22

# Fast Evolutionary Image Processing using Multi-GPUs

Jun Ando and Tomoharu Nagao
*Yokohama National University*
*Japan*

## 1. Introduction

In the realization of machine intelligence, image processing and recognition technologies are gaining in importance. However, it is difficult to construct image processing in each problem. In this case, a general-purpose method that constructs image processing without depending on problems is necessary.
On the other hand, Evolutionary Computation studies are widely applied to image processing. Evolutionary Computation is an optimizing algorithm inspired by evolutional processes of living things. We have previously proposed a system that automatically constructs an image-processing filter: Automatic Construction of Tree-structural Image Transformation (ACTIT). In this system, ACTIT approximates target image processing by combining tree-structurally several image-processing filters prepared in advance with genetic programming (GP), which is a type of Evolutionary Computation. We have proven that ACTIT is an effective method for many problems.
However, such complex image processing requires a great deal of computing time to optimize tree-structural image processing if ACTIT is applied to a problem that uses large and numerous images. Therefore, it is important to obtain fast evolutionary image processing. Some methods allow us to obtain fast processing, improve the algorithm, and implement fast hardware and parallel processing.
In this chapter, we employ a Graphics Processing Unit (GPU) as fast hardware to ACTIT for realization of fast image processing optimization. Moreover, the system calculates in parallel using multiple GPUs and increases in speed. We experimentally show that the optimization speed of the proposed method is faster than that of ordinary ACTIT.
This chapter is composed of the following. Section 2 discusses related works, ACTIT, General Purpose GPU (GPGPU), and parallel processing in Evolutionary Computation. Section 3 describes Multi-GPUs-ACTIT, which is the proposed system in this chapter. Section 4 experimentally shows that the proposed system is effective. Finally, section 5 describes our conclusions and future work.

## 2. Related works

### 2.1 ACTIT

ACTIT is a study of image processing using GP. It automatically constructs a tree-structural image transformation by combining several image-processing filters prepared in advance

Source: Image Processing, Book edited by: Yung-Sheng Chen,
ISBN 978-953-307-026-1, pp. 572, December 2009, INTECH, Croatia, downloaded from SCIYO.COM

with GP by referring to training image sets. The individual in GP is a tree-structural image transformation. A tree-structural image transformation is composed of input images as terminal nodes, non-terminal nodes in the form of several types of image-processing filters, and a root in the form of an output image.

