Human Re-identification System on Highly Parallel GPU and CPU Architectures

Sławomir Bąk[1], Krzysztof Kurowski[2], and Krystyna Napierała[3]

[1] INRIA Sophia Antipolis, PULSAR group, France
slawomir.bak@inria.fr
[2] Poznań Supercomputing and Networking Center, Poland
krzysztof.kurowski@man.poznan.pl
[3] Institute of Computing Science, Poznań University of Technology, Poland
krystyna.napierala@cs.put.poznan.pl

Abstract. The paper presents a new approach to the human reidentification problem using covariance features. In many cases a distance operator between signatures based on generalized eigenvalues has to be computed efficiently, especially once the real-time response is expected from the system. This is a challenging problem as many procedures are computationally intensive tasks and must be repeated constantly. To deal with this problem we have successfully designed and tested a new video surveillance system. To obtain the required high efficiency we took the advantage of highly parallel computing architectures such as FPGA, GPU and CPU units to perform calculations. However, we had to propose a new GPU-based implementation of the distance operator for querying the example database. In this paper we present experimental evaluation of the proposed solution in the light of the database response time depending on its size.

Keywords: Re-identification, Covariance Matrix, Generalized Eigenvalues, High Performance Computing, GPU.

1 Introduction

Human re-identification is one of the most challenging and important problems in computer vision and pattern recognition. The re-identification problem can be defined as a determination whether a given person of interest has already been observed over a network of cameras. This issue (also called the *person re-identification problem*) can be considered on different levels depending on the information cues currently available in the system. Biometrics such as *face*, *iris* or *gait* can be used to recognize identities. Nevertheless, in most video surveillance scenarios such detailed information is not available due to a low video resolution or a difficult segmentation (crowded environments, *e.g.* airports, metro stations). Therefore a robust modeling of a global appearance of an individual is necessary to re-identify a given person of interest. In these identification techniques (named *appearance-based approaches*) clothing is the most reliable information about the identity of an individual (there is an assumption that individuals wear

A. Dziech and A. Czyżewski (Eds.): MCSS 2011, CCIS 149, pp. 293–300, 2011.

the same clothes between different sightings). A model of appearance has to handle differences in illumination, pose and camera parameters to allow matching appearances of the same individual observed in different cameras. High accuracy of re-idenfication approaches can only be achieved using an appearance representation based on descriptors which are invariant across different camera views. Recently, a covariance descriptor [9] has proved its effectiveness in recognition [1] and classification approaches [8]. It has been shown that the performance of the covariance features is superior to other methods, as rotation and illumination changes are absorbed by the covariance matrix [1]. Moreover, *integral images* [10] used for fast covariance computation make this descriptor very efficient concerning extraction of the covariance.

However, as covariance matrices do not lay on Euclidean space, it is necessary to apply complex differential geometry to compute a distance between two covariances. The distance operator is computationally heavy as it requires solving *the generalized eigenvalues problem*. As a consequence, matching of the covariance descriptors is slower than matching of other computer vision descriptors, which are usually represented by vectors laying on Euclidean space. This often makes covariance-based approaches difficult to apply in real-time systems in spite of their effectiveness. Hence, we propose a new hybrid architecture based on Graphics Processing Units (GPU) and Field Programmable Gate Arrays (FPGA) accelerators to take advantage of high performance computing and make covariance-based approaches applicable to large-scale databases.

This paper makes two contributions. First, we describe a new GPU- and FPGA-based architecture for the person re-identification problem, which can be easily adjusted to other video surveillance problems, such as object recognition or classification (Section 3). Second, we propose an implementation for finding generalized eigenvalues and eigenvectors for distance operator, using NVIDIA GPU architecture (Section 5). We evaluate our approach in Section 6 before concluding the paper.

