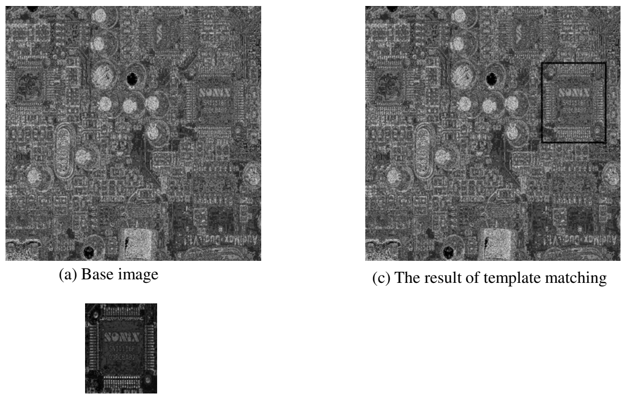
Fast and Accurate Template Matching using Pixel Rearrangement on the GPU

Akihiro Uchida, Yasuaki Ito, and Koji Nakano
Department of Information Engineering,
Hiroshima University
1-4-1 Kagamiyama, Higashihiroshima, Hiroshima, 739–8527 Japan
{*uchida, yasuaki, nakano*}*@cs.hiroshima-u.ac.jp*

***Abstract*—A GPU (Graphics Processing Unit) is a specialized processor for graphics processing. GPUs have the ability to perform high-speed parallel processing using its many processing cores. To utilize the powerful computing ability, GPUs are widely used for general purpose processing. The main contribution of this paper is to show a new template matching algorithm using pixel rearrangement. Template Matching is a technique for finding small parts of an image which match a template image. The feature of our proposed algorithm is that using pixel rearrangement, multiple low-resolution images are generated and template matching for the low-resolution images is performed to reduce the computing time. Also, we implemented our algorithm on a GPU system. The experimental results show that, for an input image with size of 4096 $\times$ 4096 and a template image with size of 256 $\times$ 256, our implementation can achieve a speedup factor of approximately 78 times over the conventional sequential implementation.**

***Keywords*-Template matching; GPU; CUDA; Parallel processing;**



(a) Base image

(b) Template image

(c) The result of template matching

Figure 1. Example of Template Matching

I. Introduction

GPU (Graphics Processing Unit) is a hardware device equipped with many processors that are specialized in the graphics processing. Recent GPUs can be used for not only the graphics processing but also the general purpose processing because GPUs have the high speed computing with their high parallelism. CUDA (Compute Unified Device Architecture) is a developing environment to utilize GPUs for a general purpose processing [1]. Using CUDA, we can develop parallel algorithms to be implemented on GPUs and many studies have been devoted to implement parallel algorithms [2], [3].

Template matching is one of the techniques for detecting a given template from an image that is called a *base image*, and examine whether the template exists in the base image to be detected. It is widely used for industrial manufacturing, robot navigation, geographical research, image registration, etc [4], [5], [6], [7]. Figure 1 shows an example of template matching.

Given a template image and a base image, template matching is to find a position such that a subimage in the base image is the most similar to the template image. There are some measurements of the similarity between a template image and a subimage of the base image. In this paper, we use *the normalized correlation coefficient* as the similarity measure [8]. It is a normalized measurement with the average and standard deviation of a template image and a base image.

To reduce the computing time of template matching, numerous methods have been developed. One of the most famous methods is coarse-to-fine template matching [4], [9], [10]. It locates a low-resolution template image into the low-resolution base image, and then refines the search at higher resolution levels. In this algorithm, it is important to make low-resolution images because the low-resolution template image is not always found in the low-resolution base image. For example, let us consider the case that a low-resolution image is made by sampling every two pixels. When a base image and a template image are checkered patterns as shown in Figure 2, their low-resolution images may be different from each other. Although the base image includes the template image, it occurs that the template matching may be failed. To avoid such faults by sampling, blurred low-resolution images are made by low-pass filter such as Gaussian filter, Laplacian filter, wavelet transform, and so on [11], [12], [13], [14], [15]. However, it is not always possible to find the low-resolution template image. On the other hand, to accelerate the speed of template matching, various methods supported by hardware acceleration with GPUs [16], [14], [17], [18] and FPGAs [19], [20] have been

