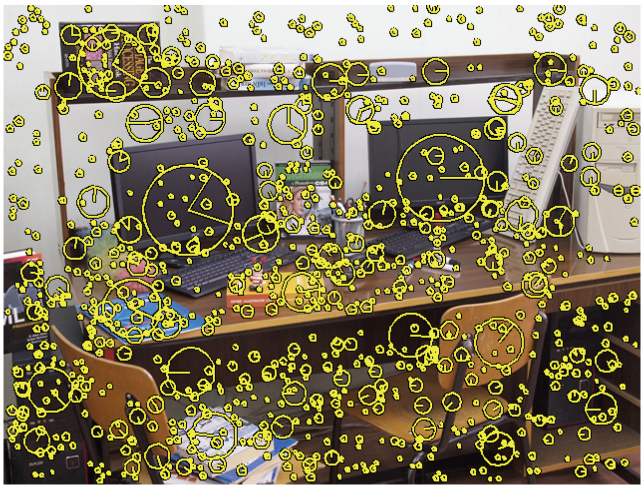
Table 1 shows the experimental data for the computational time increase of the *Detection* and *Description* stages depending on the image size. The results presented correspond to resolutions from 800×600 to 4032×3024 pixels, regarding the standard VGA resolution of 640×480 pixels as basis. As it can be seen, the processing time increase at the *Detection* stage follows the trend of the image size in both of the algorithms. SIFT algorithm shows the only exception for the maximum resolution image. This is due to the scale space building scheme implemented in SIFT, based on doubling the size of the input image for the first octave of Gaussian image pyramid calculation. For SURF, the growth factor is similar to that of SIFT for the entire range of resolutions, as opposite to the *Description* stage. It can also be seen that the *Description* processing stage at resolutions over 2MPx is less dependent on the increase of the images pixels being processed than the *Detection* stage.

Figures 3(a) and *(b)* show the computation time (for complete runs and per algorithm's stage) required to process images of different resolutions, as well as the relative computational speed of OpenSURF with respect to SIFT++. The time ratios for *Description* and *Detection* stages are given in *figure 4*. The percentages of the computational time spent at each of the workflow steps of SIFT and SURF for images with small and high resolution are shown in *figures 5* and *6*.

The most time consuming operations in both implementations are related to Steps C3 and C6 (see *figure 1*). They take up to ~80% of the overall computation time. The computational cost of building the scale space in SIFT remains relatively high regardless of the number of pixels being processed: more than 50% of the time for 640×480 pixels image, and less than 60% for 4032×3024 pixels. The relative time increase is proportional to the image resolution in SURF. These observations can be explained with the