2 Motivations and Related Work

There is an increasing demand for effective surveillance systems, *i.e.* systems that can perform low-cost, low-power and high-speed operations. Recently, recognition problems have become one of the most important tasks in video surveillance. As recognition is an extremely difficult task, the existing approaches are computationally heavy. Thus, a new high performance architectures are necessary to apply these approaches to real-time systems. In [7] biologically-inspired algorithms are adjusted to GPU to perform large-scale object recognition. Similarly, in [6] GPU-based neural network is presented to recognize human faces.

We introduce a surveillance system based on FPGA and GPU architectures giving much better computing facilities in comparison with traditional CPU-based systems. The most demanding part of the system – the generalized eigenvalues calculations – is performed using the GPU architecture. There are already a few approaches of finding eigenvalues of symmetric matrices using GPUs [5],

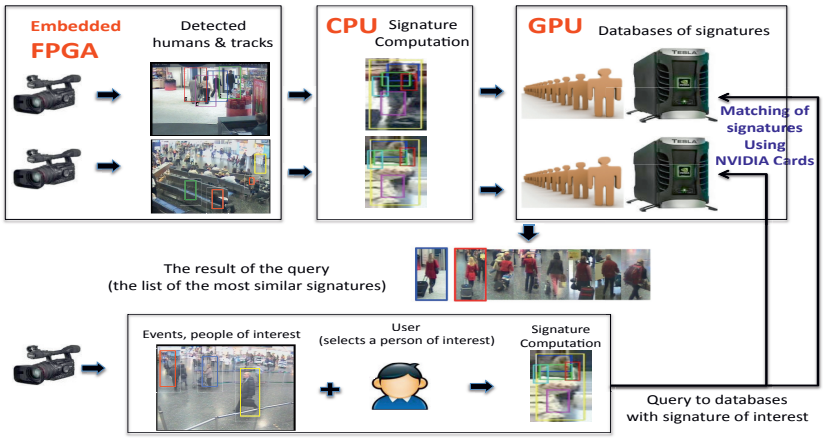


Fig. 1. FPGA- and GPU-based architecture for the person re-identification

but these implementations are focused on computation of the eigenvalues of large matrices (the implementations are optimized for matrices larger than 1024). In contrary to these approaches, we concern a domain-specific problem where it is necessary to solve the generalized eigenvalues problem for a large number of small matrices (the covariance descriptor is mostly represented by square matrix of size between 8 and 16). To the best of our knowledge, we are the first to propose an implementation for finding the generalized eigenvalues and eigenvectors of a large amount of small matrices using NVIDIA GPU cards.

3 System Architecture

Our GPU- and FPGA-based surveillance system for the person re-identification problem (Fig. 1) is designed to assign the tasks to the most suitable architectures. The system consists of a network of cameras with embedded FPGA units, which preprocess a video stream (image denoising, object detection and classification). FPGA units are well suited for video processing tasks which usually expose a high level of parallelism. Therefore, only the necessary information (*blobs* of interest) is sent in the network, without the need of transfering the whole video stream to the central unit. The partially transformed data is collected on a central unit with a CPU and GPU processor. In our approach a *human appearance* is represented by a set of covariance matrices extracted on different resolutions from detected body parts [1]. In total, the human signature is represented by 26 covariance matrices of the size 11. Covariance signature can be computed on CPU efficiently using *integral images*. Signature is then stored on a GPU unit which serves as a database of signatures. The advantage of such a solution is that, first, CPU unit is offloaded from storing this information, and second, when the distance is calculated, the data is already stored on a proper unit. We use a Tesla S1070 with four GPU units, on which there is 4GB of available global memory

for each unit. Taking into account the free space needed for calculations for a query to a database, about 200,000 signatures can be stored in the database on one unit. The user (administrator of a surveillance system) can select an object of interest and query the database with a new signature. The new signature is compared using the distance operator to all signatures in the database. Distance operator is calculated in parallel directly on the GPU unit. The result is a list of the most similar signatures. The most important part of the system is the database stored on GPU and the calculation of distance on this architecture. Preprocessing on FPGA is not that crucial for making the system real-time, therefore we show only how to calculate the distance on GPU (Sec. 5), and we evaluate experimentally the database response time (Sec. 6).