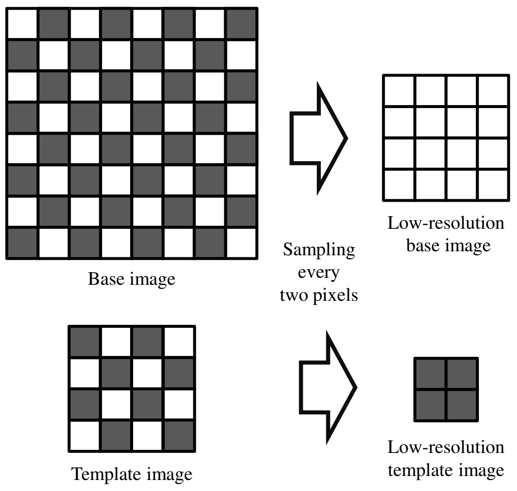

Base image
Sampling every two pixels
Low-resolution base image
Template image
Low-resolution template image

Figure 2. Low-resolution images by sampling

presented.

The main contribution of this paper is to present a new template matching algorithm with pixel rearrangement. This algorithm generates low-resolution images by rearranging pixels of the base image. In the existing sampling-based algorithms, one low-resolution image is generated, and template matching is performed for the image. However, our algorithm generates k^2 low resolution images by sampling every k pixels. The sampling is performed for the base image shifted pixel by pixel from 0 to $k-1$ pixels. Therefore, in our algorithm, given an $n \times n$ base image, k^2 sampled images whose size is $\frac{n}{k} \times \frac{n}{k}$ are generated. More specifically, let $I'_{s,t}$ $(1 \leq s, t \leq k)$ be k^2 sampled images, and they are sampled from a base image such that each sampled image is

$$I'_{s,t}(x, y) = I(kx + s, ky + t) \quad (1 \leq x, y \leq k).$$

Figure 3 shows an example of sampling for $k = 3$. Since the size of a base image is 6×6, k^2 sampled images of size 2×2 are generated. The readers should have no difficulty to confirm that the size of a base image is equal to the total size of the k^2 sampled images. Therefore, we can say that our sampling manner is equivalent to rearrangement of a base image. Also, a template image is reduced by sampling every k pixels. Figure 4 shows an example of base image and its low-resolution images with pixel rearrangement for $k = 64$. In this example, $k^2 = 4096$ low-resolution images are generated. Although the base image and template image are checkered patterns shown in Figure 2, at least one low-resolution image includes the same pattern as the low-resolution template. In other words, if the base image includes the template image, it is always to find it in one of the low-resolution images using our template matching with pixel rearrangement. After that, we perform template matching for the low-resolution template image and each sampled base image. Form the results thus obtained, template matching is performed for the corresponding positions in the original base image.

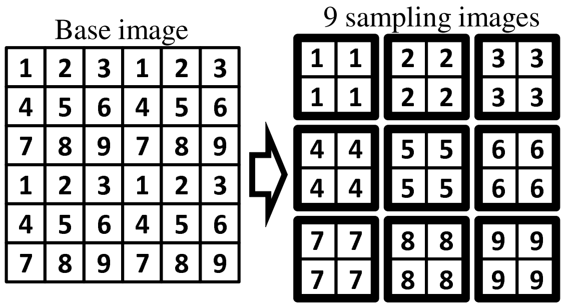


Figure 3. Pixel rearrangement for $k = 3$

Also, we have implemented it in a modern GPU system, Nvidia GeForce GTX 480. Our GPU implementation has achieved approximately 78 times speedup over a conventional CPU implementation,

The remainder of this paper is organized as follows: Section II introduces the similarity between two images. Section III shows the proposed template matching with pixel rearrangement. Section IV briefly describes about CUDA architecture. The GPU implementation is shown in Section V. Section VI exhibits the performance of our proposed algorithm on the GPU. Finally, Section VII offers concluding rearms.

