Rafał Frączek
Bogusław Cyganek
Kazimierz Wiatr

PARALLELIZED ALGORITHMS FOR FINDING SIMILAR IMAGES AND OBJECT RECOGNITION

Abstract   *The paper addresses the issue of searching for similar images and objects in a repository of information. The contained images are annotated with the help of the sparse descriptors. In the presented research, different color and edge histogram descriptors were used. To measure similarities among images, various color descriptors are compared. For this purpose different distance measures were employed. In order to decrease execution time, several code optimization and parallelization methods are proposed. Results of these experiments, as well as discussion of the advantages and limitations of different combinations of methods are presented.*

The problem of finding similar images or objects in them is important because as it turns out, many applications in industry, entertainment and digital library require access and query of image data sets. This access can be performed using a text based queries (e.g. find images with a given tag). Another possibility is to access the image database using a reference image. In this approach, the user may want to find an image that is the same or similar in some way to the reference image (QbE – Query by Example). Another possibility in this scenario is that a user wants to find all images containing a specified object (e. g. a car, a building, etc.). This type of systems are based, in most cases, on features extracted from the media. Sets of features are generally referred to as "descriptors" and their instances are called "descriptor values". The descriptor values are the meta-data of the media.

Over the past years, several methods of Query by Example have been presented. For instance, the VICTORY project [14] is a good example of advanced research in the area of QbE – in this case focusing on searching for 3D objects. Another example of a system for images retrieval is a program called Picture Finder [9] which is a fast image retrieval library software for image-to-image matching. Similarity is measured based on spatial distribution of color and texture features. It is especially optimized for fast matching within large data-sets.

In the case of objects recognition, several methods have been proposed so far. Most of them are based on the concept of "points of interest" which was introduced by Moravec [17]. Lowe [13] has proposed a method named SIFT (Scale Invariant Feature Transform) which is based on selecting stable features in the scale space domain. Alhwarin et al. [1] have proposed a method for efficient SIFT features matching which is based on determining the scale factor of the target object in the test and reference image. Duchenne and Bach [4] have proposed a tensor-based algorithm for SIFT features matching. The method uses tensors to solve the high-order feature matching problem.

In [5] we present our previous solution for finding similar images. The method described in this paper is an extension to the previous one. The difference lies in employing new image descriptors, adding the possibility of finding objects in images and application of several optimization and parallelization techniques using the OpenMP [3] standard.

1. Goals of our research

Our work is focused on both finding similar images and particular objects in the whole scene depicted in an image. When developing a system for accomplishing this goal, a question arises: which descriptors and which distance measures should be applied for the fast and most accurate results. We address this question as well and try to discuss results of our research in the area of QbE. We examine several color descriptors, especially most common variants of the SIFT descriptor as well as mpeg-7 [16] descriptors.

Another aspect of content based images retrieval is choice of the most appropriate distance (similarity) measure [6] [20]. This is a big challenge in the case of SIFT descriptor because it is composed of variable number of key points with assigned feature vectors. This representation enforces non-trivial approaches for descriptors matching. To address this problem, two possible methods of similarity measure for SIFT descriptors were examined.

Other very important aspect is the effectiveness of the applied algorithms in terms of execution time. As it turns out, the procedure of computing and matching SIFT features can take long time and therefore it is necessary to optimize its performance using some techniques of code optimization and parallelization.

2. Brief description of the algorithm
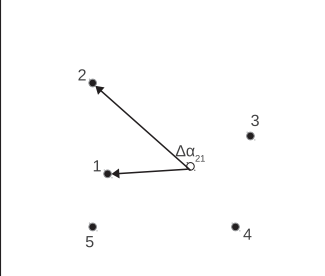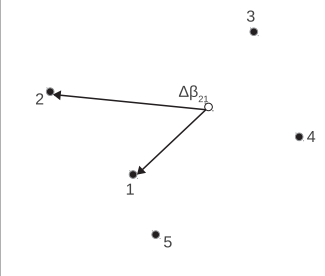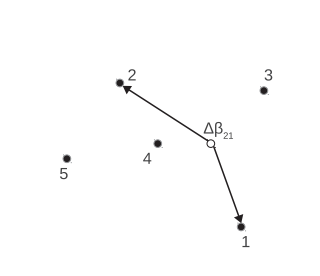
Image features extraction and classification