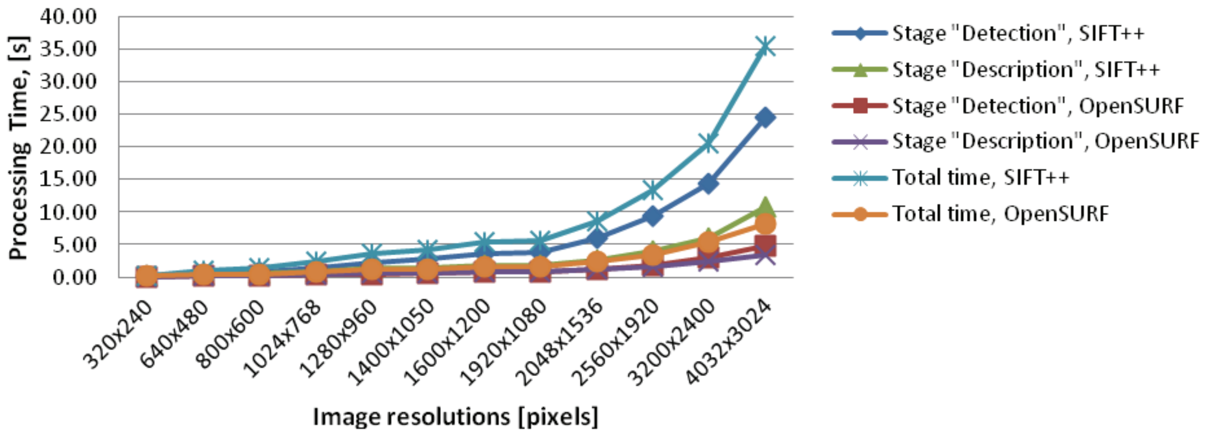


(a) OpenSURF

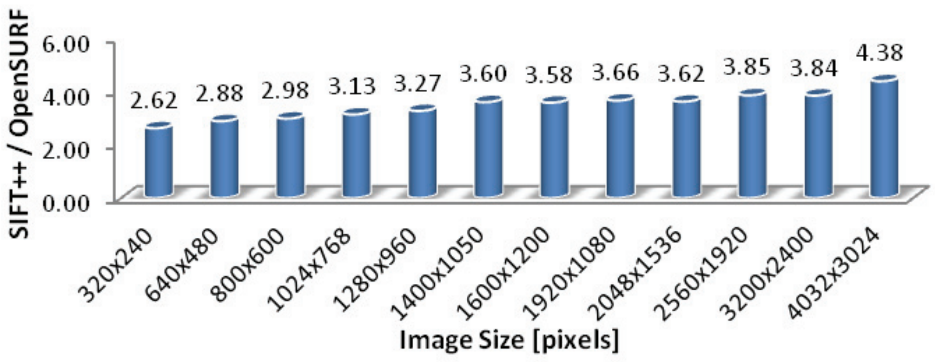


(b) SIFT++

Figure 2. Results of feature detectors OpenSURF and SIFT++



(a) Computational time required to process all test images



(b) Relative processing speed of SIFT++ with respect to OpenSURF

Figure 3. Process time comparison between SIFT++ and OpenSURF

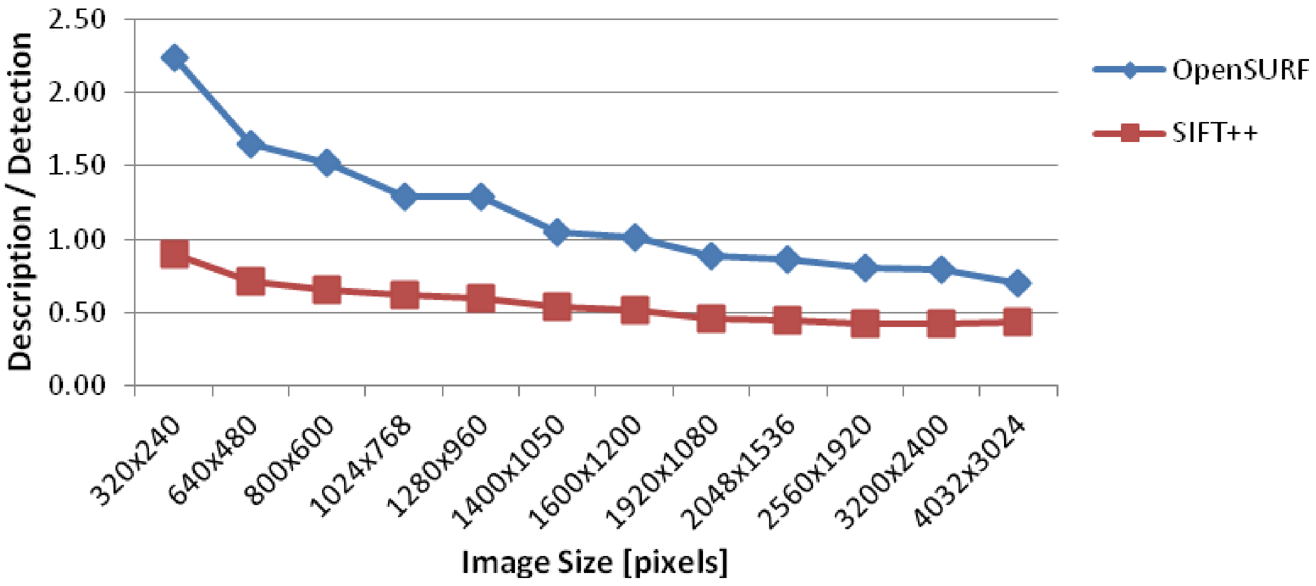
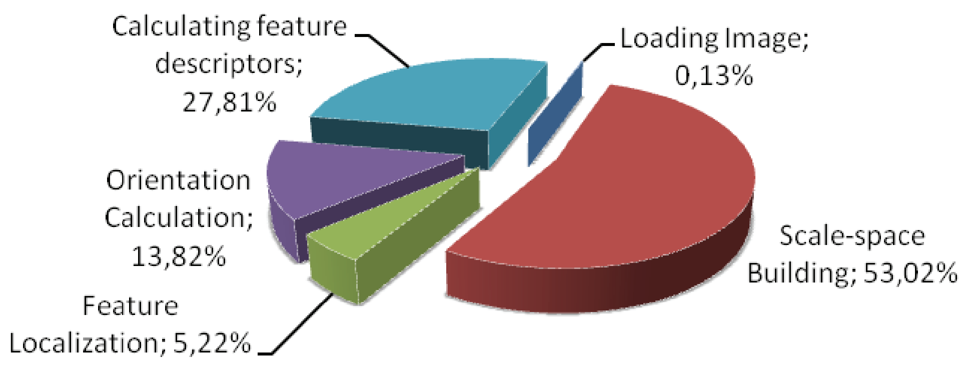
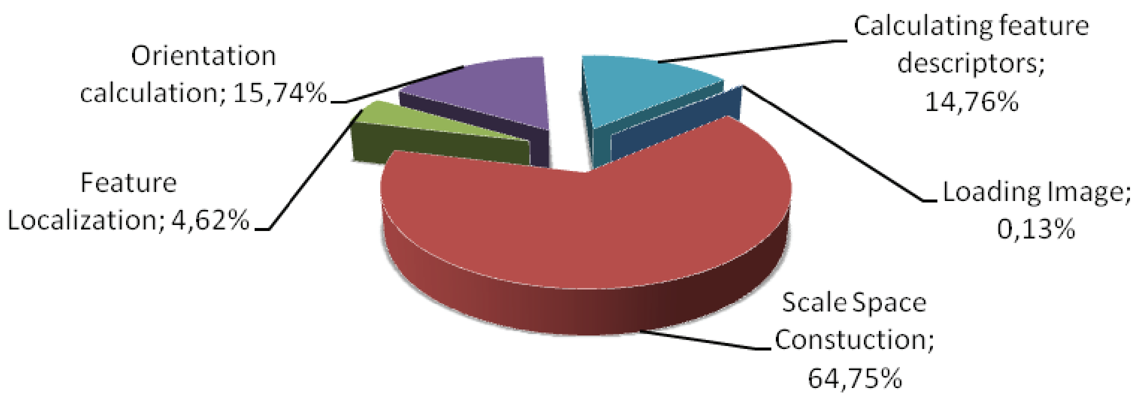


Figure 4. Process time ratio for *Description* and *Detection* stages of SIFT++ and OpenSURF

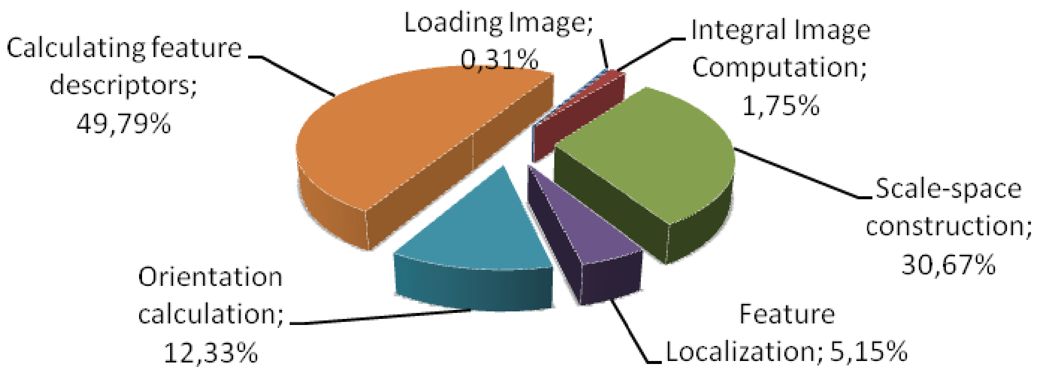


(a) For image of size 640x480 pixels

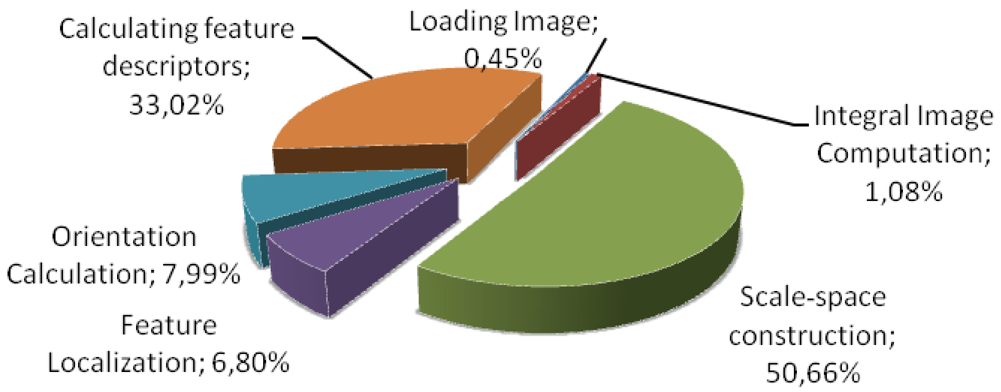


(b) For an image of size 4032x3024 pixels

Figure 5. Computational time percentage for each step of SIFT++ for varying size images



