Accelerating Phase Correlation functions using GPU and FPGA: A comparison study

Kentaro Matsuo, Tsuyoshi Hamada, Masayuki Miyoshi, Yuichiro Shibata, and Kiyoshi Oguri
Graduate School of Science and Technology
Nagasaki University
Bunkyo 1–14, Nagasaki-shi, Nagasaki, 852–8521 Japan
Email: aniki@pca.cis.nagasaki-u.ac.jp

***Abstract*—In this paper, we present a comparison study about implementatons of phase correlation function using GPUs, ASIC and FPGAs. The Phase Only Correlation(POC) method demonstrates high robustness and subpixel accuracy in the pattern matching and the image registration. However, there is a disadvantage in computational speed because of the calculation of 2D-FFT etc. We have proposed a novel approach to accelerate POC method using GPU to solve the calculation cost problem. Using our GPU-based POC implementation, each POC calculation can be done within 2.36 milli seconds using a GPU for 256 $\times$ 256 pixels, on the other hand, within 27.15 milli seconds for Cinderella II 100 MHz (ASIC), 4.51 milli seconds for Xilinx XC2V6000 66 MHz (FPGA). These results show that, for POC calculation and FFT-based computations in general, GPUs are very competitive in terms of performance and performance figures, whereas FPGAs are competitive in terms of performance per frequency figures.**

I. Introduction

Image matching is one of the central tasks of image processing, which detects image similarity, scaling, translation and rotation between some target image and the reference image. This has been widely used in machine vision industry for the purposes of registration such as fingerprint matching or position adjustment in machines.

There are a lot of methods of the image matching. A pixel base direct matching between two images called template matching is simple and efficient for the small patch of images. However, this approach is not robust for illumination or 3D projection changes, and does not operate correctly when the scale or angle of images is changed. Furthermore sub-pixel matching is impossible in template matching. A Fourier transform base matching is also direct matching between both Fourier transforms of original images. A correlation function base matching has sub-pixel accuracy, therefore this method is used for position adjustment in machines. Since the cross-correlation function between A and B is inverse Fourier transform of the product of Fourier transform of A and complex conjugate of Fourier transform of B, the performance of FFT is the key in this method.

Recently, a high-accuracy image matching method using a Phase-Only correlation (POC) function has been developed [1], [2], [6], [7]. This POC base matching is more robust in illumination changes than simple correlation function base matching. Using the POC function, we can estimate the translation displacement as well as the degree of similarity between two image blocks from the location and height of the correlation peak, respectively. It has been demonstrated that this matching technique can estimate the displacement between two images with 1/100-pixel accuracy when the image size is about 100x100 pixels.

POC base image matching requires more computational power than simple correlation function base matching, therefore it was not used in real applications at first. Progress of the semiconductor technologies brings this method into reality. POC base matching was implemented in a application specific integrated circuits (ASIC). In this implementation, the POC operation of 256x256 pixels is done in 27.15ms [3]. Fingerprint matching using POC was implemented in FPGA [4]. Object search in machine vision using POC was also implemented in FPGA [5].

In this paper, we present a new approach to accelerate POC method using GPU (Graphics Processing Unit). GPU is a dedicated hardware for rendering 3D object. Since new type of GPU has more than 100 processing elements, not only rendering process but also general purpose applications are accelerated. Nvidia has developed a design language and its environment called CUDA (Computer Unified Device Architecture) in Nov. 2006. We have implemented POC base image matching which detects translation and rotation between 2 images in CUDA.

II. Algorithm of POC and RIPOC

We have implemented POC for displacement and Rotation Invariant Phase Only Correlation (RIPOC) for rotation on GPU. We summarize these algorithm.

A. POC

Fourier transform's phase component has many information about object shape. POC uses this property. Figure 1 demonstrates the outline of POC. Following is the detail of POC algorithm.