The SIFT algorithm was originally proposed by David G. Lowe. It is a method for extracting distinctive and invariant features form images. The descriptor has been successfully used in a variety of computer vision problems. SIFT descriptor extracts so called key points in an image. Each keypoint has assigned a vector of features describing the distribution of local gradients in the nearest neighborhood of a given key point. In short, the extraction procedure runs as follows: scale-space image filtration, localization of the extreme points (keypoints), assigning the descriptor values to each key point. In our research, we also use different variants of the SIFT algorithm such as the Opponent SIFT, PCA-SIFT, etc. The difference is that they use other color spaces for computation (e. g. OpponentSIFT).

One of the most important drawbacks of the SIFT descriptor is its high dimensional feature space. One of the possible solutions is the application of the Principal Component Analysis [11]. PCA is most widely used method for dimensionality reduction. PCA has also been successfully applied in many various image processing problems including object recognition [18] or features selection [7]. Although PCA is based on the assumption of Gaussian distributions, it is shown [21] that this method is well-suited to representing SIFT keypoints patches and can significantly improve SIFT matching performance in some cases.

We also use two descriptors that belong to the mpeg-7 standard. The Edge Histogram [15] descriptor computes the spacial distribution of all edges in the image. The descriptor is represented by a vector of 80 values. In turn, Dominant Color is a compact descriptor that calculates representative colors and their percentages. The extraction procedure is based on the Generalized Lloyd Algorithm (GLA) [8].

If we want to compare two images, we need to compute a distance between two descriptors. In the reported experiments we have employed and tested two approaches for calculating the distance between two SIFT descriptors. Simple Graph Matching (SGM) consists in searching the best matches for each feature vector in the query image. In the case of mpeg-7 descriptors, the appropriate metrics [2][22] described in the literature were used.

Table 1

Computation of similarity metric.

reference keypoints	similar distribution	different distribution

Topology measure

In order to make object recognition less sensitive to keypoints mismatching, we propose to add a special simple measure that tells how much two sets of matched keypoints are similar in terms of their spatial distribution. The basic idea here is that if two points P_{11} and P_{12} of image 1 are matched to points P_{21} and P_{22} of image 2, then the geometric relation between P_{11} and P_{12} and the one between P_{21} and P_{22} should be similar. The main idea of our proposition is explained in the Table 1.

The procedure for calculating the similarity measure is as follows. Some keypoints from the reference set are matched with some keypoints in the current image. This way two subsets are obtained. Each keypoint in the A subset has assigned a keypoint in the B subset. Matched keypoints are denoted with the same index in the table. The mass center is computed for each of the two subsets. As it is shown in the table, for two subsequent keypoints, vectors originating in the mass center are computed. Then the angle $(\Delta\alpha_{ij})$ between this two vectors is calculated. The same procedure is also applied to the subset in the current image resulting in the angle $\Delta\beta_{ij}$. If the spatial distribution of keypoints in the subsets A is the same or similar to the spatial distribution in the subset B, then the difference between angles $\Delta\alpha_{ij}$ and $\Delta\beta_{ij}$ will be very small. Therefore the total similarity measure is proposed, as denoted in equation 1. The S measure is near 1.0 if two subsets are similar in terms of spatial keypoints distribution and near 0 if the spatial distributions are significantly different. The proposed similarity measure is translation, rotation and scale invariant.

$$S = 1 - \frac{1}{2\pi N} \sum_{i=1}^{N-1} ||\Delta\alpha_{i,i-1}| - |\Delta\beta_{i,i-1}|| \qquad (1)$$

Code optimizations
Parallelization

In order to boost the code performance in terms of execution speed, some implementation enhancements have been applied. The first possible optimization is to

introduce some improvements that will increase both spatial and time locality. These changes make it possible for the processor to fully utilize the cache memory resulting in lower data access latencies. The second possible enhancement is the introduction of parallelism in order to fully utilize modern multi-core processors. The goal of the parallelization is to transform some parts of the sequential code into a new structure that makes it possible to execute computations in parallel on multi-core processors or systems. This goal can be achieved by employing all extensions provided by the OpenMP library which makes it easy to parallelize time-consuming loops. In our work, we have used an advanced hyperplane method [19, 12] that utilizes possible dependencies among nested loops. In general, the transformation from the sequential code to the parallel one is not always an easy task and can also be time-consuming. Below, we briefly present two techniques that have been applied in our code in order to increase its performance.