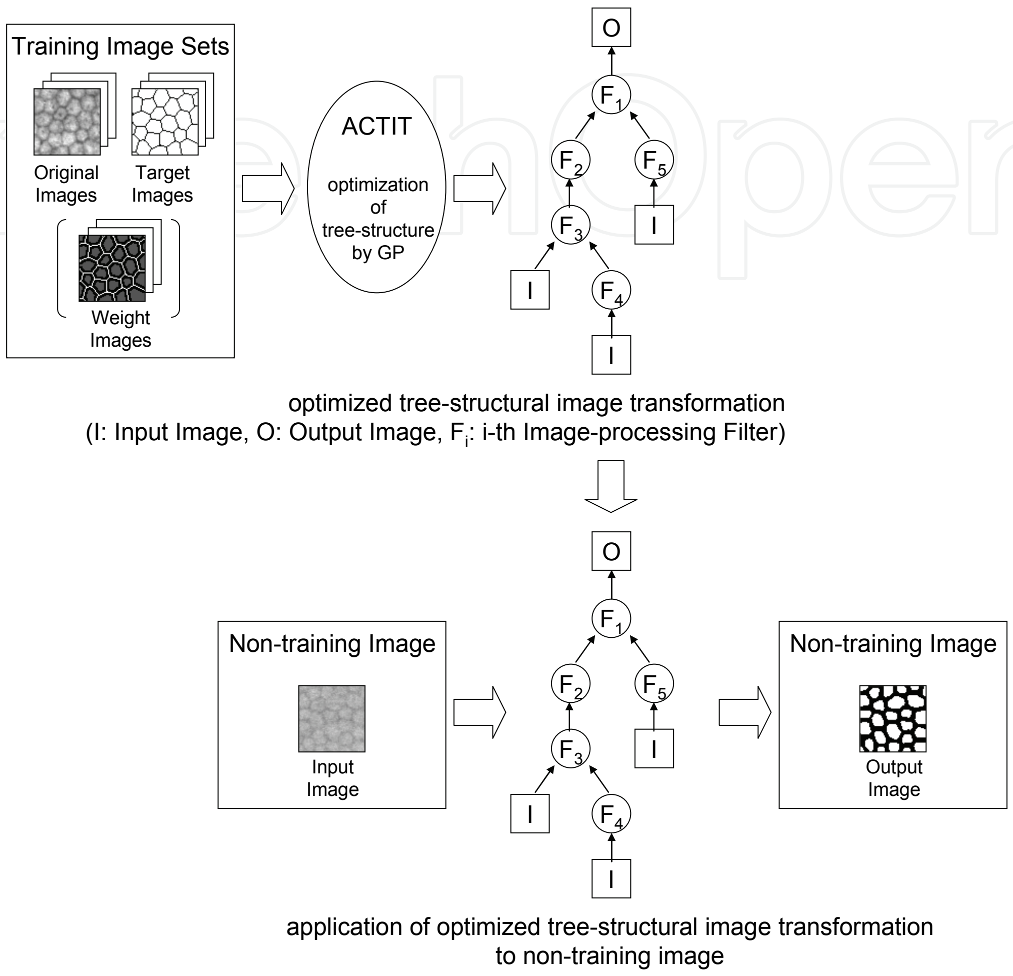

Training Image Sets
Original Images
Target Images
Weight Images
ACTIT
optimization of tree-structure by GP
O
F1
F2
F5
F3
I
F4
optimized tree-structural image transformation
(I: Input Image, O: Output Image, Fi: i-th Image-processing Filter)
Non-training Image
Input Image
Output Image
application of optimized tree-structural image transformation to non-training image

Fig. 1. The processing flow of the ACTIT system.

Figure 1 shows the processing flow of the ACTIT system. Training image sets are prepared, including several original images, their target images and weight images that indicate the important degree of pixel. We set the parameters that GP uses to optimize the tree structure and feed the training image sets to ACTIT. Then, ACTIT optimizes the tree-structural image transformation by means of GP. As a result, we can obtain an optimized tree-structural image transformation that has maximum fitness.
The tree-structural image transformation applies a certain processing mechanism to images that have the same characteristics. If the constructed tree-structural image transformation is appropriate, we can expect similar effects to the images that have the same characteristics as those learned. We prove that ACTIT is an effective method for a number of problems, such as 2D image processing for the detection of defects and 3D medical image processing.

## 2.2 GPGPU

The computational power of GPU on general graphics boards has been improving rapidly. Simple computational power per unit time of GPU has previously been superior to that of

CPU. Past GPUs performed only fast fixed CG processing. However, the latest GPUs have graphics pipelines that can be freely programmed and replaced to perform complex CG processing. Thus, presently, in research that puts GPU to practical use for the general purpose of calculating, GPGPU is a popular technique.

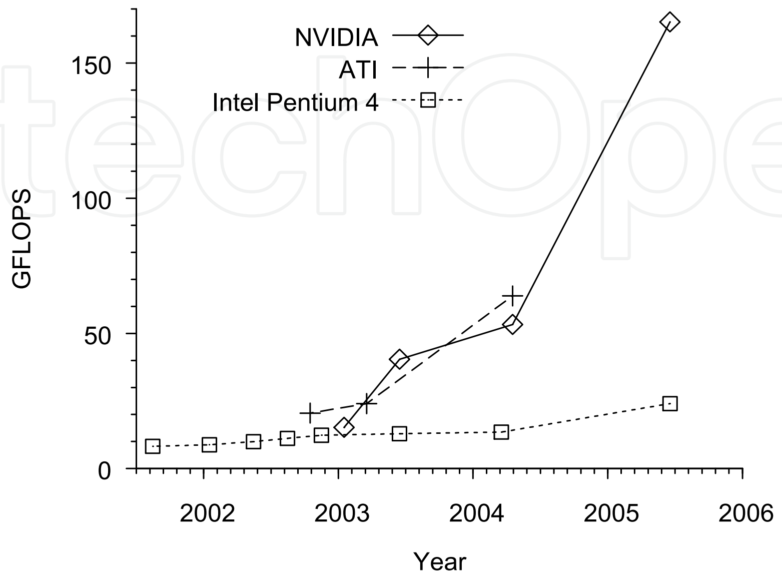


Fig. 2. The computational power of CPU and GPU in recent years.