(a) For image of size 640x480 pixels



(b) For an image of size 4032x3024 pixels

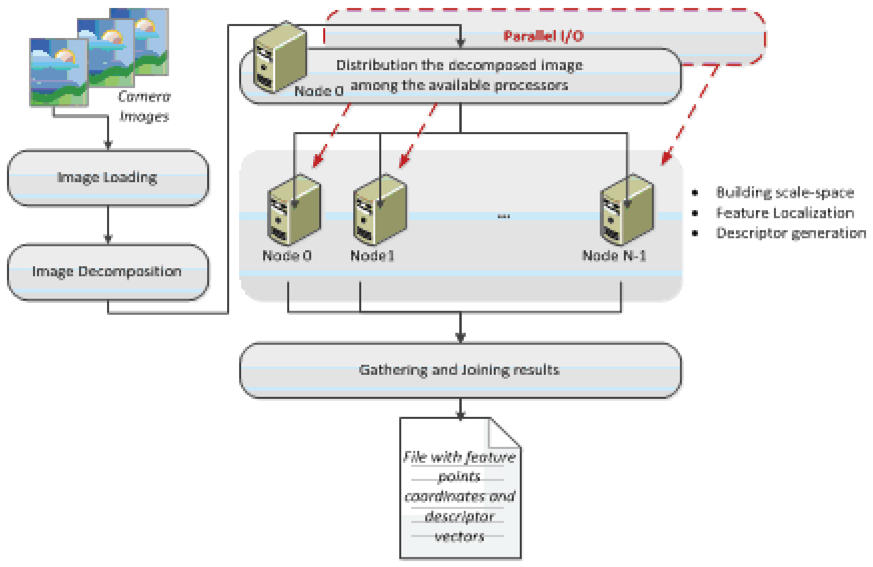
Figure 6. Percentage of the computational time for individual steps of OpenSURF for images of varying sizes

fundamentally different ways of implementation of this step, adopted in the two algorithmic solutions. Whereas in SURF the formation of the scale-space is based on the original image filtered with rectangular (box) filters of increasing size, the image pyramid in SIFT is built by repetitive upscaling of the original image, followed by filtering of the new image with a Gaussian filter of an appropriate size. Thus, the construction of the different levels of the image pyramid in SIFT is carried out sequentially, each subsequent level being built on the previous one. In SURF, the levels are generated simultaneously, which significantly improves its performance. The use of an integral image description in SURF further boosts the time decrease at this stage, reducing the computational complexity of the "convolution" operation and thus speeding up the filtering process.

The time of execution of the *Description* stage is determined mainly by the number of the detected features. Larger image size and bigger number of features require longer processing time. As it can be seen from the results, illustrated in *figures 5* and *6*, the steps related to the calculation of the *orientations* and the *descriptors* of the detected features take a relatively more time for smaller images in both algorithms. Furthermore, even if the generated output sets of features for both algorithms are numerically comparable in a percentage ratio, this stage is almost two times longer in SURF than in SIFT. As shown in *figure 4*, the same trends are observed for different image sizes. The time ratio of the Description/Detection stages for both algorithms is almost the same for the largest size images (*figure 4*). In SURF this ratio is bigger than SIFT for smaller images as a result of the more efficient scheme of scale space building that reduces the computation costs for Detection stage. However, unlike SURF, in SIFT the computational complexity remains consistently high, regardless of the processed image size. As can be seen in *figure 3 (a, b),* SURF is much more efficient with respect to the processing speed. The benefit of SURF is more than 2.5 times for the lower resolution images and the time decrease is more than four times for maximum resolution images.

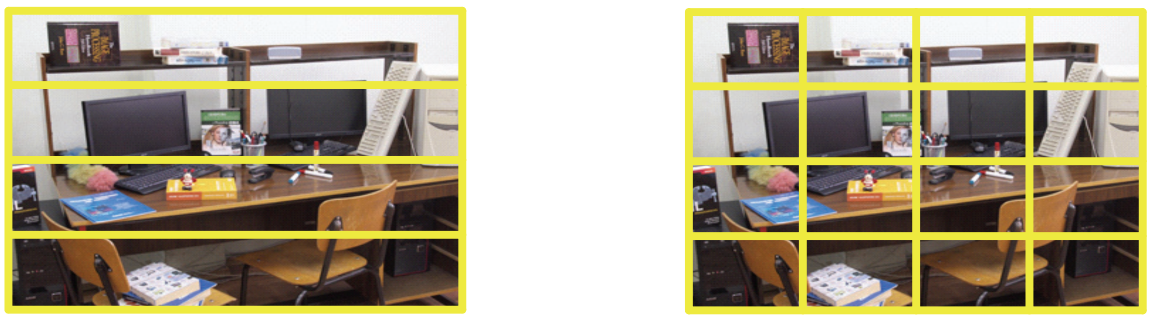
The analysis of the experimental results of serial implementations of SIFT and SURF algorithms leads to the following conclusions:

• The total execution time for SIFT and SURF depends on the image resolution and the number of the image features being detected. For small image sizes (VGA image, 640x480 pixels), the execution time is approximately 0.97 s for SIFT++ and 0.34 s for OpenSURF and for the larger image resolution (12 Mpx, 4032x3024) – 32.6 s for



Camera Images
Image Loading
Image Decomposition
Parallel I/O
Distribution the decomposed image among the available processors
Node 0
Node 0
Node1
...
Node N-1
Building scale-space
Feature Localization
Descriptor generation
Gathering and Joining results
File with feature points coordinates and descriptor vectors

Figure 7. Parallel computational model with data decomposition; SPMD parallel paradigm



(a) row-wise decomposition

(b) block-wise decomposition

Figure 8. Types of image decompositions

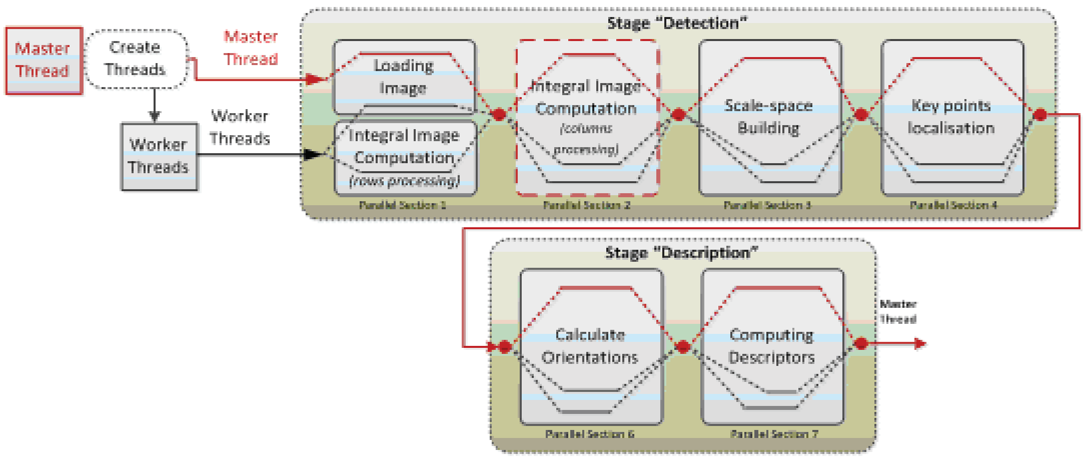


Figure 9. Multithread parallel computational model

SIFT++ and 8.1 s for OpenSURF.