Hyperplane method

One of the possible strategies for code parallelization (especially for loops) is to use so called hyperplane method. The key idea is to find a subset of all iterations that can be executed in the same time.

The dependency vector between two iterations $I = [i_1, i_2, \ldots, i_n]^T$ and $J = [j_1, j_2, \ldots, j_n]^T$ is defined as $D = J - I = [j_1 - i_1, j_2 - i_2, \ldots, j_n - i_n]^T$. One iteration depends on the other if the two iterations refer to the same memory address and one of the reference is a write operation. If D is lexicographically greater than a zero vector ($D > 0$) then iterations I and J satisfy the lexicographical order $I > J$.

In an n-dimensional space, a *hyperplane* is defined as a set of tuples:

$$\{(x_1, x_2, \ldots, x_n) \mid q_1 x_1 + q_2 x_2 + \ldots + q_n x_n\} \tag{2}$$

where $q_1, q_2, \ldots, q_n$ are called hyperplane coefficients and k is a hyperplane constant. Both q_i and k are rational numbers. A vector $q = (q_1, q_2, \ldots, q_n)$ defines a hyperplane family. Each hyperplane belonging to this family has the same hyperplane vector but different k value. For instance, in a two-dimensional space, a vector $q = (1, 0)$ defines a hyperplane family where two points belong to the same hyperplane as long as they have the same row index (column index does not matter).

Two points a_1 and a_2 belong to the same hyperplane if

$$q^T a_1 = q^T a_2 \tag{3}$$

If $qD > 0$ then all iterations I such that $qI = t$ belong to the same hyperplane and therefore can be executed in parallel.

Barrier optimization

Another transformation that has been applied was barrier optimization. Since it is shown [10] that barriers are critical to the performance of OpenMP programs, it is necessary to remove redundant barriers wherever possible. Barriers can prevent programs from getting maximum speed-up even if the parallel code covering is above 90%. Some possible strategies for removing redundant barriers include: synchronizing

only related threads, privatizing some variables in order to reduce the number of barriers, eliminating redundancy barriers, data-flow analysis and data dependence test to implement "doacross" parallelism. If there is no data dependence across the barrier, it is considered redundant. A dependency occurs when a program instruction refers to the data of a preceding statement. To determine whether there is such a dependency or not, a test can be performed. The idea is to tell whether there exists a solution to a set of dependency equations and inequities. For instance, let us consider a code in the Table 2.

Table 2

Code sample.

```
#pragma omp parallel
#pragma omp for
for(j=0;j<M;j++) vec{[}2{*}j{]}=...;
#pragma omp for
for(j=0;j<M;j++) vec{[}2{*}j+1{]}=...;
```

If there exists a data dependence between the implicit barrier after the first "for" loop, then there exist an j_1 in the first loop and j_2 in the second one that will satisfy the following equality and inequality.

$$2 \cdot j_1 = 2 \cdot j_2 + 1, \quad 0 \leqslant j_1 \leqslant M, \ 0 \leqslant j_2 \leqslant M \tag{4}$$

Since, there is no such two j_1 and j_2 that satisfy this equation, then there is no data dependency between these two loops and the barrier can be safely removed.