| | |
|---|---|
| 2000 | NVIDIA released DirectX 8 which supports programmable *shader* architecture for the first time. |
| 2001 | NVIDIA GeForce 3 series GPU, which actually supports programmable *shader* architecture, appeared. |
| 2002 | NVIDIA released the 3D graphics language, "Cg (C for graphics)". |
| 2004 | Research report on GPGPU, "GP2" is held in Los Angeles for the first time. |
| 2005 | GPGPU session is newly established at CG festival SIGGRAPH sponsored by the Association for Computing Machinery (ACM). |

Table 1. The history of GPGPU.

Figure 2 shows the progress of the computational power of CPU and GPU over the past several years. Simple computational power per unit time of GPU has previously been superior to that of CPU during this time. The growth rate per year of GPU has also been superior to that of CPU.
Table 1 shows the history of GPGPU. Studies relating to GPGPU have only recently begun. NVIDIA GeForce 3 series GPU, which in practice supports programmable *shader* architecture, appeared in 2001. In 2002, NVIDIA released a high-level *shader* language Cg (C for graphics) and a toolkit that includes its compiler. Cg is a 3D graphics language similar to C language, and NVIDIA co-developed Cg with Microsoft. Formerly, it was necessary to code by hand with the assembly language to program using GPU. However, presently it is possible to generate an optimized code; GPU made by NVIDIA is the best technique for use

with Cg. In 2005, GPGPU session was established at the CG festival SIGGRAPH, sponsored by the Association for Computing Machinery.
GPU programming is without doubt different from CPU programming. For instance, GPU does not have random access memory space that can be freely read and written to when it calculates. GPU has an architecture specializing in parallel processing. This means that GPU is a stream processor. Therefore, GPGPU is effective for applications that satisfy the following three demands:

- Processed data are of a huge size.
- There is little dependency between each data.
- The processing of data can be highly parallel.

Therefore, GPGPU is effective for calculating matrices, image processing, physical simulations, and so on. Recently, programming languages specializing in GPGPU, Sh, Scout and Brook have been released. In addition, in 2006, NVIDIA released CUDA (Compute Unified Device Architecture), which performs general-purpose applications on GPU. Thus it is now relatively easy to program with GPU.

### 2.3 Parallel processing in evolutionary computation

Many studies have proven the performance of genetic algorithm (GA) and GP in parallel. The following show the main parallel models in GA and GP.

1. Island model: In an island model (Fig. 3), the population in GA and GP is divided into sections of population (Islands). Each section of population is assigned to multiple processors and applied to normal genetic operators in parallel. Exchange of individuals between sections of population (Migration) is performed. Each section is independently evaluated. Therefore, we expect that each section retains the variety of the entire population.

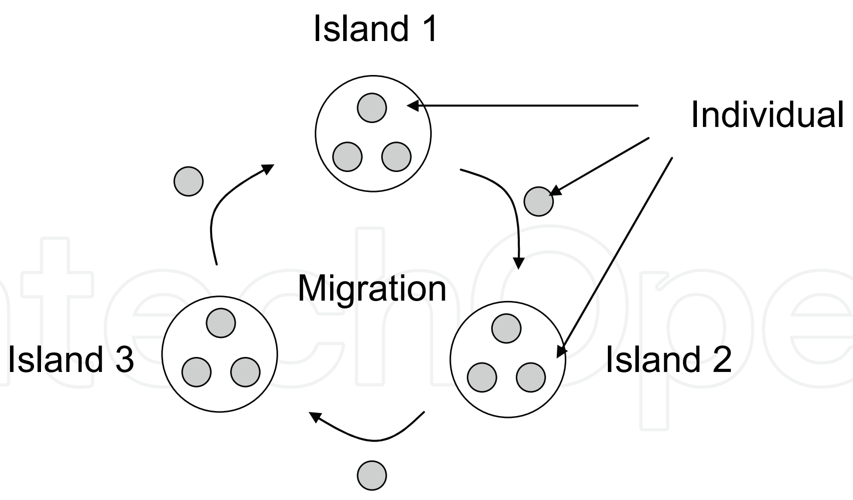


Fig. 3. Island model.