1) definition: Now consider $\tilde{f}(x_1, x_2)$ as a 2D image defined in continuous space with real-number indices x_1 and x_2. And δ_1 and δ_2 represent sub-pixel displacement of $\tilde{f}(x_1, x_2)$. The displaced image can be indicated as $\tilde{f}(x_1 - \delta_1, x_2 - \delta_2)$. $f(x_1, x_2)$ and $g(x_1, x_2)$ are spatially sampled images of $\tilde{f}(x_1, x_2)$ and $\tilde{f}(x_1 - \delta_1, x_2 - \delta_2)$, and are given by

978-0-7695-3714-6/09 $25.00 © 2009 IEEE
DOI 10.1109/AHS.2009.53

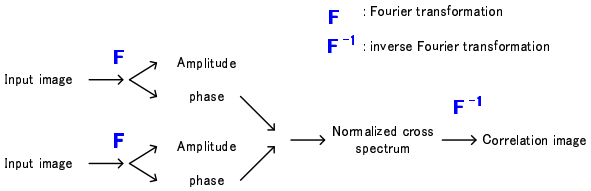

Fig. 1. Algorithm of POC

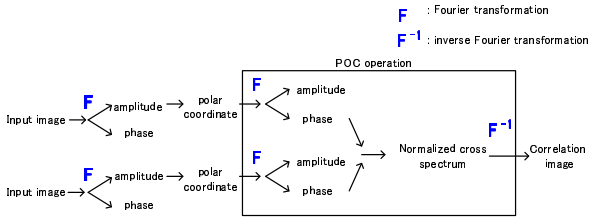

Fig. 2. Algorithm of RIPOC

$$f(n_1,n_2) = \tilde{f}(x_1,x_2)\mid_{x_1=n_1T_1,x_2=n_2T_2}, \quad (1)$$
$$g(n_1,n_2) = \tilde{f}(x_1-\delta_1,x_2-\delta_1)\mid_{x_1=n_1T_1,x_2=n_2T_2}, \quad (2)$$

where T_1 and T_2 are the sampling intervals, and index ranges are given by $n_1 = -M_1,\cdots,M_1$ and $n_2 = -M_2,\cdots,M_2$, and $N_1 = 2M_1+1$ and $N_2 = 2M_2+1$ are image size.

2) Fourier transformation: $F(k_1,k_2)$ and $G(k_1,k_2)$ are 2D Discrete Fourier Transforms (2D-DFTs) of $f(n_1,n_2)$ and $g(n_1,n_2)$ which are spatially sampled images. $F(k_1,k_2)$ and $G(k_1,k_2)$ are defined as

$$F(k_1,k_2) = \sum_{n_1,n_2} f(n_1,n_2)W_{N_1}^{k_1n_1}W_{N_2}^{k_2n_2} = A_F(k_1,k_2)e^{j\theta_F(k_1,k_2)}, \quad (3)$$
$$G(k_1,k_2) = \sum_{n_1,n_2} g(n_1,n_2)W_{N_1}^{k_1n_1}W_{N_2}^{k_2n_2} = A_G(k_1,k_2)e^{j\theta_G(k_1,k_2)}, \quad (4)$$

where k_1 and k_2 are index of frequency, and are given by $k_1 = -M_1,\cdots,M_1$ and $k_2 = -M_2,\cdots,M_2$, and $W_{N_1}^{k_1n_1}$ and $W_{N_2}^{k_2n_2}$ are twiddle factor, and are defined as $W_{N_1}^{k_1n_1} = e^{-j\frac{2\pi}{N_1}k_1n_1}$ and $W_{N_2}^{k_2n_2} = e^{-j\frac{2\pi}{N_2}k_2n_2}$. The operator $\sum_{n_1,n_2}$ means $\sum_{n_1=-M1}^{M_1}\sum_{n_2=-M2}^{M_2}$. $A_F(k_1,k_2)$ and $A_G(k_1,k_2)$ are amplitude components, and $e^{j\theta_F(k_1,k_2)}$ and $e^{j\theta_G(k_1,k_2)}$ are phase components.

3) normalized cross spectrum: The normalized cross spectrum $R(k_1,k_2)$ is given by

$$R(k_1,k_2) = \frac{F(k_1,k_2)\overline{G(k_1,k_2)}}{|F(k_1,k_2)\overline{G(k_1,k_2)}|} = e^{j\{\theta_F(k_1,k_2)-\theta_G(k_1,k_2)\}}, \quad (5)$$

where $\overline{G(k_1,k_2)}$ mean the complex conjugate of $G(k_1,k_2)$. $e^{j\{\theta_F(k_1,k_2)-\theta_G(k_1,k_2)\}}$ are differential between phase components of two images.