II. The Image Similarity between a Template Image and a Base Image

The main purpose of this section is to define the similarity $R(T, I)$ for a template image T and a base image I to clarify our work in this paper.

There are many measurements of the similarity in template matching. In this paper, we use *the normalized correlation coefficient* as the similarity [8]. The normalized correlation coefficient is used to measure the correlation between two variables. We use it to evaluate the similarity of a template image and a base image in template matching as follows.

First, let us define the similarity of two images A and B of the same size. For simplicity, we assume that they are square, that is, the size of two images is $m \times m$. Let $A(i, j)$ and $B(i, j)$ denote the intensity level of an (i, j) pixel $(1 \leq i, j \leq m)$ of A and B, respectively. The normalized correlation coefficient $R(A, B)$ between the two images A and B is computed by the following formula:

$$R(A, B) = \frac{\sum_{1 \leq i,j \leq m} (A(i,j) - \overline{A})(B(i,j) - \overline{B})}{\sqrt{\sum_{1 \leq i,j \leq m} (A(i,j) - \overline{A}) \sum_{1 \leq i,j \leq m} (B(i,j) - \overline{B})}}$$

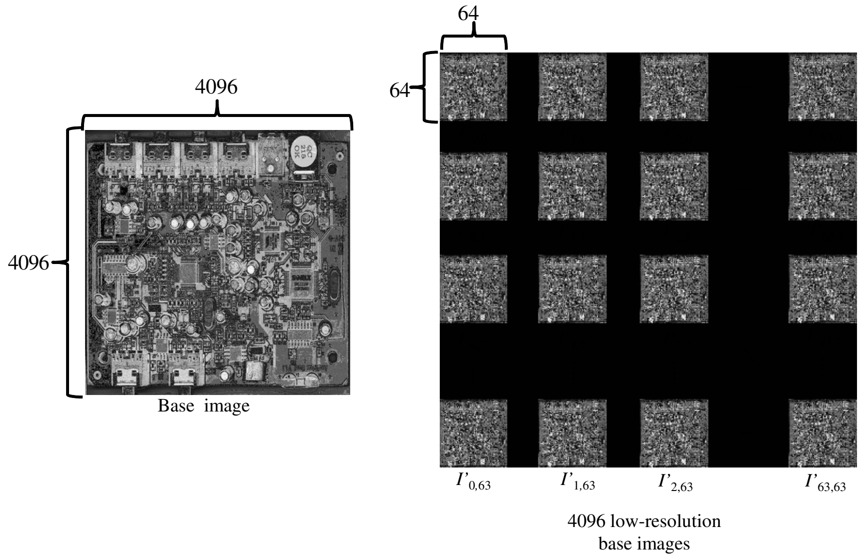


Figure 4. Example of base image and its low-resolution images with pixel rearrangement for $k = 64$

where $\overline{A} = \frac{1}{m^2}\sum A(i,j)$ and $\overline{B} = \frac{1}{m^2}\sum B(i,j)$ are the average pixel values of A and B, respectively. The normalized correlation coefficient $R(A,B)$ takes a real number in the range $[-1,+1]$. Larger value of the normalized correlation coefficient implies that two images A and B are more similar. It should be clear that the normalized correlation coefficient $R(A,B)$ can be computed in $O(m^2)$ time by a sequential algorithm in an obvious way.