3. Experiments

Image features

In order to determine which descriptors and which distances are most suitable for finding similar images, we have conducted an experiment which goal was to measure the number of correctly identified similar images. First, the database was created. It consists of 2400 test images downloaded from the Flickr website with the accompanying user-generated keywords. All images in the database were randomly divided into two semantic categories, for example: cars, offices, flowers, scenery, sunset and cell phone. Each category is comprised of 400 images. We do not split the data-set into training and test subsets because we do not use classifiers that need separate training process. The result of searching the similar image is considered correct if the most similar image belongs to the same semantic category as the reference one. For tests, Edge Histogram (EHD), Dominant Color (DCD), SIFT, PCA-SIFT, OpponentSIFT descriptors were used. In the case of mpeg-7 descriptors (EHD, DCD), standard distance measures defined in literature are used. For SIFT and OpoponentSIFT, we use features matching that consists in finding the best match which runs as follows. Suppose we have the first SIFT descriptor of N key points and the

second SIFT descriptor of M key points. The distance between these two descriptors is calculated in the following way. For each feature vector in the first descriptor the nearest (D_1) and the second nearest (D_2) feature vectors in the second descriptor are found. The distance between two feature vector is computed as the Euclidean distance. If D_1/D_2 is greater that the optimal Lowe threshold then this match is acceptable. Otherwise, it is ambiguously matched and rejected as a correspondence. We assume the Lowe threshold to be equal to 0.6 which is the recommend value in [13]. If the match is acceptable, then the Euclidean distance between two feature vectors is added to the global distance between two images (descriptors). To ensure that the calculated distance has symmetric property $(d(x, a) = d(a, x))$, all feature vectors in the second descriptor are matched in the same way to feature vectors in the first descriptor. Finally, the global distance between descriptors is normalized by the number of matches.

In the case of PCA-SIFT, high-dimensional vectors are projected onto low-dimensional feature space. This projection is encoded as a patch eigenspace. In our experiments we use the eigenspace that is provided in [21]. This eigenspace was constructed in the following way. The first three stages of the SIFT algorithm are run on a diverse collection of images. This results in 21000 patches of 41x41 size. Each patch is centered at a key-point and the horizontal and vertical gradient maps are extracted. The feature vector is created by concatenating gradient maps and is composed of $2 \cdot 39 \cdot 39 = 3042$ elements. Finally, PCA is applied to the covariance matrix of these vectors. The top $n = 20$ eigenvectors are saved in a file and are used for the projection matrix for PCA-SIFT.

For evaluation purposes, we compute F-measure which is given by $F = 2 \cdot (precision \cdot recall)/(precision + recall)$. These metrics are defined as follows: $recall = tp/(tp + fn)$, $precision = tp/(tp + fp)$, where tp stands for "true positive", fn means "false negative" and fp is "false positive".

Table 3
Results for similar images searching.

descriptor	F-measure	Execution time
EdgeHistogram	0.82	23%
OpponentSIFT	0.80	91%
PCA-SIFT	0.80	100%
C-SIFT	0.77	91%
SIFT	0.76	91%
Dominant Color	0.60	63%

The goal of the next experiment was to check the effectiveness of our methods for finding specific objects in a set of images. In this experiment we use standard SIFT, Opponent SIFT and PCA-SIFT descriptors. We do not use mpeg-7 descriptors because they are not suitable for object recognition. We have taken 20 images

of different reference objects. For each object the corresponding reference SIFT features are computed. Then, another set of test images (500 items) with and without reference objects was taken and SIFT features were computed for all items in the set. Reference features are matched to the features associated with each test image. The matching procedure is the same as in the case of finding similar images. After the keypoints are matched, the spatial distribution similarity metric (1) is computed. After searching procedure, all test images are sorted in descending order against this metric. As a result, the image with the best match is at the top of all results. If the best match contains the reference object, the results is considered correct (true positive). If the best match does not contain the object, the result is false positive. In a few cases, it may happen that the best match contains the object but the SIFT features are not matched correctly. This happens when an image contains the object that is partially obscured or significantly transformed with an affine transform. This situation is considered false negative.
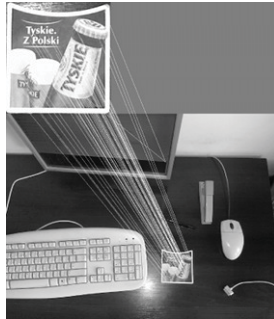

Figure 1. Example of finding an object in an image.