2. Master–slave model: In the master–slave model (Fig. 4), the fitness of individuals in GA and GP is calculated quickly in parallel. A master–slave model is generally composed of one control node (Master) and multiple calculation nodes (Slave). In this model, one control node performs genetic operators composed of selection, crossover, and mutation. Multiple calculation nodes share the task of calculating the fitness of individuals that consume computing time.

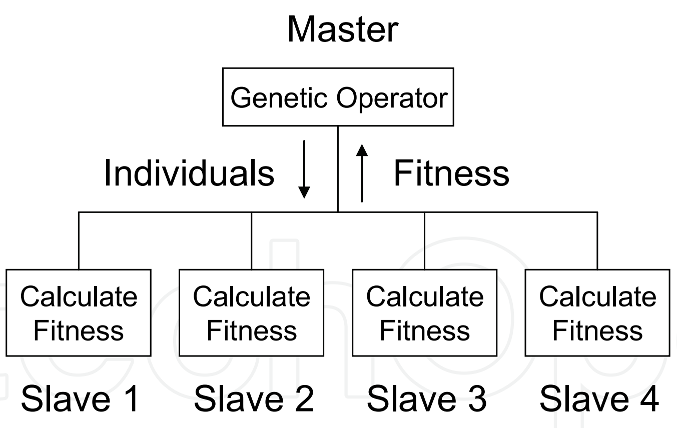


Fig. 4. Master-slave model.

3. Parallel-MGG model: The Parallel-MGG model (Fig. 5) is based on the master–slave model for fast processing. In the Parallel-MGG model, a control node sends two individuals as parents to calculation nodes. Each calculation node updates two individuals using Minimal Generation Gap (MGG) in parallel. A control node then receives two individuals of the next generation as children from each calculation node. In Parallel-MGG, the transport time between nodes is reduced because processing is asynchronous.

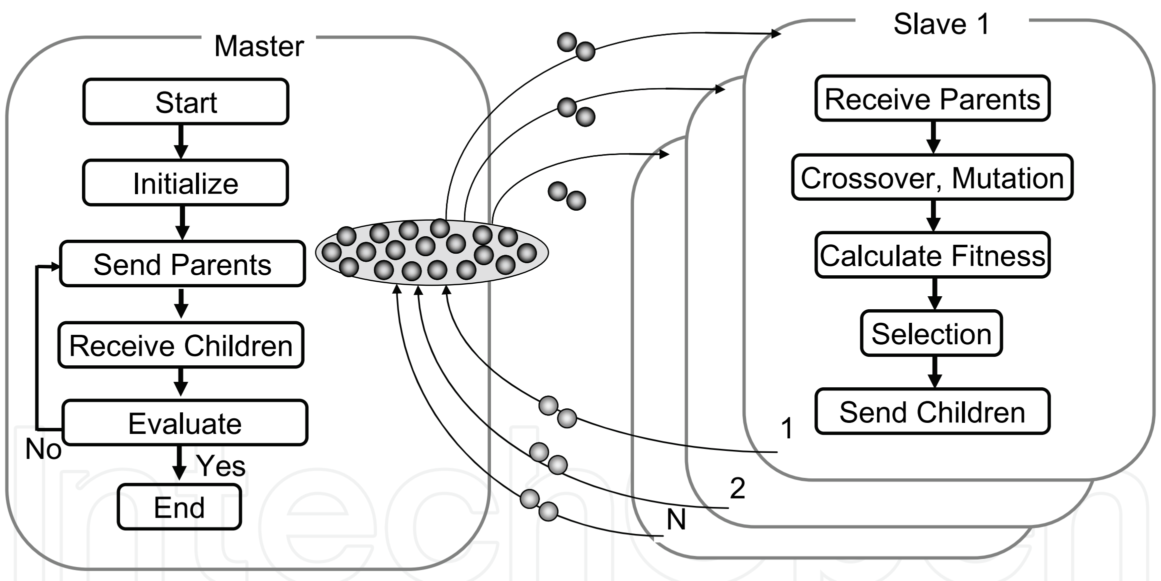


Fig. 5. Parallel-MGG model.

## 3. Fast evolutionary image processing

### 3.1 GPU-ACTIT

ACTIT requires a large amount of computing time to optimize tree-structural image processing when applied to a problem that uses large and numerous training image sets, because it needs to repeatedly create tree-structural image transformations and calculate their fitness. The computing time of the image transformation part of ACTIT accounts for 99% of the entire computing time. We therefore implement image-processing filters on programmable graphics pipelines of GPU for the purpose of reducing optimization time.