4) Phase Only Correlation function: The Phase Only Correlation function $r(n_1,n_2)$ is defined as

$$r(n_1,n_2) = \frac{1}{N_1N_2}\sum_{k_1,k_2} R(k_1,k_2)W_{N_1}^{-k_1n_1}W_{N_2}^{-k_2n_2}. \quad (6)$$

$r(n_1,n_2)$ is the 2D Inverse Discrete Fourier Transform (2D IDFT) of $R(k_1,k_2)$. Where $\sum_{k_1,k_2}$ denotes $\sum_{k_1=-M1}^{M_1}\sum_{k_2=-M2}^{M_2}$.

5) peak estimate: The POC function $\dot{r}(n_1,n_2)$ between $f(n_1,n_2)$ and $g(n_1,n_2)$ also be given by

$$\dot{r}(n_1,n_2) \simeq \frac{\alpha}{N_1N_2}\frac{\sin\{\pi(n_1+\delta_1)\}}{\sin\{\frac{\pi}{N_1}(n_1+\delta_1)\}}\frac{\sin\{\pi(n_2+\delta_2)\}}{\sin\{\frac{\pi}{N_2}(n_2+\delta_2)\}}. \quad (7)$$

This notation is valid for exactly same image. It is possible to find the location of the peak that may exist between image pixels, because $\dot{r}(n_1,n_2)$ is fitted to the $r(n_1,n_2)$ around the correlation peak. Where $\alpha < 1$, and δ_1 and δ_2 are fitting parameters. The peak of $r(n_1,n_2)$ shows degree of similarity between two images. The location of the peak shows displacement between two images.

B. RIPOC

Since phase component is associated with displacement, amplitude component is associated with rotation. RIPOC uses this property. Figure 2 demonstrates the outline of RIPOC. Following is the detail of RIPOC algorithm.

1) definition: Consider $\tilde{f}(x_1,x_2)$ as a 2D image defined in continuous space with real number indices x_1 and x_2. $\tilde{g}(x_1,x_2)$ is what rotated $\tilde{f}(x_1,x_2)$ is. θ represent this rotation angle. $f(x_1,x_2)$ and $g(x_1,x_2)$ are spatially sampled images of $\tilde{f}(x_1,x_2)$ and $\tilde{g}(x_1,x_2)$, and are given by

$$f(n_1,n_2) = \tilde{f}(x_1,x_2)\mid_{x_1=n_1T_1,x_2=n_2T_2},$$
$$g(n_1,n_2) = \tilde{g}(x_1,x_2)\mid_{x_1=n_1T_1,x_2=n_2T_2}. \quad (8)$$

2) Fourier transformation: $F(k_1,k_2)$ and $G(k_1,k_2)$ are 2D-DFTs of $f(n_1,n_2)$ and $g(n_1,n_2)$ that are spatially sampled images.

3) logarithm: $\mid F(k_1,k_2)\mid$ and $\mid G(k_1,k_2)\mid$ are given by calculating logarithm of $F(k_1,k_2)$ and $G(k_1,k_2)$.

4) polar coordinate: $F_p(m_1,m_2)$ and $G_p(m_1,m_2)$ are images that are converted to polar coordinate from $\mid F(k_1,k_2)\mid$ and $\mid G(k_1,k_2)\mid$. $F_p(m_1,m_2)$ and $G_p(m_1,m_2)$ are defined as

$$F_p(m_1,m_2) = \mid F(r_{m2}\cos\phi_{m1}, r_{m2}\sin\phi_{m1})\mid,$$
$$G_p(m_1,m_2) = \mid G(r_{m2}\cos\phi_{m1}, r_{m2}\sin\phi_{m1})\mid, \quad (9)$$