Table 4

Results for object recognition.

descriptor	F-measure	Execution time
PCA-SIFT	0.68	100%
OpponentSIFT	0.52	91%
SIFT	0.51	91%

Code optimization and parallelization

In order to properly use parallelization techniques, it is necessary to apply them to these parts of code which are computationally ineffective. For this purpose, our code was examined with profilers. These portions of the source code were parallelized with the hyperplane method. It turns out that one of the most time consuming operation is the computation of distances between feature vectors in both images. All tests were executed on the AMD FX-8120 X8 at 3.1 GHz with 8 MB KB L2 cache. The ope-

rating system was Linux Ubuntu with GCC 4.6.1 compiler. All developed software is implemented in C++ and compiled with maximum level of performance optimization (including usage of SIMD instructions). Below, a code snippet is presented.

Table 5

Original sequential code.

```
 for (unsigned int i=0; i<arg1->PointsCount; i++)
{for (unsigned int j=0; j<arg2->PointsCount; j++)
{unsigned char {*} __restrict__ irow=(arg1->features)+i{*}ncol;
 unsigned char {*} __restrict__ jrow=(arg2->features)+j{*}ncol;
 unsigned long local_dist=0;
 for (unsigned int k=0; k<arg1->FeaturesCount/3; k+=4)
{int v=(int)({*}(irow++)); int u=(int)({*}(jrow++));
 int v1=(int)({*}(irow++)); int u1=(int)({*}(jrow++));
 int v2=(int)({*}(irow++)); int u2=(int)({*}(jrow++));
 int v3=(int)({*}(irow++)); int u3=(int)({*}(jrow++));
 local_dist+=((u-v){*}(u-v)+(u1-v1){*}(u1-v1))+
 +((u2-v2){*}(u2-v2)+(u3-v3){*}(u3-v3));}}}
```

The code presented in table computes the total distance between two sets of SIFT features. It consist of three nested "for" loops. The code has been parallelized with the hyperplane method. In our experiments we have used two possible versions of loop parallel code after the sequential loop transformation.

Table 6

Parallelized code with k.

```
#pragma omp parallel private(i,j) shared (arg1,arg2)
for (unsigned int i=0; i<arg1->PointsCount; i++)
{for (unsigned int j=0; j<arg2->PointsCount; j++)
{unsigned char {*} __restrict__ irow=(arg1->features)+i{*}ncol;
 unsigned char {*} __restrict__ jrow=(arg2->features)+j{*}ncol;
 unsigned long local_dist=0;
#pragma omp for schedule(runtime)
for (unsigned int k=0; k<arg1->FeaturesCount/3; k+=4)
{int v=(int)({*}(irow++)); int u=(int)({*}(jrow++));
 int v1=(int)({*}(irow++)); int u1=(int)({*}(jrow++));
 int v2=(int)({*}(irow++)); int u2=(int)({*}(jrow++));
 int v3=(int)({*}(irow++));
 int u3=(int)({*}(jrow++));
 local_dist+=((u-v){*}(u-v)+(u1-v1){*}(u1-v1))+
 +((u2-v2){*}(u2-v2)+(u3-v3){*}(u3-v3)); }}}
```

We have measured both speedup $(T(1)/T(n))$ and parallelization efficiency $(T(1)/(nT(n))$ where $T(1)$ is the execution time on one thread and $T(n)$ is the execution time on n threads.

The next test was to measure the impact on the computation time the existence of implicit barriers. For all parallelized loops, the data dependency test was performed