4 Covariance Descriptor

In [9] the covariance of d-features has been proposed to characterize a region of interest. Below we describe only *the geodesic distance* definition proposed in [3], as its computation is the main topic of our work. The distance between two covariance matrices is defined as

$$\rho(C_i, C_j) = \sqrt{\sum_{k=1}^{d} \ln^2 \lambda_k(C_i, C_j)} \tag{1}$$

where $\lambda_k(C_i, C_j)_{k=1\ldots d}$ are the generalized eigenvalues of C_i and C_j, determined by $\lambda_k C_i x_k - C_j x_k = 0,\ k = 1 \ldots d$ and $x_k \neq 0$ are the generalized eigenvectors. Traditional methods work efficiently for dimension d below 5, so often a dimension of covariance matrix has to be decreased to conform the requirements of real-time systems at the expense of accuracy.

5 GPU Implementation

5.1 GPU Architecture

GPU is a high performance architecture in which the emphasis is put on the computing units instead of data caching and control flow units in contrast with CPU. Threads work in a SIMD model and are grouped into blocks. All threads of a block reside on the same processor core and are split into basic scheduling units called warps, consisting of 32 threads. We use the NVIDIA Tesla S1070 with 4 graphic cards, each consisting of 240 processors.

5.2 Generalized Eigenvalues Problem

The calculation of generalized eigenvalues of positive definite symmetric matrices A and B (two corresponding covariance matrices from the compared signatures) is defined by the equation $\mathbf{A}x = \lambda \mathbf{B}x$, where λ denotes a vector of eigenvalues and x is the eigenvector. This equation can be decomposed to the equation $(\mathbf{L}^{-1}\mathbf{A}\mathbf{L}^{-T})\mathbf{L}^T x = \lambda(\mathbf{L}^T x)$, where $\mathbf{L}$ is the upper triangular matrix calculated as $\mathbf{B} = \mathbf{L}\mathbf{L}^T$. The decomposed equation corresponds to original eigenvalues problem. We solve the generalized eigenvalues problem in 4 steps:

1. Calculate the Cholesky Decomposition to solve the equation $\mathbf{B} = \mathbf{L}\mathbf{L}^T$
2. Calculate twice the Forward Substitution to solve $\mathbf{C} = (\mathbf{L}^{-1}\mathbf{A}\mathbf{L}^{-T})\mathbf{L}^T x$
3. Tridiagonalize the symmetric matrix $\mathbf{C}$ to facilitate finding the eigenvalues
4. Find the eigenvalues of $\mathbf{C}$ with the Bisection Algorithm

Let us note that there are many methods to calculate the eigenvalues of a symmetric matrix. On CPU we use a Jacobi algorithm, which does not need to perform the tridiagonalization first, however this algorithm is not easy to parallelize. For the GPU implementation we first tridiagonalize the matrix and then use the bisection algorithm, which can be parallelized much more efficiently.

5.3 Porting the Algorithm to the GPU Architecture

Two sources of parallelism can be used in the algorithm. The first one is obvious – a comparison of two signatures involves comparing n pairs of covariance matrices, which can be naturally processed in parallel. The second one involves the parallelism extracted from each step of the equation performed on a given pair of covariance matrices. We will now briefly describe how we use the parallelism of each procedure to efficiently port it to the GPU architecture.

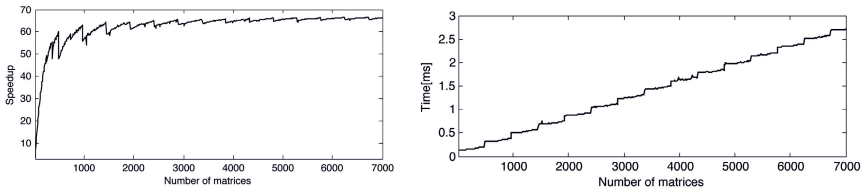
In **Cholesky Decomposition of n matrices**, the formula to calculate each element of the $\mathbf{L}$ upper triangular matrix is given as

$$L_{j,j} = \sqrt{B_{j,j} - \sum_{k=1}^{j-1} L_{j,k}^2}, \quad L_{i,j} = \frac{1}{L_{j,j}}(A_{i,j} - \sum_{k=1}^{j-1} L_{i,k}L_{j,k}) \quad for \;\; i > j \quad (2)$$