Fig. 4. A high-speed POC/RIPOC calculation system by GPU

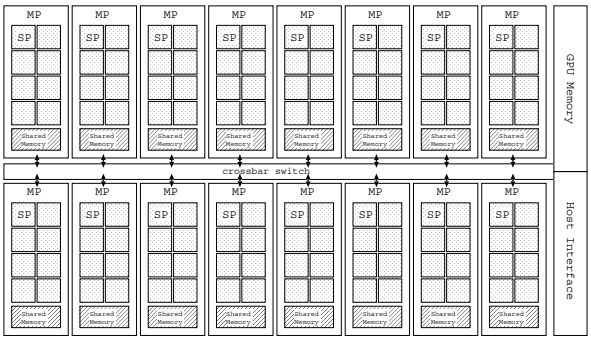

Fig. 5. Architecture of GeForce 8800 GTS

where m_1 and m_2 are the coordinates of images that were converted to the polar coordinate, and that are given by $m_1 = -M, \cdots, M$ and $m = -M, \cdots, M$. In addition, ϕ_{m_1} that is angle and r_{m_2} that is radius are defined as

$$\phi_{m_1} = \frac{\pi}{N} m_1, \quad r_{m_2} = m_2 + M. \tag{10}$$

In addition, the converter use bilinear interpolation.

5) POC operation: We can get the rotation by applying POC operation to $F_p(m_1, m_2)$ and $G_p(m_1, m_2)$.

III. IMPLEMENTATION

This section describes the GPU implementation of POC and RIPOC. In this implementation, we use the CUFFT function of CUDA library for 2D FFT. At first, elapsed time of the other part is ten times longer than the elapsed time of CUFFT. After tuning the memory layout and the order of each processes, elapsed time of CUFFT is about half of total processing time. The following subsections describe the implementation details. Figure 3 indicates the ability of our implementation. Each two pictures from many pictures which we took of our GPU cluster are examined using POC and RIPOC to detect similarity, displacement, and rotation. Then using this information, all pictures are rearranged into one picture successfully.

A. Implementation environment

We use a Nvidia's GeForce 8800 GTS GPU. Figure 5 shows GeForce 8800 GTS architecture. In the figure, SP indicates Stream Processor. This is a 32-bits scalar processor. 8 SPs compose a SIMD type multi processors (MP). GPU chip consists of 16 MPs. Each MP has a 16KB shared memory which is shared by 8 SPs. Outside of the chip, there is a 512MB on board memory, which is called GPU memory. In CUDA, CPU is called Host, and GPU is called Device. A CUDA program consists of a host program and GPU kernel functions. The host program which runs on CPU transfers the data between CPU memory and GPU memory and invokes GPU kernel functions. While CUDA language is based on C language, CUDA inhibits recursive operations nor some pointer operations in kernel functions. A typical flow of a CUDA program is as below (figure 6).

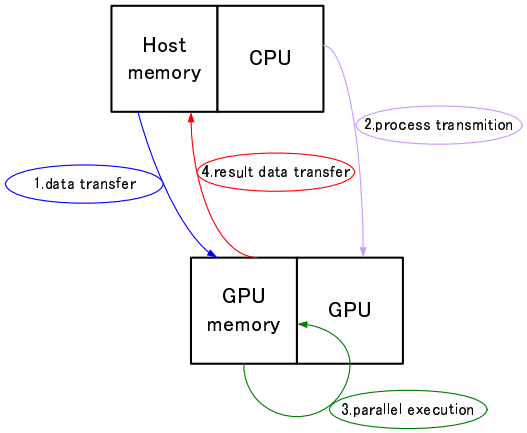

Fig. 6. Typical flow of processing in the CUDA

1) CPU transfers the date from CPU memory to GPU memory.
2) CPU invokes GPU.
3) Cores of GPU operate in parallel.
4) CPU transfers the data from GPU memory to CPU memory.

B. Implementation of POC

Each step of the POC algorithm mentioned in II-A is implemented as kernel functions.