Table 7

Parallelized code with j.

```
#pragma omp parallel private (i,k) shared (arg1,arg2,j)
for (unsigned int i=0; i<arg1->PointsCount; i++)
{#pragma omp for shedule(runtime)
for (unsigned int j=0; j<arg2->PointsCount; j++)
{unsigned char {*} __restrict__ irow=(arg1->features)+i{*}ncol;
 unsigned char {*} __restrict__ jrow=(arg2->features)+j{*}ncol;
 unsigned long local_dist=0;
for (unsigned int k=0; k<arg1->FeaturesCount/3; k+=4)
{int v=(int)({*}(irow++)); int u=(int)({*}(jrow++));
 int v1=(int)({*}(irow++)); int u1=(int)({*}(jrow++));
 int v2=(int)({*}(irow++)); int u2=(int)({*}(jrow++));
 int v3=(int)({*}(irow++));
 int u3=(int)({*}(jrow++));
 local\_dist+=((u-v){*}(u-v)+(u1-v1){*}(u1-v1))+
 +((u2-v2){*}(u2-v2)+(u3-v3){*}(u3-v3));}}}
```
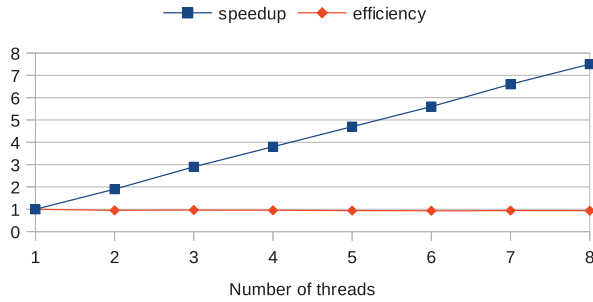

Figure 2. Results for code parallelization.

and barriers were removed in all possible cases. Our code for computing SIFT features and distances was optimized in this way. The results for 8 threads are shown in figure. The speedup of 20% and efficiency of 0.15 are relatively low because in the tested code there are not so many barriers so their overall impact is not very significant.

In our original code, we have used two different arrays for storing SIFT features. In order to improve spatial locality we use a technique called *array merging*. The idea is to sort contiguously in memory all values that are accessed in subsequent loop iteration. This is achieved by defining a struct that contains all values for each point. When the cache retrieves the cache block containing the x valuer for point i, it will automatically prefetch the y value for that point and probably values for the several next points depending on the cache line size. Our code before and after modification is shown in tables and.

For now, our code exhibits some temporal locality. This is achieved by keeping the j index constant on the inner loop which results in each $j - th$ element being
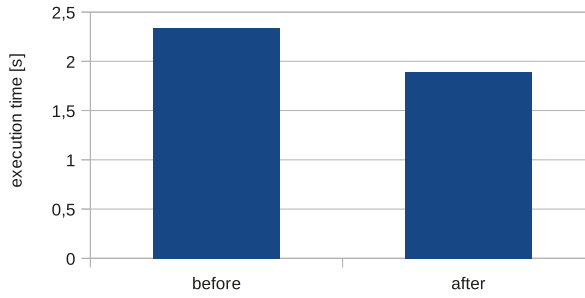
Figure 3. Results for barrier optimization.

Table 8

Original unoptimized code.

```
for (j=0;j<N;j++)
for (i=0;i<N;i++)
{sx+=abs(x[j]-x[i]);
 sy+=abs(y[j]-y[i]);};
```

Table 9

Optimized code with array merging.

```
struct SiftData{int x,y;}
SiftData SIFT[N];
for (j=0;j<N;j++)
for (i=0;i<N;i++)
{sx+=abs(SIFT[j].x-SIFT[i].x);
 sy+=abs(SIFT[j].y-SIFT[i].y);};
```

repeatedly reused. However, during each outer loop iteration, it cycles through all possible i indexes. This results in time between subsequent uses of the same data being long. A solution is to group a small portion of input data together in small blocks that will easily fit in the fastest level of cache.
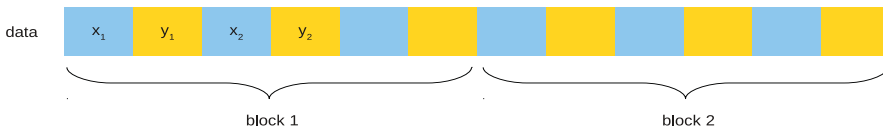

Figure 4. Blocking.