All calculations are performed using fast shared memory. We use the Cholesky-Crout algorithm which calculates the matrix column by column, as processing the data in column order ensures the memory coalescence. To calculate an element of a column, only the elements from the columns to the left are needed. So, all the elements in one column can be processed in parallel. For our matrices of size 11, we can use 11 threads to calculate each column in parallel, using 11 iterations to calculate the whole matrix. The natural decomposition of a problem would be to assign one matrix to a block. However, 11 threads is much less than the size of a warp, which is the smallest scheduling unit. Assigning two matrices to a block enables to process two matrices without the increase of processing time, as 22 threads still form only one warp scheduled in one operation.

Forward Substitution solves the equation $\mathbf{L}\mathbf{x} = \mathbf{A}$. Two forward substitutions solve the equation $\mathbf{C} = (\mathbf{L}^{-1}\mathbf{A}\mathbf{L}^{-T})\mathbf{L}^T\mathbf{X}$. We process them together to avoid copying the intermediate data to the global memory. An element $x_{m,k}$ of a matrix $\mathbf{x}$ is calculated as $x_{m,k} = (A_m - \sum_{i=1}^{m-1} L_{m,i}x_{i,k})/L_{m,m}$. Columns of $\mathbf{x}$ are calculated independently. As a result, for one matrix we need 11 threads, and following our observations from Cholesky Distance procedure, we assign 22 threads to one block.

Tridiagonalization is a part of the generalized eigenvalues algorithm which has the lowest level of paralellism. We use the Householder transformation [4] in which $n - 2$ iterations need to be performed sequentially; in each iteration



(a) Speedup - CPU vs GPU (b) Time (ms) of Bisection algorithm

Fig. 2. Speedup for finding generalized eigenvalues and time of Bisection algorithm

the appropriate elements in the *k-th* row and column are zeroed. Some computations such as matrix multiplication and vector-vector multiplications can be parallelised using $n \times n$ threads for each matrix. As 121 threads is more than a size of a warp, we can assign one matrix per block without loosing the efficiency.

Bisection Algorithm is used to calculate the eigenvalues of a symmetric tridiagonal matrix **C** with a given approximation (for details see e.g. [2]). The core function of this algorithm is the `Count()` procedure returning the number of eigenvalues present in a given interval. The algorithm starts with the initial interval constructed using Gerschgorin's theorem. Then, it is divided in two and `Count()` procedure returns a number of eigenvalues in each subset. The unempty nodes are further subdivided into two subsets, until each subset contains only one eigenvalue and the size of a subset is not bigger than the assumed approximation (for our purposes 10e-6 is enough). The `Count()` function can be calculated independently in each node. As only 11 eigenvalues can be found, on each level of the binary tree there will be only up to 11 unempty nodes. We therefore use 11 threads to calculate one matrix and again we assign 2 matrices to a block.

6 Experimental Results

To evaluate the database response time, we calculated the performance for matrices number ranging from 10 to 3000. The database of signatures is stored directly on the GPU (see Section 3), therefore we do not take into account the time of the data transfer, because the reference signatures already reside in the device memory, and the time of transferring the query signature is negligible.

In Fig. 2(a) we present the speedup obtained with comparison to the optimal CPU implementation (Jacobi algorithm from LTI library, see discussion in 5.2). The speedup grows with the number of matrices, and reaches its maximum (66) from about 1500 matrices (corresponding to about 50 signatures). Distributing the database of signatures between 4 GPU cards of Tesla, and performing the calculations for a query signature on all cards in parallel, would result in further speedup improvement. Table 1 presents the time of the component procedures for 5 numbers of matrices. The Cholesky and Forward Substitution times are the lowest. The biggest impact on the total time comes from tridiagonalization procedure, which has the lowest degree of parallelism.