Suppose that a base image I and a template image T are given. Let $n \times n$ and $m \times m$ $(n > m)$ be the size of a base image I and a template image T, respectively. Also, let $I(x,y)$ and $T(x,y)$ denote the intensity levels of (x,y) pixels in I and T, respectively. Let $I[x,y]$ $(1 \leq x,y, \leq n-m+1)$ denote an $m \times m$ subimage of I that includes all pixels $I(i',j')$ $(x \leq i' \leq x+m-1$ and $y \leq j' \leq y+m-1)$. We define the similarity $R(T,I)$ between a template T and a base image I as follows:

$$R(T,I) = \max_{1\leq x,y\leq n-m+1} R(T,I[x,y]).$$

Clearly, $R(T,I)$ is larger if I has a more similar subimage to T. Also, the position (x,y) that gives the maximum value of $R(T,I[x,y])$ corresponds to the most similar subimage $I[x,y]$ to the template image T. Let us evaluate the computing time necessary to compute $R(T,I)$ and the most similar position $R(T,I)$ by a sequential algorithm. For an $m \times m$ template image T and a subimage $I[x,y]$, the value of $R(T,I[x,y])$ can be computed in $O(m^2)$ time. Hence, the evaluation of $R(T,I[x,y])$ for all $I[x,y]$ $(1 \leq x,y \leq n-m+1)$ takes $(n-m+1)^2 \times O(m^2) = O(n^2m^2)$ time.

III. Template Matching with Pixel Arrangement

This section describes our proposed template matching algorithm with pixel arrangement. Given an $n\times n$ base image I and an $m \times m$ template image T, in this algorithm, k^2 low-resolution base images $I'_{s,t}$ $(1 \leq s,t \leq k)$ by pixel rearrangement such that

$$I'_{s,t}(x,y) = I(kx+s, ky+t) \quad (1 \leq x,y \leq k),$$

as shown in Figure 5. The size of each $I'_{s,t}$ is $n/k \times n/k$.

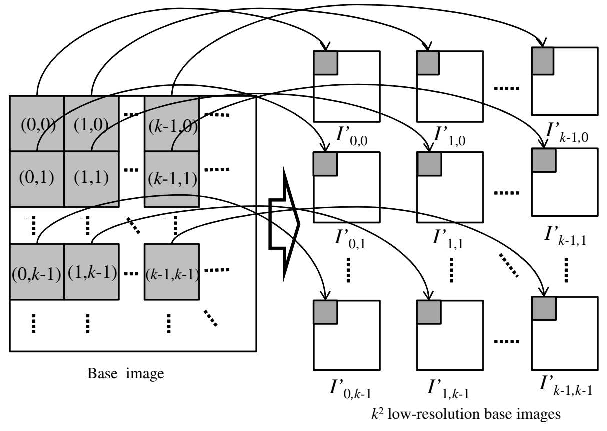


Figure 5. Pixel rearrangement

Also, a low-resolution template image T' is generated by sampling every k pixels. After that, template matching for the low-resolution template image and each low-resolution base image is performed. After that, we perform template matching for the low-resolution template image and each sampled base image. Form the results, template matching is performed for the corresponding positions in the original base image. When an $n \times n$ base image and an $m \times m$ template image are given, our proposed template matching algorithm with pixel rearrangement as follows.

Template Matching Algorithm with Pixel Rearrangement

Step 1. A low-resolution template image T' is generated by sampling every k pixels from a template image. After that, k^2 low-resolution base images $I'_{s,t}$ $(1 \leq s, t \leq k)$ are generated by pixel rearrangement.

Step 2. For each $I'_{s,t}$, the similarity $R(T', I'_{s,t})$ is computed. If the similarity is larger than a threshold value t, its coordinate is made a candidate position for the next step.

Step 3. The candidate positions are transformed to corresponding positions in the original base image. For each position, the similarity between original base image and the template image is computed.

Step 4. The position that has the largest similarity is output as the result.

The details of each step are shown, as follows.