After applying all optimizations described in previous sections, the overall performance test was carried out. Figure shows averaged results achieved on the AMD

Table 10

Optimized code with blocking and array merging.

```
for (k=0;k+2{*}blocksize<N;k+=blocksize)
for (l=a+blocksize;l+blocksize<N;l+=blocksize)
for (i=k;i<k+blocksize;i++)
for (j=l;j<l+blocksize;j++)
{sx+=abs(SIFT{[}j{]}.x-SIFT{[}i{]}.x);
 sy+=abs(SIFT{[}j{]}.y-SIFT{[}i{]}.y);}
```
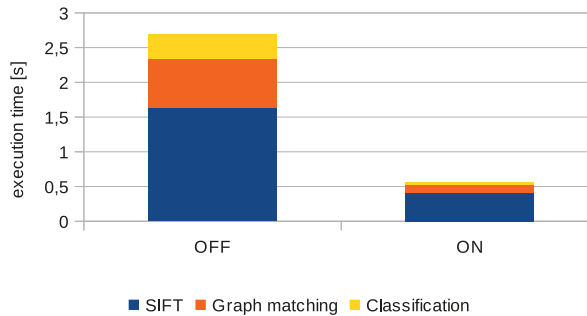
Table 11

Results for different optimizations.

Applied optimizations	Execution time
without blocking and array merging	100%
with array merging only	69%
with blocking and array merging	61%

FX-8120 X8 at 3.1 GHz with 8 MB KB L2 cache. The experiment was carried out using 8 threads. The overall speedup is 4.7 with parallelization efficiency 0.59.

Figure 5. Execution time with and without optimizations.

4. Results analysis

Tables and contain averaged results. The F-measure has been used in order to analyze the performance of the descriptors. The best results in similar image searching were obtained using the Edge Histogram, OpponentSIFT and PCA-SIFT descriptors. OpponentSIFT, C-SIFT and PCA-SIFT tend to outperform the standard SIFT. The reason is that C-SIFT and OpponentSIFT are invariant to some image features (e.g. lightness). In our experiments PCA-SIFT turned out to be slightly more accurate that

the standard SIFT descriptor. Edge Histogram achieves slightly better performance than the SIFT descriptors. The downside of all SIFT variants is that they are much more computationally expensive that the Edge Histogram. Dominant Color is characterized by the poorest performance because this descriptor extracts only dominant colors without their spatial distribution. Although, the Dominant Color yields poor results on its own, it can be used in the planned development as a part of the hybrid method which is supposed to combine information extracted by various descriptors. In the case of finding objects in images, PCA-SIFT significantly outperformed other SIFT descriptors. This confirms that the generalization step (PCA) leads to more accurate features matching. Our results are not compared to those obtained by other authors because we use different methodology in our experiments. For example, we consider two images similar if the belong to the same semantic category while other authors do not use such an assumption.

Applied code optimizations show that rearranging data layout in memory can yield significant performance increase. Especially array merging turns out to be very effective way of optimizing code performance. Employing hyperplane method combined with removal of redundant barriers makes it possible to achieve further code execution acceleration due to utilization of multi-core processors.

5. Conclusions

The paper addresses the issue of finding objects and similar images using different sparse image descriptors. At the beginning the color and edge descriptors used for features extraction as well as the methods of their comparison were briefly described. The goal was to determine the best algorithms as well as to select the ones especially pertinent for hardware acceleration. Finally, the description of the test and results are presented.

The obtained results indicate that Edge Histogram, PCA-SIFT and Opponent-SIFT descriptors are the most effective for selecting the similar images. In comparison to our previous results in [5], Edge Histogram has confirmed its high robustness for similar images finding. OpponentSIFT has also confirmed that its effectiveness is comparable to the one achieved with Edge Histogram.

The further work will be focused on implementing and testing the hybrid descriptor as well as hardware implementation of the SIFT descriptors. The hybrid method will aggregate information from different descriptors (e. g. OpponentSIFT, Dominant Color etc.) and meta-data (especially keywords). Another aspect of the further development will include employing the hardware acceleration using modern graphics cards.

Acknowledgements

References

[1] Alhwarin F.: Improved sift-features matching for object recognition. In *BCS International Academic Conferenc*, pp. 179–190, 2008.