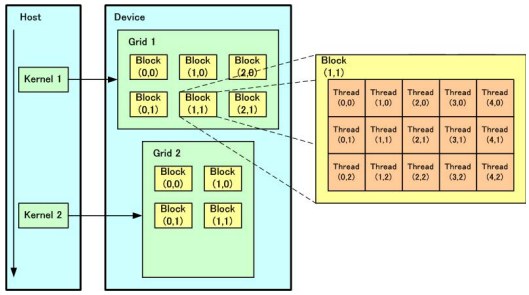

Fig. 7. Programing model of the CUDA

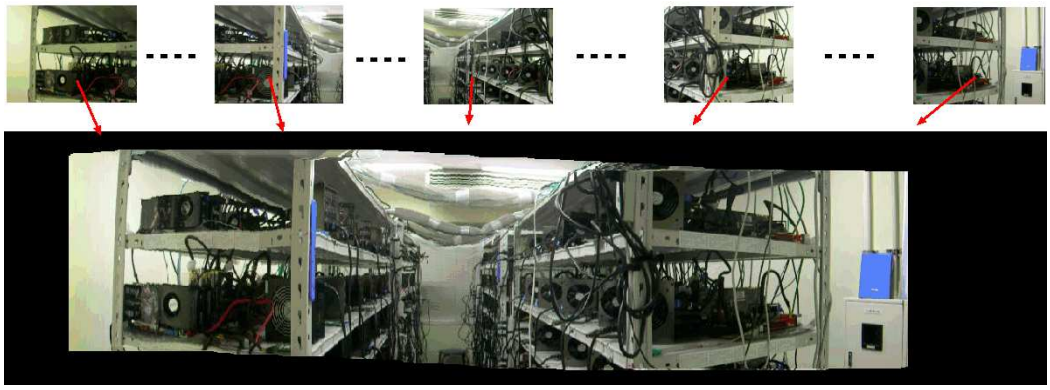

Fig. 3. Picture composition example

1) Fourier transfer: We uses CUFFT library of the CUDA for 2D DFT. As the range of $n_1 C n_2$ is defined as $-M, \cdots, M$ in expression (3), in order to use CUFFT it is necessary to rearrange those into $0, \cdots, 2M$ and to shift the data.

2) Thread and memory allocation: In CUDA architecture, a large number of threads require a small number of physical processing units, or SPs. By these mechanism almost all physical processing unit operate simultaneously. A block is a component where those threads which share the same shared memory are described. And every inter-memory transfers are done by some continuous size. Therefore we must allocate thread and memory in continous order. For example, memory of $0, \cdots, 99$ must be allocated in block 1, memory of $100, \cdots, 199$ must be allocated in block 2, and so on.

3) Phase only correlation function: Phase only correlation function is 2D IDFT of normalized cross spectrum. We use CUFFT for this 2D IDFT.

4) Peak estimation: At first values of phase only correlation function are examined pixel by pixel. Then the pixel where the value is maximum is detected. Then sub-pixel fitting near the maximum pixel is done.

C. The details of the implementation at the code level

In order to explain clearly, we compare two codes. The first one is RIPOC code (figure 8) in C language which runs total in CPU. The second one is RIPOC code in CUDA. The CPU only code is summarized as follow.

12-13 line
: 2D FFT of initial images ... `FFT`

14-21 line
: polar coordinates conversion of the of spectrum amplitude ... `POL`

22 line
: POC operation ... `POC`

23 line
: rotation of image ... `ROT`

24 line
: POC operation ... `POC`

The `my_FFTW` function includes data shift operation mentioned before.

```
 1 struct PocData {
 2   double min_dx,  min_dx2;
 3   double min_dy,  min_dy2;
 4   double score_min;
 5   double max;
 6   int peak_x,  peak_y;
 7 };

 8 PocData
 9 cal_poc_on_cpu (double *img1, double *img2, int X, int Y)
10 {
11   PocData  result;

12   my_FFTW(img1,  fftw_out1);
13   my_FFTW(img2,  fftw_out2);

14   for(int i=0 ; i<Y*(X/2+1) ; ++i) {
15     fftw_out1_power[i] = log( sqrt( fftw_out1[i][0]*fftw_out1[i][0] +
16                         fftw_out1[i][1]*fftw_out1[i][1] ) );
17     fftw_out2_power[i] = log( sqrt( fftw_out2[i][0]*fftw_out2[i][0] +
18                         fftw_out2[i][1]*fftw_out2[i][1] ) );
19   }
20   xy2rs(fftw_out1_power,  polar_coordinates1);
21   xy2rs(fftw_out2_power,  polar_coordinates2);

22   my_poc(polar_coordinates1, polar_coordinates2, NULL, &result);

23   my_rot(img1,  result.min_dy/Y*M_PI,  img1);

24   my_poc(img1,  img2,  NULL, &result);

25   return (result);
26 }
```

Fig. 8. RIPOC in C language

Next, we compare GPU code with above CPU only code. The GPU code is summarized as follow.