Step 1: In this step, a low-resolution template image T' is generated by sampling every k pixels from an $m \times m$ template image. Since the size of T' is $\frac{m}{k} \times \frac{m}{k}$, it takes $O(\frac{m^2}{k^2})$-time. After that, to obtain k^2 low-resolution base images $I'_{s,t}$ $(1 \leq s, t \leq k)$, pixel rearrangement is performed such that

$$I'_{s,t}(x, y) = I(kx + s, ky + t) \quad (1 \leq x, y \leq k),$$

as shown in Figure 5. The size of each $I'_{s,t}$ is $\frac{n}{k} \times \frac{n}{k}$. Since the above operation is just rearranging pixels in the base image, its computing time is $O(n^2)$.

Step 2: In Step 2, template matching between the low-resolution template image T' and each low-resolution base image $I'_{s,t}$ is performed, that is computing similarities $R(T', I'_{s,t})$ $(1 \leq s, t \leq k)$, In the template matching, if the similarity is larger than a certain threshold t, its coordinate is stored as a candidate position for the next step. Since the sizes of each low-resolution image and the low-resolution template image are $\frac{n}{k} \times \frac{n}{k}$ and $\frac{m}{k} \times \frac{m}{k}$, $O(\frac{n^2m^2}{k^4})$-time is necessary to perform each template matching. Therefore, this step takes $O(\frac{n^2m^2}{k^2})$ in total.

Step 3: In this step, the candidate positions are transformed to corresponding positions in the original base image. We assume that the number of the candidate positions is l found in Step 2 and let $p_i = (x_i, y_i)$ $(1 \leq i \leq l)$ be the transformed candidate positions. For each p_i, template matching is performed, that is computing similarities $R(T, I[x_i, y_i])$. It takes $O(m^2)$-time to perform the template matching for each p_i. Therefore, the total computing time in this step is $O(lm^2)$.

Step 4: In Step 4, the maximum similarity position in Step 3 is output as the result. To find the maximum position from l candidates, it takes $O(l)$-time.

According to the above, the total running time is $O(\frac{n^2m^2}{k^2} + lm^2)$. If l is small, it is close to $O(\frac{n^2m^2}{k^2})$.

IV. Compute Unified Device Architecture (CUDA)

Graphics Processing Units (GPUs) can achieve a high computational throughput due to their large number of processing cores and different memory spaces. All the processing cores are organized into several streaming multi-core processors as shown in Figure 6. For fully utilizing all the processing cores of a GPU, numerous threads are required. Compute Unified Device Architecture (CUDA) [1] organizes these threads into a large *grid* of *thread blocks*. Each thread block contains a number of *threads* which can be executed on an assigned streaming multi-core processor. Threads of a thread block are organized into several *warps* and each warp contains 32 threads. At a time, only a half warp of a thread block can be executed by the assigned streaming multi-core processor concurrently. The grid will launch a segment of codes, named a *kernel*, to occupy a GPU device at a time. Actually, CUDA is a new parallel programming model and instruction set architecture. CUDA comes with a software environment that allows developers to use C-like high-level programming language.

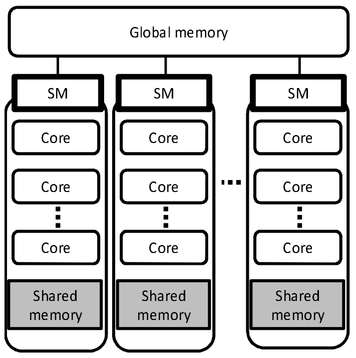


Figure 6. CUDA Hardware Architecture

On the other hand, GPUs can provide different memory spaces for different applications and each memory space has its own advantages and drawbacks. In CUDA architecture, each memory space has a corresponding specification. In this paper we only introduce few of them shown as follows.

Global memory is a main device memory of GPUs and which is off-chip memory. Therefore it has heavy access latency to each processing core. Fortunately, CUDA provides a technique known as *coalescing* [21] to hide the access latency of the global memory. When 16 sequential threads access 16 sequential and aligned values in the global memory, the GPU will automatically combine them into a single transaction.