[2] Chee S., Dong K., Soo-Jun P.: Efficient use of mpeg-7 edge histogram descriptor. *ETRI Journal*, 24(1), 2002.

[3] Cyganek B.: Adding parallelism to the hybrid image processing library in multi-threading and multi-core systems. In *IEEE 2nd International Conference on Networked Embedded Systems for Enterprise Applications*, pp. 1–8, 2011.

[4] Duchenne O., Bach F.: A tensor-based algorithm for high-order graph matching. *IEEE Transactions On Pattern Analysis and Machine Intelligence*, 33(12), 2011.

[5] Fraczek R., Cyganek B.: Evaluation of image descriptors for retrieval of similar images. *Intelligent Tools for Building a Scientific Information Platform Studies in Computational Intelligence*, 390:217–226, 2012.

[6] Fraczek R., Grega M., Liebau N., Leszczuk M., Luedtke A., Janowski L., Papir Z.: Ground-truth-less comparison of selected content-based image retrieval measures. *Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering*, 40:101–108, 2010.

[7] Fukunaga K., Koontz W.: Application of the karhunen-loeve expansion to feature selection and ordering. *IEEE Trans. Communications*, 19(4), 1970.

[8] Gersho A., Gray R.: *Vector Quantization and Signal Compression*. Kluwer Academic Publishers, 1992.

[9] Hermes T., Miene A., Herzog O.: Graphical search for images by picture-finder. *Multimedia Tools and Applications, Special Issue on Multimedia Retrieval Algorithmics*, 2005.

[10] Huang C., Yang X.: Performance analysis and improvement of openmp on software distributed shared memory system. In *EWOMP 03*, 2003.

[11] Joliffe I. T.: *Principal Component Analysis*. Springer-Verlag, 1986.

[12] Kandemir M., Choudhary A., *et al.*: A data layout optimization technique based on hyperplanes. *Center for Parallel and Distributed Computing*, 1997.

[13] Lowe D. G.: Distinctive image features from scale-invariant keypoints. *International Journal of Computer Visions*, 60(2):91–110, 2004.

[14] Mademlis A., Daras P., Tzovaras D., Strintzis M. G.: 3d volume watermarking using 3d krawtchouk moments. *International Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications*, 2007.

[15] Manjunath B., Salembier P., Sikora T.: *Introduction to MPEG-7: Multimedia Content Description Interface*. John Wiley and Sons Ltd., 2002.

[16] Martinez J., Koenen R., Pereira F.: Mpeg-7: the generic multimedia content description standard. *IEEE Multimedia*, 9(2):78–87, 2002.

[17] Moravec H. P.: Towards automatic visual obstacle avoidance. In *Proc. of the 5th International Joint Conference on Artificial Intelligence*, p. 584, 1977.

[18] Murase H., Nayar S.: Detection of 3d objects in cluttered scenes using hierarchical eigenspace. *Pattern Recognition Letters*, 18(4), 1997.

[19] Poliwoda M.: Automatically loops parallelized, efficiency of parallelized code. *PAK*, 54(8), 2008.

[20] similar images finder.: smartimagedenoiser.com/download. online.

[21] Yan K., Sukthankar R.: Pca-sift: A more distinctive representation for local image descriptors. In *Proc. of the 2004 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, vol. 2, pp. 506–513, 2004.

[22] Yang N. C., Kuo C. M., Chang W. H., Lee T. H.: A fast method for dominant color descriptor with new similarity measurer. *Journal of Visual Communication and Image Representation*, 19(2):92–105, 2008.

Affiliations

Rafał Frączek

 AGH University of Science and Technology, Academic Computer Center Cyfronet AGH, Krakow, Poland, `rafalfr@agh.edu.pl`

Bogusław Cyganek

 AGH University of Science and Technology, Academic Computer Center Cyfronet AGH, Krakow, Poland, `cyganek@agh.edu.pl`

Kazimierz Wiatr

 AGH University of Science and Technology, Academic Computer Center Cyfronet AGH, Krakow, Poland, `wiatr@agh.edu.pl`