11-14 line
: data transfer form CPU to GPU ... `CPY`

15-18 line
: data shift, and CUFFT ... `FFT`

19-22 line
: polar coordinates conversion of the spectrum amplitude ... `POL`

```
 1 struct PocData {
 2   double min_dx,   min_dx2;
 3   double min_dy,   min_dy2;
 4   double score_min;
 5   double max;
 6   int peak_x,   peak_y;
 7 };


 8 PocData  cal_poc_on_gpu(float *img1, float *img2,  int X,  int Y)
 9 {
10   PocData   result;


11   cudaMemcpyAsync( d_img1,  img1, sizeof(float)*Y*X,
12               cudaMemcpyHostToDevice, my_stream);
13   cudaMemcpyAsync( d_img2,  img2, sizeof(float)*Y*X,
14               cudaMemcpyHostToDevice, my_stream);


15   my_shift_exe2<<<BLKNUM,THLNUM>>>(d_img1, d_shift1,
16                         d_img2, d_shift2, X, Y);
17   cufftExecR2C(fft_plan_r2c, d_shift1, d_ffted1);
18   cufftExecR2C(fft_plan_r2c, d_shift2, d_ffted2);


19   my_power_exe2<<<BLKNUM, THLNUM>>>(d_ffted1, d_pow1,
20                         d_ffted2, d_pow2, Y*(X/2+1));
21   my_xy2rs_exe2<<<BLKNUM, THLNUM>>>(d_pow1, d_rs1,
22                         d_pow2, d_rs2, X, Y);


23   my_poc_on_gpu( d_rs1, d_rs2, NULL, &result , X, Y);


24   my_rot_exe<<<BLKNUM, THLNUM>>>(d_img1, d_rs1,
25                         result.min_dy/Y*M_PI, X, Y);


26   my_poc_on_gpu( d_rs1, d_shift2, NULL, &result, X, Y);

27   return (result);
28 }
```

Fig. 9. RIPOC in CUDA

23 line
execute of POC ... `POC`
24-25 line
rotation of image ... `ROT`
26 line
execute of POC ... `POC`

Except the initial image data transfer, there is very little data transfer in this code and every process use GPU internal memory. List I shows breakdown of processing time for RIPOC in CPU and GPU [8].

TABLE I
BREAKDOWN OF PROCESSING TIME (MILLISECOND)

	256x256		512x512		1024x1024	
	GPU	CPU	GPU	CPU	GPU	CPU
CPY	0.18	-	0.48	-	1.64	-
FFT	0.30	4.47	1.17	23.30	4.23	111.62
POL	0.37	8.02	1.78	32.75	6.26	134.45
POC	0.70	33.99	2.04	67.88	6.97	223.91
ROT	0.11	1.10	0.41	4.42	1.59	17.83
POC	0.70	34.20	2.04	64.63	6.96	224.37
Total	2.36	81.78	7.92	192.98	27.65	712.18

TABLE II
EXECUTING TIME COMPARISON

Function	Cinderella II @ 100MHz	FPGA XC2V6000 @ 66MHz	Core 2 Quad @ 2.4GHz	GeForce 8800GTS
2D-FFT	8.88ms	2.01ms	2.23ms	0.13ms
POC	27.15ms	4.51ms	82.7ms	2.36ms

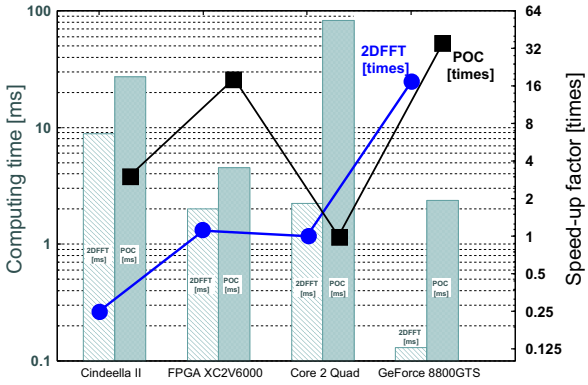

Fig. 10. Executing time comparison

IV. EXPERIMENT RESULT

List 2 and figure 10 show comparison result. There are four cases. The first one is ASIC implementation [3], the second one is FPGA implementation [9], the third one is CPU (Core 2 Quad) implementation, the last one is proposed GPU implementation. And this comparison is done at 256×256 pixels image size. There are little condition difference. The first and the second POC exclude RIPOC, the third and the last POC include RIPOC. Processing time of only POC operation is 34ms on Core 2 Quad, and 0.7ms on GPU. Estimation of sub-pixel displacement is done by 11×11 points fitting. The gettimeofday function is used for estimating of processing time. The processing time doesn't contain memory allocation time and memory free time.