Shared memory is a sort of on-chip memory and which is located within each streaming multi-core processor. It has almost no access latency and only visible to the thread block which is executed by the corresponding streaming multi-core processor. In practice, the shared memory can be used as a cache to hide the access latency of the global memory.

V. GPU Implementation

In this section, we show an implementation of parallel execution of template matching with pixel rearrangement using a GPU. The idea of our GPU implementation is to reduce the number of access to the global memory using the shared memory. In Step 1 and Step 2 shown in Section III, since the execution for each low-resolution image can be performed in parallel the execution for each low-resolution image can be performed in parallel. Therefore, one thread block is assigned to the execution for each low-resolution image and every thread block runs in parallel. Moreover, in each thread block, pixel rearrangement in Step 1 and template matching for low-resolution in Step 2 are executed with plural threads in parallel.

Suppose that an $n \times n$ base image and an $m \times m$ template image are given and they are stored to the global memory. First, a grid which consists of k^2 thread blocks is performed. Each thread block is composed of 512 threads and performs the process for one of the k^2 low-resolution base images $I'_{s,t}$ $(1 \leq s, t \leq k)$ in Step 1 and Step 2. Each thread block reads corresponding pixels of $I'_{s,t}$ from the base image in the global memory and store them to the shared memory. Similarly, it reads the low-resolution template image T' by sampling every k pixels and store them to the shared memory. Then, using $I'_{s,t}$ and T' in the shared memory, the similarity $R(T', I'_{s,t})$ is computed in parallel using 512 threads. Each thread computes the similarity and the result is stored to the shared memory. In the computation of the similarity, we utilize *sequential similarity detection algorithm* (SSDA) [22] to reduce the computing time. SSDA accelerates the computation of the normalized correlation coefficient by reducing unnecessary computation for a certain threshold value. In our implementation, the similarity is computed such that it is no less than 0.95. The positions whose similarity is no less than 0.95 are stored to the global memory as candidate positions. The process of the above can be done using the shared memory except reading the input images and storing the results to reduce the number of access to the global memory.

After computation of the similarity for each $I'_{s,t}$, the similarity for the original resolution of the base image and the template image is computed, which corresponds to Step 3. If the number of candidate positions found in the previous step is l, the similarity for each candidate position is computed by l thread blocks that is composed of 512 threads. Each thread block transforms the position in the low-resolution base image to that in the original resolution base image. Then, it reads a corresponding $m \times m$ subimage in the transformed position and the original resolution template image. Since the memory reading mostly accesses sequential pixels in the global memory, almost all accesses are benefited by coalescing access. In this computation of the similarity, SSDA is utilized to reduce the computing time. After computing the similarity, it is stored in the global memory. Then, the maximum similarity position of l candidate positions is output using one thread block that is composed of one thread, which corresponds to Step 4.

VI. Performance Evaluation

We have implemented and evaluated our proposed template matching with pixel rearrangement on the GPU. For the purpose of comparison, we have also implemented a conventional software approach without support of the GPU. We have used Nvidia GeForce GTX 480 with 480 processing cores (15 streaming multi-core processors which has 32 processing cores) running in 1.4GHz and 3GB memory. To evaluate software approach, we have used Intel Core i7 870 running in 2.93GHz and 8GB memory.

To evaluate our proposed algorithm, we have compared the execution time of template matching with and without pixel rearrangement in software implementation. Table I shows the performance. We have used a base image shown in Figure 4 and two template images shown in Figure 7 that are included in the base image. Also, in the template matching with pixel rearrangement, we have used the parameter $k = 64$, that is we have generated 64^2 low-resolution base images by sampling every 64 pixels. Since the sizes of the base image and the template images are 4096×4096 and 256×256, the sizes of one of the low-resolution base images and the low-resolution templates image are 64×64 and 4×4, respectively. In the implementation with pixel rearrangement, the number of candidate positions found in Step 2 for template images (a) and (b) was 76 and 1873, respectively. The computing time depends on the number of candidate positions. Therefore, the computing time is different between them. The results of both are equivalent and using pixel rearrangement, our proposed algorithm achieved approximately 2000 times speedup. Since the speedup factor is no more than $k^2 = 64^2 = 4096$ as shown in Section III, the result is reasonable.