Table 1. Time[ms] of component procedures

N	Cholesky	Forward.S	Tridiag.	Bisect.	Total.GPU	Total.CPU
200	0.037	0.035	0.140	0.147	0.359	16
400	0.049	0.071	0.272	0.187	0.579	32
600	0.082	0.078	0.389	0.325	0.874	48
800	0.093	0.112	0.522	0.352	1.08	64
1000	0.126	0.120	0.657	0.505	1.41	80

In Fig. 2(a) one can notice the *"stairs"* (decrease of speedup) which occur in regular intervals every 480 matrices. This is a result of a similar effect observed in component functions (see e.g. Bisection Algorithm in Fig. 2(b)) and is correlated to the number of processors. The architecture is the most efficiently used when all the processors (240 for Tesla S1070) have some blocks assigned, i.e. when k*480 matrices are processed (each block calculates two matrices). The worst case is for k*480+1 matrices – then, the time is almost equal to processing (k+1)*480 matrices, which results in sudden decrease of speedup in these points.

7 Conclusions

In this paper we demonstrate that it is possible to improve significantly the performance of examplary video surveillance procedures, in particular the distance operator for querying the database. We observe that the speedup grows with the number of matrices. Moreover, we can easily distribute the demanding calculations on many GPU units and obtain very good scalability of the system. Although GPU-based approach of four routines of generalized eigenvalues problem have been already proposed, they are optimized for solving only one large matrix. In our approach we use many very small matrices, which can be efficiently stored in a shared memory on many GPUs. This requires different memory alignments, memory access optimization and simplification of some subprocedures, but as tested experimentally could yield much better performance.

Acknowledgement

The presented work was sponsored by the UCoMS project under award number MNiSW (Polish Ministry of Science and Higher Education) Nr 469 1 N - USA/ 2009 in close collaboration with U.S. research institutions involved in the U.S. Department of Energy (DOE) funded grant under award number DE-FG02-04ER46136 and the Board of Regents, State of Louisiana, under contract no. DOE/LEQSF(2004-07).

References

1. Bak, S., Corvee, E., Bremond, F., Thonnat, M.: Person re-identification using spatial covariance regions of human body parts. In: AVSS (2010)
2. Demmel, J.W., Heath, M.T.: Applied numerical linear algebra. In: Society for Industrial and Applied Mathematics. SIAM, Philadelphia (1997)
3. Förstner, W., Moonen, B.: A metric for covariance matrices. In: Quo vadis geodesia..? TR Dept. of Geodesy and Geoinformatics, Stuttgart University (1999)
4. Householder, A.S.: Unitary triangularization of a nonsymmetric matrix. Journal of the ACM 5 (1958)
5. Lessig, C.: Eigenvalue computation with CUDA. In: NVIDIA techreport (2007)
6. Poli, G., et al.: Processing neocognitron of face recognition on high performance environment based on GPU with CUDA architecture. In: SBAC-PAD, pp. 81–88. IEEE Computer Society, Los Alamitos (2008)
7. Sriram, V.: Design-space exploration of biologically-inspired visual object recognition algorithms using CPUs, GPUs, and FPGAs. In: MRSC (2010)
8. Tosato, D., Farenzena, M., Spera, M., Murino, V., Cristani, M.: Multi-class classification on riemannian manifolds for video surveillance. In: Daniilidis, K., Maragos, P., Paragios, N. (eds.) ECCV 2010. LNCS, vol. 6312, pp. 378–391. Springer, Heidelberg (2010)
9. Tuzel, O., Porikli, F., Meer, P.: Region covariance: A fast descriptor for detection and classification. In: Leonardis, A., Bischof, H., Pinz, A. (eds.) ECCV 2006. LNCS, vol. 3952, pp. 589–600. Springer, Heidelberg (2006)
10. Viola, P., Jones, M.: Rapid object detection using a boosted cascade of simple features. In: CVPR 2001, pp. 511–518 (2001)