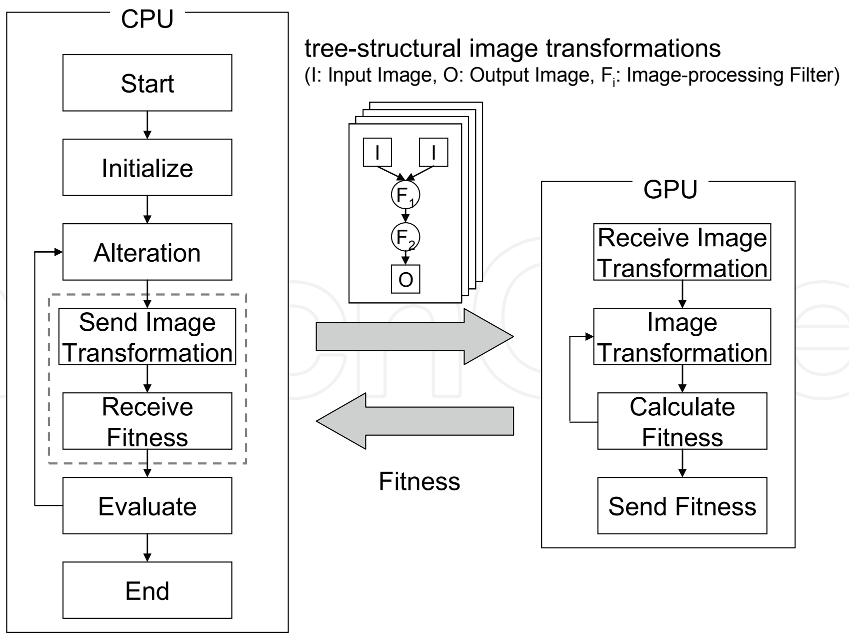
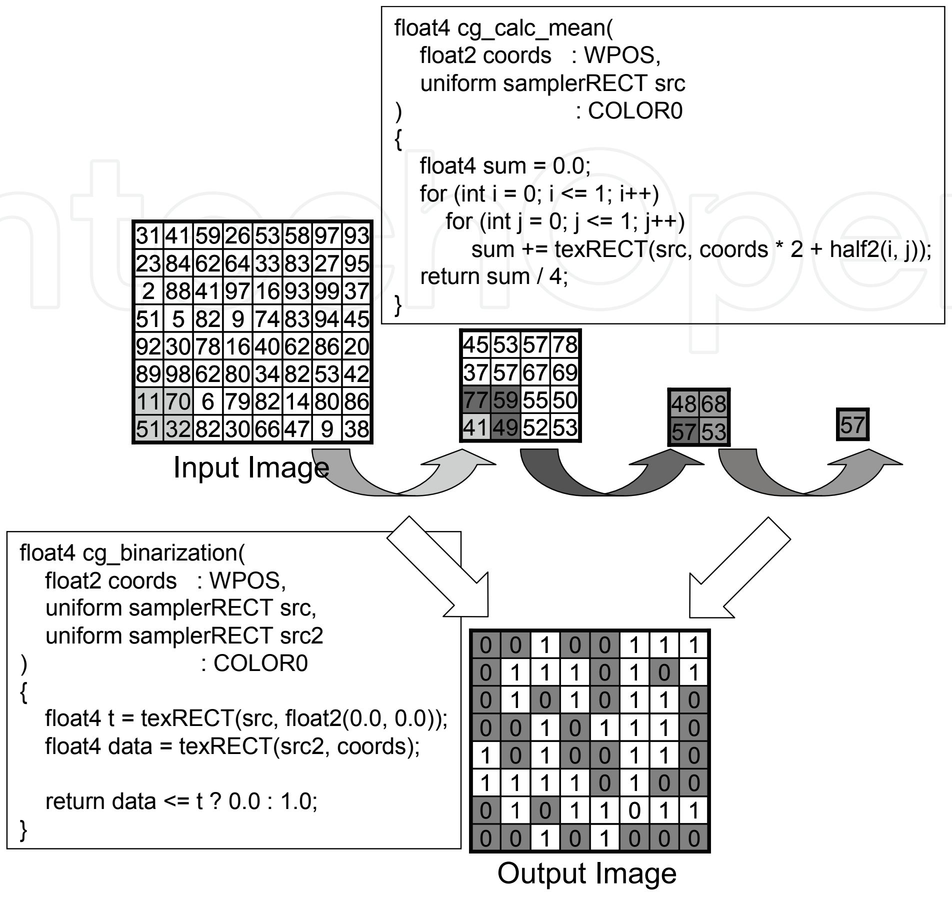