• The largest amount of time in SIFT is spent in computing the Gaussian scale-space (*Feature Detection* stage), followed by the time required in computing the gradient direction histograms (for the *Feature Description* stage). In SURF the computations, related to the scale-space construction utilizes fast approximations using integral images and square (box) filters, whose computational cost is independent of the filter size due to the integral image representation. SURF employs Haar wavelets for the *Feature Description* stage. Thus, SURF algorithm is much faster than SIFT. Nevertheless, the computational cost of the *Feature Detection* and *Feature Description* stages is generally high for both algorithms.

All these results prove the necessity to continue the research aimed to investigate the opportunities for improving the speed of the computational process, using parallel computational models and parallel computing platforms.

3. Acceleration Opportunities for SIFT and SURF

The performance of the investigated feature detection algorithms can be improved by assessing the possibilities of optimization of the program codes of their implementations. The selection of the steps for optimization is performed with the help of code profilers. The profiling analysis of the serial implementations of SIFT++ and OpenSURF with GNU *gprof* profiler has shown three functions, where the optimization efforts should be focused on. These take more than 87% of the total computational time in SIFT++ and 90% of the time in Open-SURF: (1) the scale-space building (econvolv at SIFT++ and buildResponseLayer at Open-SURF); (2) the calculation of the gradient orientations (prepareGrad at SIFT++ and getOrientation at OpenSURF) and (3) the descriptors calculation (computeKeypointDescriptor at SIFT ++ and getDescriptor at OpenSURF).

There are several techniques for performance improvement that range from serial code optimizations to exploiting the potential parallelism of the algorithms' steps and adopting parallel computational models. Both approaches require several types of activities. The first one is focused on the elimination of the unnecessary operations. This includes changes of some data and code structures – efficient data restructuring, data rearrangements; memory access optimization – loop optimizations to improve the cache locality and to eliminate the data dependency, redundant memory copy operations elimination, unnecessary function calls elimination, conditional tests, and memory references, etc. The performance can be further improved by applying a set of optimizations on a compiler level and thus generating a more efficient code. As discussed in Section 3, both the investigated implementations are compiled invoking GNU g++ compiler with the command-line flag -O3 that permits more extensive optimizations.

The second step in code optimization is to exploit the capability of the processors to provide instruction-level parallelism by executing multiple instructions simultaneously [6,9]. The most time-consuming step *Scale-Space Building* in both of the algorithms may benefit of SIMD (Single Instruction Multiple Data) technology and the usage of SSE/SSE-2 instructions to reduce the payload of the memory subsystem [10].

Besides the serial code performance optimization, a third technique for dealing with the time demanding steps of the workflow of SIFT and SURF algorithms is to divide the computational work into portions that can be computed in parallel, on a multicore and multiprocessor machines and/or utilization of graphics accelerators (GPGPU based platforms). These types of optimizations depend on the specific characteristics of the computing machinery to a significant extent and thus the designed parallel computational models have to be consistent with this specific. Both coarse-grained and fine-grained parallel strategies can be utilized for SIFT and SURF algorithms. The coarse-grained approach is particularly important for processing high-resolution images and especially in the case where the amount of data is much bigger than the available memory in the computing node. A data-parallel model of computations based on the parallel paradigm Single Program Multiple Data (SPMD) is illustrated in *figure 7*. It requires the decomposition of the processed image into parts distributed for processing on different nodes. Two types of decomposition can be utilized: row-wise or block-wise (see *figure 8*). In the second case, the number and the size of the blocks are determined based on the image resolution and the available computing nodes.

The decomposed input image is distributed for processing by the main node (Node 0) to all computing nodes, where a process that runs the algorithm over the different data parts is launched. After processing completion, all the nodes send the detected number of the feature points and their descriptors back to the main node to form an overall solution. An alternative approach is to apply the concept of parallel input/output (see the red block with dashed lines in *figure 7*), wherein any of the processes running on the computing nodes reads a part of the decomposed image in parallel with the rest.

The fine-grained approach can be applied to any or all of the processing steps in the investi-gated algorithms splitting the code in many small sub-tasks. In *figure 9* multithreading model of parallel computation is illustrated that suggests all the steps of the computational workflow to be parallelized. For OpenSURF, additional optimization is made for the *Integral Image Computation* step.

The sequential algorithm for this step [17] follows a two-pass computational procedure of prefix (cumulative) sum calculations of the pixel values in a given image: the first pass along the rows and then the other pass along the columns, based on the results of the first pass (see *figure 10*). As it can be seen, there are mutual data dependencies between the operations, which complicate

```
for each row < ImageHeight do
  row_sum = 0.0;
  for each col < ImageWidth do
    // Row Independent Calculations
    row_sum += ImageData(row,col);
    IntegralImage(row,col) = row_sum;
    if row > 1 do
      // Column Independent Calculations
      IntegralImage(row,col) += IntegralImage(row-1,col);
```

Figure 10. Pseudo code of the serial integral image computations

```
// Do parallel for all rows (PASS 1)
for all rows do
  row_sum = 0.0;
  for each col < ImageWidth do
    row_sum += ImageRow(row,col);
    IntegralImage(row,col) = row_sum;
// Do parallel for all columns (PASS 2)
for all cols do
  for each row < ImageWidth do
    IntegralImage(row,col) += IntegralImage(row-1,col);
```

Figure 11. Pseudo code of the parallelized integral image computations

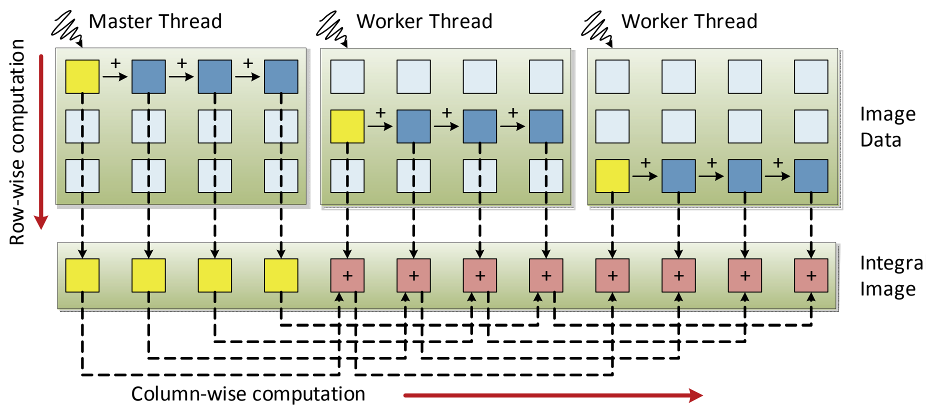


Figure 12. Scheme of the parallel integral image calculations

the parallelization of this step. Therefore, it is suggested the computations to be reorganized (see *figure 11*).