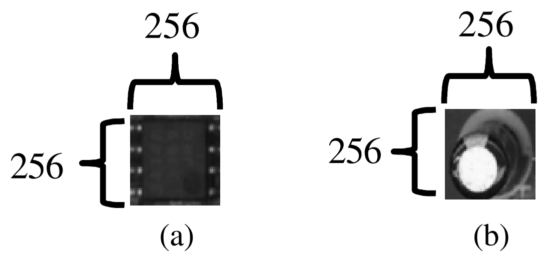


Figure 7. Template Images

Table II
PERFORMANCE OF TEMPLATE MATCHING WITH PIXEL REARRANGEMENT ON THE GPU

Template image (a)

	Step1 and Step2 [ms]	Step3 [ms]	Step4 [ms]	Total [ms]	Speed-up
CPU	3900.063	9.092	0.001	3909.156	—
GPU	33.692	1.261	0.007	34.954	111.837

Template image (b)

	Step1 and Step2 [ms]	Step3 [ms]	Step4 [ms]	Total [ms]	Speed-up
CPU	4095.032	506.867	0.008	4601.906	—
GPU	33.356	24.956	0.008	58.320	78.908

Table I
PERFORMANCE OF TEMPLATE MATCHING WITH AND WITHOUT PIXEL REARRANGEMENT IN THE SOFTWARE IMPLEMENTATION

Template image (a)

	Without pixel rearrangement	With pixel rearrangement
Step 1 [ms]	—	421.504
Step 2 [ms]	—	3478.559
Step 3 [ms]	9287119	9.092
Step 4 [ms]	58.183	0.001
Total [ms]	9287177.183	3909.156
Speed-up	—	2375.750

Template image (b)

	Without pixel rearrangement	With pixel rearrangement
Step 1 [ms]	—	422.634
Step 2 [ms]	—	3672.240
Step 3 [ms]	9231575	506.867
Step 4 [ms]	55.910	0.008
Total [ms]	9231630.910	4601.749
Speed-up	—	2006.114

Table II shows performance of our proposed template matching on the GPU and the performance of the software implementation. We have also used a base image shown in Figure 4 and two template images shown in Figure 7. Note that Step 1 and Step 2 in the GPU implementation are executed by the identical thread block. Therefore, the computing time of the two steps is shown by the total time of them in the table. The results of the GPU are equivalent to that of the CPU implementation and GPU implementation achieved at least approximately 78 times speedup.

VII. CONCLUSIONS

In this paper, we have presented a template matching algorithm with pixel rearrangement. Using pixel rearrangement, multiple low-resolution images are generated and template matching for the low-resolution images is performed to reduce the computing time. Also, we have implemented it with a modern GPU system, Nvidia GeForce GTX 480. Comparing to the performance of the CPU implementation, the GPU implementation have achieved approximately 78 times speedup.

REFERENCES

[1] NVIDIA Corporation., *NVIDIA, CUDA Architecture*, http://www.nvidia.com/object/cuda_home_new.html.

[2] D. Man, K. Uda, H. Ueyama, Y. Ito, and K. Nakano, "Implementations of parallel computation of Euclidean distance map in multicore processors and GPUs," in *Proceedings of International Conference on Networking and Computing*, November 2010, pp. 120–127.

[3] K. Ogawa, Y. Ito, and K. Nakano, "Efficient Canny edge detection using a GPU," in *Proceedings of International Workshop on Advances in Networking and Computing*, 2010, pp. 279–280.