CPU
Start
Initialize
Alteration
Send Image Transformation
Receive Fitness
Evaluate
End
tree-structural image transformations
(I: Input Image, O: Output Image, $F_i$: Image-processing Filter)
I
I
$F_1$
$F_2$
O
GPU
Receive Image Transformation
Image Transformation
Calculate Fitness
Send Fitness
Fitness

Fig. 6. GPU-ACTIT.

1. The parts of CPU and GPU: Figure 6 shows the processing flow of CPU and GPU for the proposed system. First, the system loads training image sets and image-processing filters, which are written in Cg and compiled to GPU during initialization. The system executes the alternation of generations part, composed of selection, crossover, and mutation operators, of GP on CPU. It then performs image transformation and calculates fitness on GPU.
   CPU indicates the image-processing filter and its target image that GPU executes from the image filters of tree-structural image transformation to GPU one by one during image transformation. GPU performs tree-structural image transformation according to CPU. GPU calculates the fitness of each individual, i.e., tree-structural image transformation from the difference between the target image and the output image that is a result of image transformation in calculating fitness part. These processes are repeated until the fitness of all updated individuals per iteration are calculated. CPU reads back the fitness from GPU immediately.
   The system repeats these processes until the fitness of the best individual becomes 1.0 or the iteration number becomes max. Finally, we obtain an optimized tree-structural image transformation that has maximum fitness. We can obtain faster ACTIT by reducing the number of transporting data between CPU and GPU by loading training image sets firstly and returning fitness at once. We almost allow GPU to perform processing which costs computing time.
2. Implement on GPU: Programs written for CPU cannot be applied to GPU directly, because GPU has some limitations over CPU. Therefore, we are currently implementing only simple image-processing filters on GPU. The following describes several image-processing filters implemented on GPU:
   - Calculation of current and neighboring pixels (Mean Filter, Sobel Filter, and so on).
   - Calculation of two images (Difference Filter and so on).
   - Calculation of mean, maximum, minimum value in the whole image (Binarization with Mean Value, Linear Transformation of Histogram, and so on).

Figure 7 shows a Binarization filter (mean value). We calculate fast mean value in the whole image with parallel reductions.



```
float4 cg_calc_mean(
    float2 coords   : WPOS,
    uniform samplerRECT src
)                       : COLOR0
{
    float4 sum = 0.0;
    for (int i = 0; i <= 1; i++)
        for (int j = 0; j <= 1; j++)
            sum += texRECT(src, coords * 2 + half2(i, j));
    return sum / 4;
}
```

|  |  |  |  |  |  |  |  |
|---|---|---|---|---|---|---|---|
| 31 | 41 | 59 | 26 | 53 | 58 | 97 | 93 |
| 23 | 84 | 62 | 64 | 33 | 83 | 27 | 95 |
| 2 | 88 | 41 | 97 | 16 | 93 | 99 | 37 |
| 51 | 5 | 82 | 9 | 74 | 83 | 94 | 45 |
| 92 | 30 | 78 | 16 | 40 | 62 | 86 | 20 |
| 89 | 98 | 62 | 80 | 34 | 82 | 53 | 42 |
| 11 | 70 | 6 | 79 | 82 | 14 | 80 | 86 |
| 51 | 32 | 82 | 30 | 66 | 47 | 9 | 38 |

|  |  |  |  |
|---|---|---|---|
| 45 | 53 | 57 | 78 |
| 37 | 57 | 67 | 69 |
| 77 | 59 | 55 | 50 |
| 41 | 49 | 52 | 53 |

|  |  |
|---|---|
| 48 | 68 |
| 57 | 53 |

```
float4 cg_binarization(
    float2 coords   : WPOS,
    uniform samplerRECT src,
    uniform samplerRECT src2
)                       : COLOR0
{
    float4 t = texRECT(src, float2(0.0, 0.0));
    float4 data = texRECT(src2, coords);

    return data <= t ? 0.0 : 1.0;
}
```