The calculation process divides the computations into two sub-steps (PASS 1 and PASS 2), that are executed sequentially after each other. The first step (PASS 1) performs the parallel row-wise calculations (see *figure 12*, black thick lines) and the second (PASS 2) – the parallel column-wise calculations (see *figure 12*, black dashed lines). Since the serial row-wise image input data scan of the matrix is done in the memory, it is possible to speed up the processing by simultaneous calculation of the integral image during the row-wise image scan. Thus the column-wise sum will be completed after the image is entirely scanned.

4. Experimental Results of Parallel Computational Models

The experimental studies discussed in this section concern only implementations of parallel computational models of SURF algorithm, since its algorithmic scheme is well optimized with respect to memory usage on the computing node and the processing speed, as opposite to SIFT.

Two parallel implementations of tSURF algorithm are analyzed: 1) a multithreading parallel computational model running on a multicore SMP platform, using low-level thread

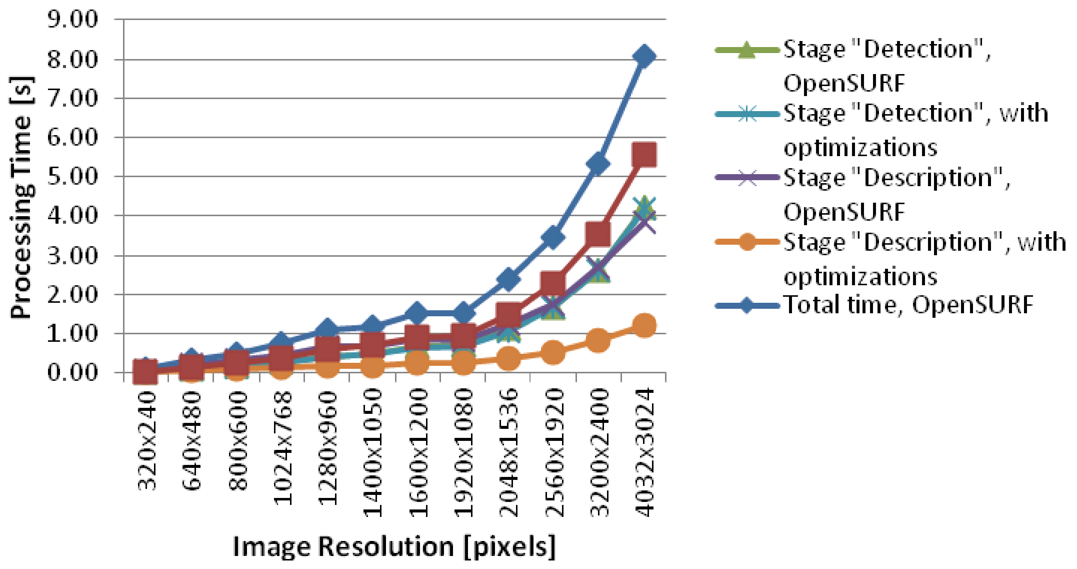


Figure 13. Process Time Comparison between OpenSURF and MT-SURF

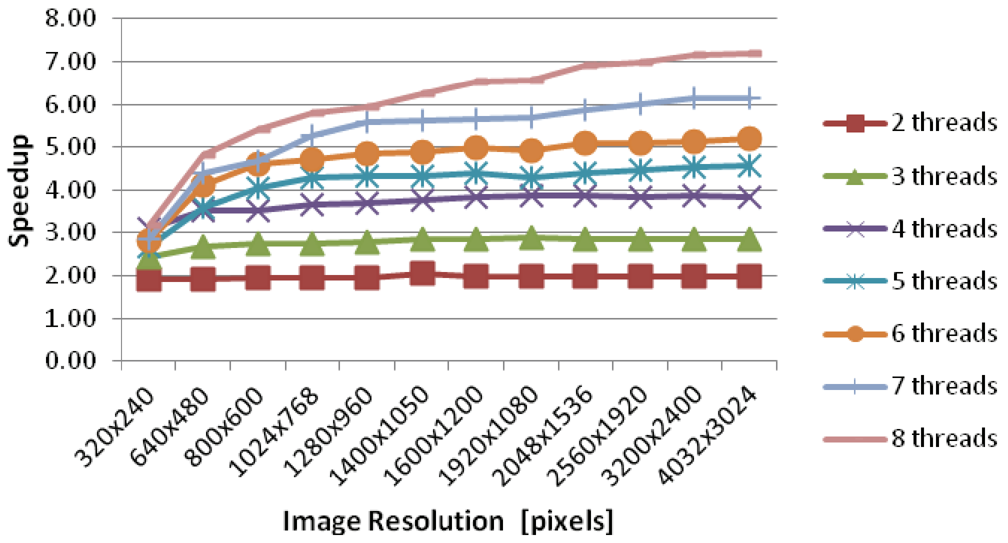


Figure 14. Results of the scalability analysis of MT-SURF on a SMP platform

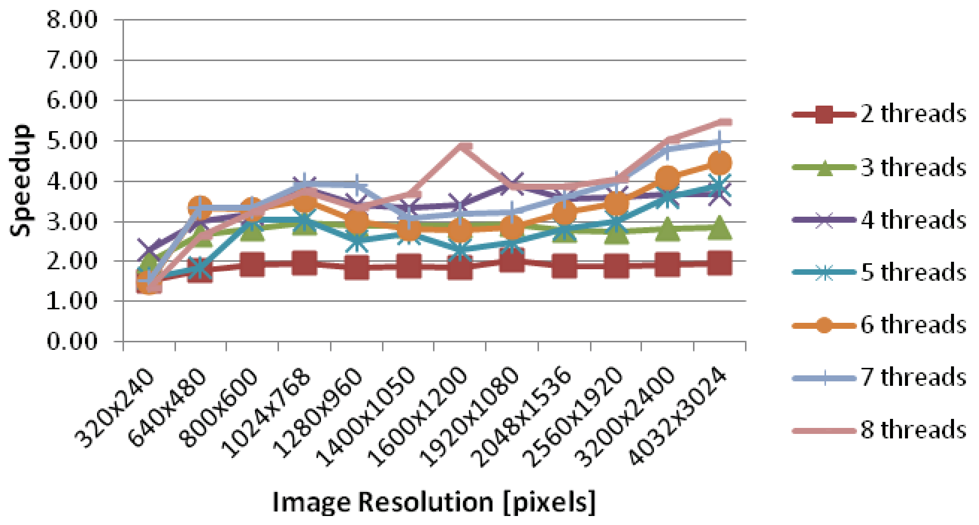


Figure 15. Speedup for Steps *Image Loading* and *Integral Image Calculation – processing image rows*