[4] S. Yoshimura and T. Kanade, "Fast template matching based on the normalized correlation by using multiresolution eigenimages," in *Proceedings of International Conference on Intelligent Robots and Systems*, 1994, pp. 2086–2093.

[5] Y. Abe, M. Shikano, T. Fukuda, F. Arai, and Y. Tanaka, "Vision based navigation system by variable template matching for autonomous mobile robot," in *Proceedings of IEEE International Conference on Robotics and Automation*, vol. 2, 1998, pp. 952–957.

[6] D. Rainsford and W. Mackaness, "Template matching in support of generalisation of rural buildings," in *Proceedings of International Symposium on Spatial Data Handling*, 2002, pp. 137–152.

[7] B. Zitov'a and J. Flusser, "Image registration methods: a survey," *Image and vision computing*, vol. 21, no. 11, pp. 977–1000, 2003.

[8] J. Rodgers and W. Nicewander, "Thirteen ways to look at the correlation coefficient," *The American Statistician*, vol. 42, no. 1, pp. 59–66, 1988.

[9] A. Rosenfeld and G. Vanderbrug, "Coarse-fine template matching," *IEEE Transactions on Systems, Man and Cybernetics*, vol. 7, no. 2, pp. 104–107, 1977.

[10] S. Tanimoto, "Template matching in pyramids," *Computer Graphics and Image Processing*, vol. 16, no. 4, pp. 356–369, 1981.

[11] E. H. Adelson and P. J. Burt, "Image data compression with the laplacian pyramid," in *Proceedings of the Conference on Pattern Recognition and Image Processing*, 1981, pp. 218–223.

[12] G. Bonmassar and E. L. Schwartz, “Improved cross-correlation for template matching on the laplacian pyramid,” *Pattern recognition letters*, vol. 19, no. 8, pp. 765–770, 1998.

[13] A. C. Berg and J. Malik, “Geometric blur for template matching,” in *Proceedings of IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2001, pp. 607–614.

[14] S. Ludwig, “Implementation of a spatio-temporal laplacian image pyramid on the GPU,” Ph.D. dissertation, Universität zu Lübeck, Feburary 2008.

[15] H. Cho and T. Park, “Wavelet transform based image template matching for automatic component inspection,” *The International Journal of Advanced Manufacturing Technology*, vol. 50, no. 9–12, pp. 1033–1039, 2010.

[16] R. Cabido, A. S. Montemayor, and Á. Sánchez, “Hardware-accelerated template matching,” in *Proceedings of Iberian Conference on Pattern Recognition and Image Analysis*, 2005, pp. 691–698.

[17] R. F. Anderson, J. S. Kirtzic, and O. Daescu, “Applying parallel design techniques to template matching with GPUs,” in *Proceedings of IEEE VECPAR 2010*, 2010.

[18] N. A. Vandal and M. Savvides, “CUDA accelerated iris template matching on graphics processing units (GPUs),” in *Proceedings of Fourth IEEE International Conference on Biometrics: Theory Applications and Systems*, 2010, pp. 1–7.

[19] K. Nakano and E. Takamichi, “An image retrieval system using FPGAs,” *IEICE Transactions on Information and Systems*, vol. E86-D, no. 5, pp. 811–818, May 2003.

[20] Y. Ren, J. Zhu, X. Yang, and S. Ye, “The application of virtex-ii pro fpga in high-speed image processing technology of robot vision sensor,” *Journal of Physics: Conference Series*, vol. 48, pp. 373–378, 2006.

[21] NVIDIA Corporation., *NVIDIA CUDA C Programming Guide*, http://developer.download.nvidia.com/compute/cuda/3_1/toolkit/docs/NVIDIA_CUDA_C_ProgrammingGuide_3.1.pdf.

[22] D. I. Barnea and H. F. Silverman, “A class of algorithms for fast digital image registration,” *IEEE Transactions on Computers*, vol. 21, no. 2, pp. 179–186, 1972.