|  |  |  |  |  |  |  |  |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 1 | 0 | 0 | 1 | 1 | 1 |
| 0 | 1 | 1 | 1 | 0 | 1 | 0 | 1 |
| 0 | 1 | 0 | 1 | 0 | 1 | 1 | 0 |
| 0 | 0 | 1 | 0 | 1 | 1 | 1 | 0 |
| 1 | 0 | 1 | 0 | 0 | 1 | 1 | 0 |
| 1 | 1 | 1 | 1 | 0 | 1 | 0 | 0 |
| 0 | 1 | 0 | 1 | 1 | 0 | 1 | 1 |
| 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 |

Fig. 7. Binarization filter (mean value).

### 3.2 Proposed parallel model

GPU-ACTIT is performed in parallel using multiple GPUs for fast processing. Parallel processing is effective for ACTIT, because the computing time of the parallelable part of ACTIT accounts for most of the entire computing time.

Figure 8 shows Multi-GPUs-ACTIT. The proposed system is composed of multiple PCs that have one GPU. The factors that prevent the system from achieving fast processing are synchronous time and transport time. There is no synchronous time, because processing is asynchronous in Parallel-MGG. Moreover, we can improve Parallel-MGG for the purpose of reducing transport time. In this new Parallel-MGG, the waiting buffer is located in each calculation node. The individual is sent to the waiting buffer in advance. Subsequent processing then starts as soon as the previous processing is finished, since the waiting buffer is utilized.

## 4. Experiments

### 4.1 Experimental setting

Here, we compare the optimization speed of the proposed system with ordinary ACTIT. The proposed system is composed of five PCs (one server and four clients) connected by a LAN network. Figure 9 shows the outside of the system.

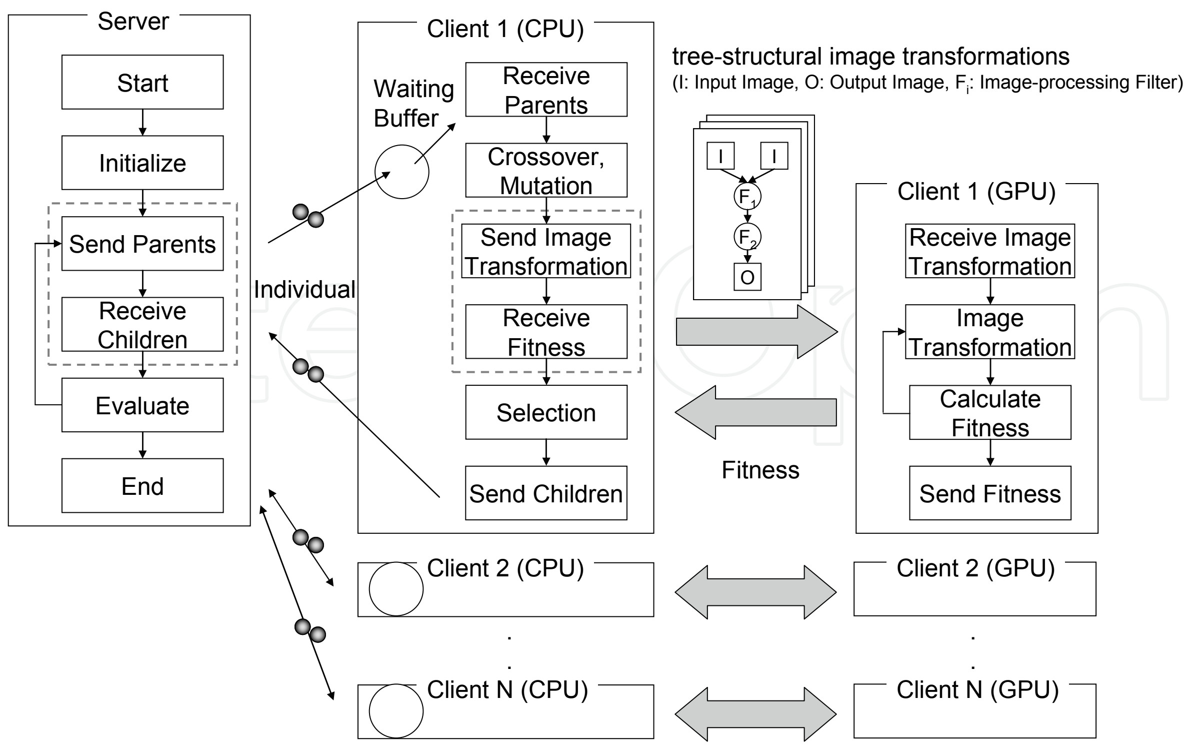


Fig. 8. Multi-GPUs-ACTIT.