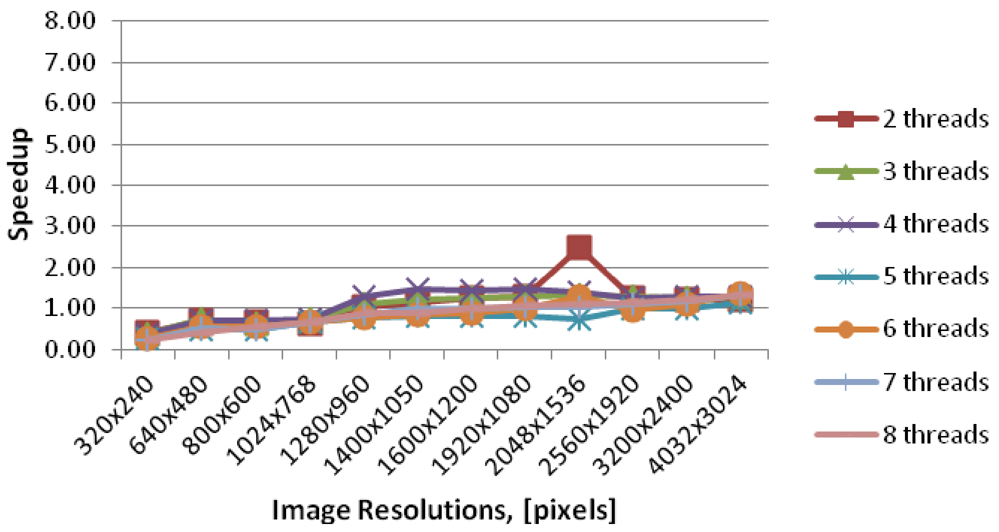


Figure 16. Speedup for Step *Integral Image Calculation – processing image columns*

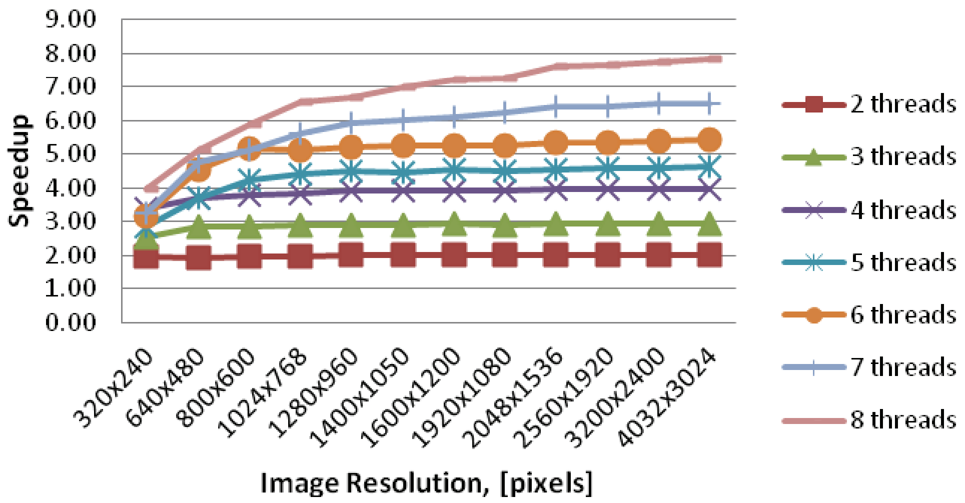


Figure 17. Speedup for Step *Building Scale-space*

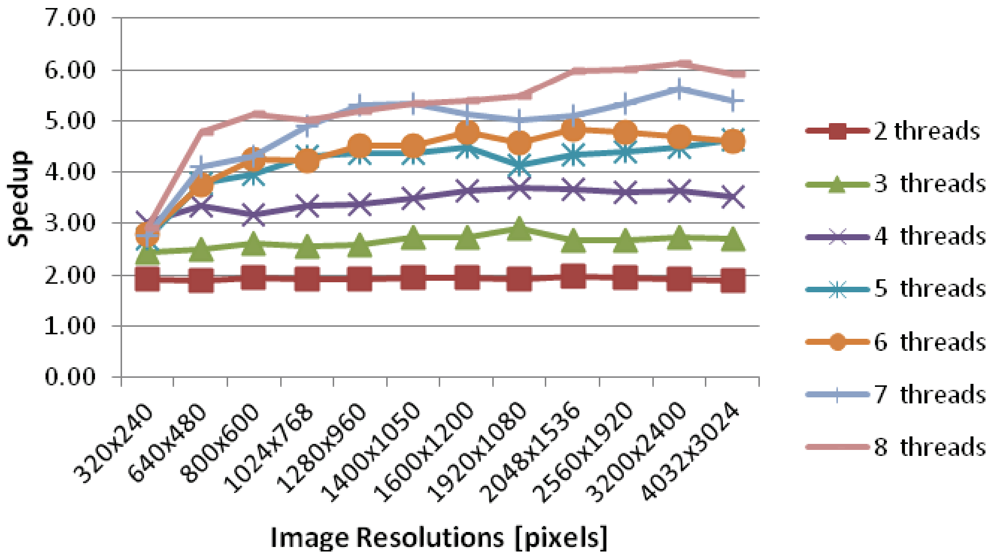


Figure 18. Speed for Steps *Calculate Orientations* and *Computing Descriptors*

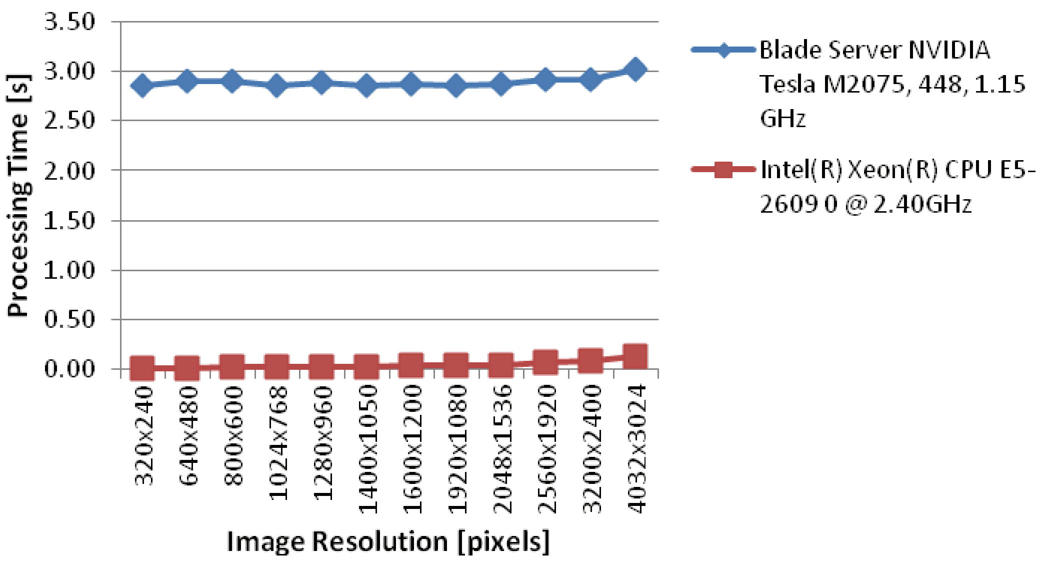


Figure 19. Parallel processing time with CUDA-SURF on a GPGPU Blade Server NVIDIA Tesla M2075