V. ARGUMENT

The processing time on GPU is about 10 times faster than ASIC. However, it is necessity to have more debate because ASIC data is very old and methods for estimating sub-pixel displacement are different. The processing time on GPU is about twice faster than FPGA. However, it is necessity to have more debate because FPGA implementation excludes RIPOC and there are few data of FPGA implementation. The processing time on GPU is 30.6 times faster than core 2 quad CPU. However, it is necessity to have more debate because only one core of CPU is used. If 4 core and single precision float point operation are used, CPU performance become octuple. In addition, this experiment uses only one GPU. There must be more consideration about the case of using plural GPUs on one PC or plural GPUs on plural PCs. This implementation uses CUDA general-purpose FFT library CUFFT for FFT operations. The performance of POC and RIPOC can be improved if dedicated FFT routine is

implemented. For example, the mixing two FFT operations of two images increases the total performance, because two FFT operation have great amount of common part. Because the normalised cross spectrum's amplitude is 1, it is possible to simplify IFFT operation. This also contributes to the speedup.

VI. CONCLUSION

We have devised the speedup technique using GPU(Graphics Processing Unit), and have implemented POC(Phase Only Correlation) and RIPOC(Rotation Invariant Phase Only Correlation) operations on Nvidia GeForce8800 GTS with CUDA environment. In this implementation, we use the CUFFT function of CUDA library for 2D FFT. At first, elapsed time of the other part is ten times longer than the elapsed time of CUFFT. After tuning the memory layout and the order of each processes, elapsed time of CUFFT is about half of total processing time.

We have also compared the GPU implementation with ASIC or FPGA implementations. The processing time is 2.36ms at 256×256 image size, 7.92ms at 512×512 image size, and 27.65ms at 1024×1024 image size on one GPU. POC/RIPOC operation is more 10 times faster than dedicated LSI and 2 times faster than FPGA.

REFERENCES

[1] C. D. Kuglin, D. C. Hines, "The phase correlation image alignment method", Proc. Int. Conf. Cybernetics and Society, pp. 163-165, 1975.

[2] Q. S. Chen, M. Defrise, F. Deconinck, "Symmetric phase-only matched filtering of Fourier-Mellin transforms for image registration and recognition", IEEE Trans. Pattern Anal. Mach. Intell. ,vol. 16, no. 12, pp. 1156-1168, Dec. 1994.

[3] M. MAKOTO, K. ATSUSHI, I. HIDEAKI, "Development of "Cinderella II" Image Processing LSI", Savemation Rev, vol. 17, no. 2, pp. 64-71, 1999.

[4] K. Ito, H. Nakajima, K. Kobayashi, T. Aoki, and T. Higuchi, "A fingerprint matching algorithm using phase-only correlation", IEICE Trans. Fundamentals, vol. E87-A, no. 3, pp. 682-691, March 2004.

[5] S. Yamamoto and S. Hirai, "Robust and Video-frame Rate Tracking of Planar Motion by Matched Filtering on FPGA", Proc. of IEEE Int. Conf. on Robotics and Automation, pp. 4707-4712, New Orleans, April, 2004.

[6] T. Kenji, A. Takafumi, S. Yoshifumi, H. Tatsuo, K. Koji, "High-Accuracy Subpixel Image Registration Based on Phase-Only Correlation", IEICE trans. fundamentals, vol. E86-A, no. 8, pp. 1925-1934, august 2003.

[7] K. Takita, T. Aoki, Y. Sasaki, T. Higuchi, K.Kobayashi, "A sub-pixel correspondence search technique for computer vision applications", IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences, 2004.

[8] M. Frigo, and S. G. Johnson, "The Design and Implementation of FFTW3", Proceedings of the IEEE, vol. 93, no. 2, pp. 216-231, 2005.

[9] K. Shimizu, S. Hirai, "Realtime and Robust Motion Tracking by Matched Filter on CMOS+FPGA Vision System", Proc. IEEE Int. Conf. on Robotics and Automation, pp. 788-793, Rome, April, 2007.