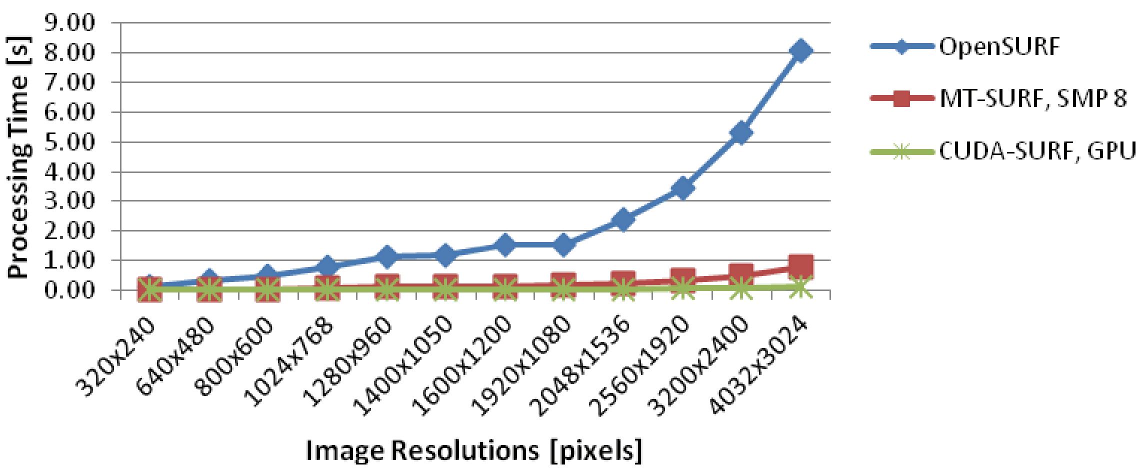


Figure 20. Process time comparison between OpenSURF, MT-SURF and CUDA-SURF

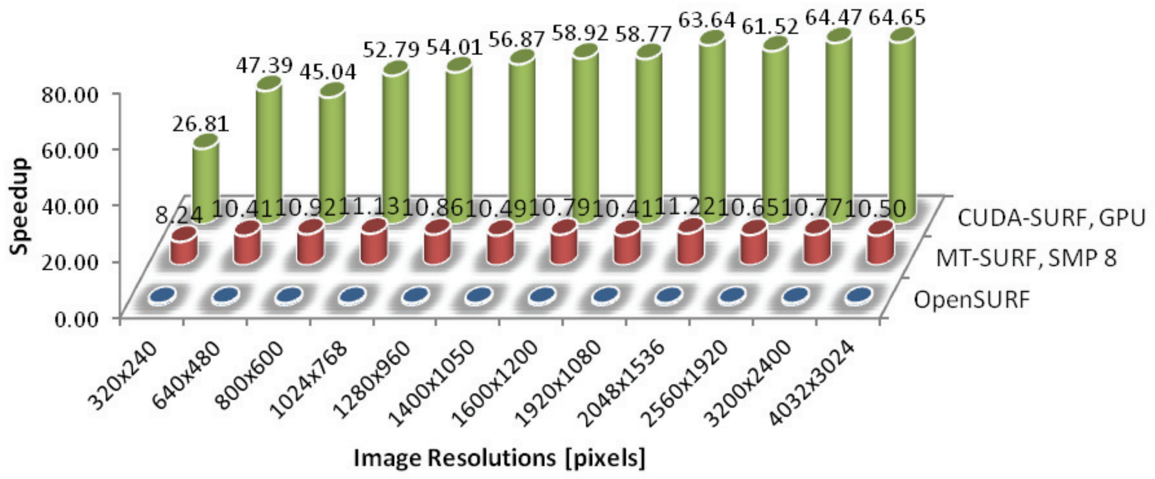


Figure 21. Speedup comparison between MT-SURF and CUDA-SURF using OpenSURF as a basis

Table 2. Percentage reduction of the computational time with MT-SURF

*	1	2	3	4	5	6	7	8	9	10	11	12
Sw	61,52	53,85	50,33	47,98	45,36	40,51	39,48	37,08	38,43	34,31	33,56	31,49
So	76,35	74,37	73,05	72,62	72,13	70,50	70,04	69,63	70,35	69,32	69,28	68,55
Sd	23,84	14,99	10,62	11,75	5,17	4,15	3,22	3,27	6,43	0,97	0,34	0,59
***Legend**	1: 320x240; 2: 640x480; 3: 800x600; 4: 1024x768; 5: 1280x960; 6: 1400x1050						7: 1600x1200; 8: 1920x1080; 9: 2048x1536; 10: 2560x1920; 11:3200x2400; 12: 4032x3024					
	Sw: Percentage of reduction of the total processing time So: Percentage of reduction of the time for stage *Feature detection* Sd: Percentage of reduction of the time for stage *Feature description*											

libraries [3] in combination with OpenMP API [2]; 2) CUDA based parallel computational model running on the graphics processor platform NVIDIA Tesla M2075 [5]. The utilized implementation for MT-SURF is based on the algorithm proposed in [18], and for CUDA-SURF the solution found in [12].

The aim of the experimental test is to measure the performance of both computational models for images with different resolutions processed on each of the two platforms in terms of the processing speed and the efficiency of both computational models.

4.1. Experimental Analyses of Multithreaded Parallel Computational Models

The experimental results of the model shown in *figure 9* are presented in *figure 13*, and the efficiency gained is given in *table 2*.

As seen, the overall improvement in the duration of the total processing time is approximately 43% ± 9% and it depends on the image resolution. The average improvement for the detection step is within 71% ± 2%, and for the description step it is about 7%. The minimum values are achieved for the smallest resolution image: less than 1%.

The experimental results concerning the scalability test of the multithread parallel computational model are given below. The data given are averaged for 20 runs for each test image. The speedup of the multi-threaded parallel model is shown in *figure 14*. The speedup is calculated as a ratio of the sequential execution time and the time of the parallel model using several threads (up to 8 for the experimental platform). The maximum speed up for steps *Image Loading* and *Integral Image Calculation – integration by rows* of approximately 3 times (2,85 ± 0,84) is achieved for the test images of resolution between 320x240 and 1280x960 pixels utilizing 8 threads (*figure 15*). The speedup for the megapixel formats (between 1400x1050 and 4032x3024 pixel) running on the 8 core machine, is 4.4 ± 0.65. The remaining three steps of the algorithm are shown in *figures 16, 17* and *18* respectively. The results show that the step of the integral image construction does not scale well because of the column-wise image data matrix scan that will lead to intensive memory reads. In contrast, the distribution of the computing work between multiple threads at the stage of the scale-space construction shows better scalability results. The speedup for the megapixel formats reaches 7.49, utilizing 8 threads. The speedup of the parallel computational model at the last step of feature descriptors calculation is 5.74 for 8 threads, which may be explained by the inefficient second level cache operations and increased bus traffic as a result of the usage of linked lists.

4.2. Experimental Analysis of a CUDA Based Parallel Computational Model of SURF

CUDA based parallel computational model consists of (1) Memory allocation on GPU and image transfer from the main memory to the device memory; (2) GPU based computing of the integral image representation of the input image; (3) GPU based calculations for generation of the Gaussian Scale Space, local extremes search and 3x3x3 non-maximum suppression; (4) Down-load of the detected feature points to a host and free the device memory; (5) GPU and CPU based calculations of the orientations and (6) Descriptors construction for each feature.

The experimental results for the overall processing time (in milliseconds) of the parallel CUDA-SURF implementation are given in *table 3*, averaged over 20 runs for each tested image.

The processing times for GPGPU platform (host + graphics device), as well as the processing time of the kernels on the GPU only, are shown in *figure 20*.

The results show lack of significant speedup when a single image is processed due to the overhead communication delays for data transfer between the host and the device memory. When a large amount of image data has to be processed by the computationally intensive SURF algorithm, the additional communication time is negligible compared to the computational time. It is evident (see *table 3*) that the data loading time does not change significantly with the image size, and thus, with the large image resolution the efficiency of GPU accelerated processing becomes apparent.

The results of the comparative analysis of the processing time for the three selected models and their implementation in OpenSURF, MT-SURF and CUDA-SURF are given in *figures 20* and *21*.

5. Conclusions

In this paper two popular contemporary algorithms for discrete feature detection and description SIFT and SURF were analyzed. On the basis of the timing performance

Table 3. Computational times at different stages of processing in ms with CUDA-SURF on a Blade Server NVIDIA Tesla M2075

*	1	2	3	4	5	6	7	8	9	10	11	12
C1	0,36	0,97	1,49	2,24	3,31	4,10	5,01	5,53	7,99	12,53	19,10	29,99
C2	2862,83	2906,07	2902,42	2847,35	2866,09	2845,05	2844,68	2842,14	2835,71	2867,88	2845,98	2905,36
C3	0,16	0,34	0,51	0,77	1,14	1,33	1,65	1,84	2,60	3,84	5,76	8,96
C4	0,30	0,71	0,81	0,99	1,53	2,20	2,52	1,84	3,31	4,64	6,87	10,08
C5	1,33	1,98	2,55	3,76	4,94	5,50	7,40	7,20	10,80	17,06	24,76	39,74
C6	1,31	2,58	3,56	4,63	6,51	6,49	8,44	7,72	11,81	17,05	25,56	35,52
***Legend**	1: 320x240; 2: 640x480; 3: 800x600; 4: 1024x768; 5: 1280x960; 6: 1400x1050						7: 1600x1200; 8: 1920x1080; 9: 2048x1536; 10: 2560x1920; 11:3200x2400; 12: 4032x3024					
	C1: Loading images in the host memory (CPU) C2: GPU initialization – memory allocation C3: Image data transfer from main memory to the device memory C4: Scale space building C5: Building scale-space and feature localization C6: Computing features descriptions											

analysis of the sequential implementations of both algorithms, the most time-consuming steps for each stage of processing: detection and description of the feature points were determined. Two parallel computational models were analyzed. The experimental results demonstrate the efficiency of the models in providing a large number of features for a short time. The results of the fine grain implementation reveal when these models are efficient and what is the expected benefit of their implementation on more sophisticated hardware, such as GPGPU. The obtained results of the performance speed analysis and scalability tests with multithreading and GPU-based implementations are presented.

Acknowledgment

This paper is partially supported by the internal competition of the Technical University of Sofia under Grants 2014 – 141PD0015-09.

References

1. Bay, H., T. Tuytelaars & L. Van Gool. Surf: Speeded up Robust Features. Computer Vision–ECCV 2006, Springer Berlin Heidelberg, 2006, 404-417.

2. Chapman, B., G. Jost & R. Van Der Pas. Using OpenMP: Portable Shared Memory Parallel Programming. – *MIT Press*, 10, 2008.

3. Drepper, U. & I. Molnar. The Native POSIX Thread Library for Linux. White Paper, Red Hat Inc, 2003.

4. Evans, C. Notes on the OpenSurf Library. University of Bristol, Tech. Rep. CSTR-09-001, January 2009.

5. Glaskowsky, P. N. NVIDIA's Fermi: the First Complete GPU Computing Architecture. White Paper, 2009.

6. FAMILY, IBM PowerPC Microprocessor.Vector/SIMD Multimedia Extension Technology Programming Environments Manual, 2005.

7. Fenlason, J. & R. Stallman. GNU gprof: the GNU Profiler. Manual, Free Software Foundation Inc, 1997.

8. Hartley, R. & A. Zisserman. Multiple View Geometry in Computer Vision. Cambridge University Press, 2003.

9. Hennessy, J. L. & D. A. Patterson. Computer Architecture: A Quantitative Approach. Elsevier, 2012.

10. Intel Corporation. Intel 64 and IA-32 Architectures Optimization Reference Manual, 2009.

11. Lowe, D. G. Distinctive Image Features From Scale-Invariant Key Points. – *International Journal of Computer Vision*, 60 (2), 2004, 91-110.

12. Schulz, A., F. Jung, S. Hartte, D. Trick, C. Wojek, K. Schindler & M. Goesele. CUDA SURF – a Real-time Implementation for SURF, 2010.

13. Spivey, J. M. Fast, Accurate Call Graph Profiling. Software: Practice and Experience, 34 (3), 2004, 249-264.

14. Szeliski, R. Computer Vision: Algorithms and Applications. Springer, 2010.

15. Tuytelaars, T. & K. Mikolajczyk. Local Invariant Feature Detectors: A Survey. – *Foundations and Trends® in Computer Graphics and Vision*, 3 (3), 2008, 177-280.

16. Vedaldi, A. Sift++ Source Code and Documentation [online], 2009. www.robots.ox.ac.uk/~vedaldi/code/siftpp.html.

17. Viola, P. & M. J. Jones. Robust Real-time Face Detection. – *International Journal of Computer Vision*, 57 (2), 2004, 137-154.

18. Zhang, N. Computing Parallel Speeded-up Robust Features (P-SURF) via POSIX Threads. Emerging Intelligent Computing Technology and Applications, Springer Berlin Heidelberg, 2009, 287-296.

Manuscript received on 14.07.2014

Iva Nikolova *is an assistant professor of computer systems engineering at Technical University of Sofia. She received a M.Sc. degree in Computer Systems and Technologies from the Department of Computer Systems at the Faculty of Computer Systems and Control. The major fields of her professional and scientific research interests include digital image processing, parallel computer architectures, high-performance computer systems, parallel algorithms, parallel programming.*

Contacts:
Technical University of Sofia
inni@tu-sofia.bg