Fig. 9. The outside of the system.

Table 2 shows the specifications of the PC used. Intel Core 2 Duo E6400 CPU and NVIDIA GeForce 7900 GS GPU are utilized in these experiments. We program with GPU using OpenGL and Cg.

We implement 37 types of one or two input and one output simple image-processing filters. GPU can calculate four planes (red, green, blue, and alpha) at the same time. Therefore, we prepare four training image sets. The dimensions of each image are 64 × 64, 128 × 128, 256 × 256, 512 × 512, and 1024 × 1024, respectively. GP parameters employ common values. The alternation model used by GP is MGG.

| | |
|---|---|
| PC | DELL Dimension 9200 |
| CPU | Intel Core 2 Duo E6400 |
| RAM | 2GB |
| Graphics Board | NVIDIA GeForce 7900 GS (G71) |
| Network | 100Mbps Ethernet |
| OS | Microsoft Windows XP Professional SP2 |
| Graphics API | OpenGL |
| *Shader* Language | Cg (C for graphics) |

Table 2. Specifications of the PC used.

### 4.2 Experimental results

1. Comparison of ordinary CPU-ACTIT and one GPU-ACTIT: First, we compared the optimization speed of one GPU-ACTIT with ordinary CPU-ACTIT. Figure 10 and Table 3 show the experimental results. The horizontal axis denotes image size, and the vertical axis denotes optimization speed. In Fig. 10 and Table 3, values are based on the optimization speed of 1.0 for CPU-ACTIT using images of 64 × 64.

| Image Size | CPU-ACTIT | GPU-ACTIT |
|---|---|---|
| 64 × 64 | 1.0 | 6.3 |
| 128 × 128 | 0.7 | 21.1 |
| 256 × 256 | 0.9 | 73.3 |
| 512 × 512 | 1.1 | 102.0 |
| 1024 × 1024 | 1.3 | 104.3 |

Table 3. Details of experimental results of a comparison of CPU-ACTIT and GPU-ACTIT.

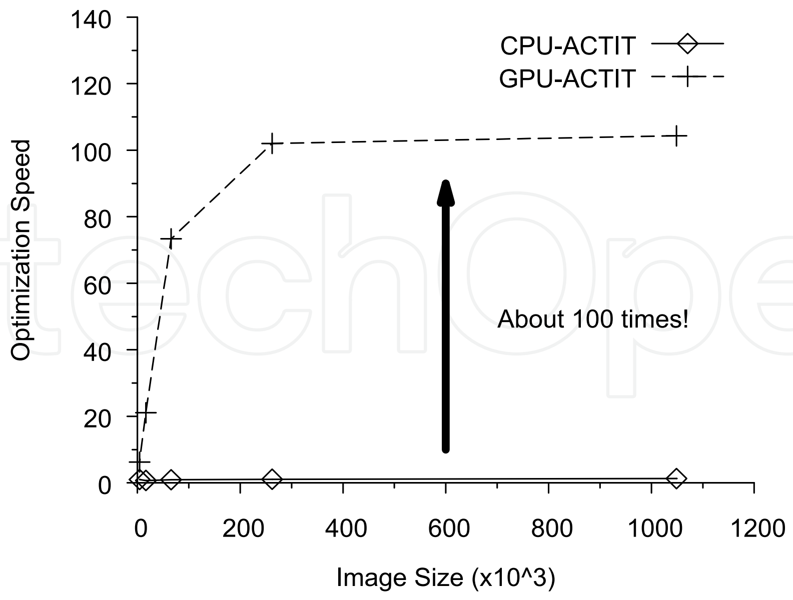


Fig. 10. Experimental results of a comparison of CPU-ACTIT and GPU-ACTIT.

The optimization of GPU-ACTIT was about 10 times faster than that of CPU-ACTIT with a small image, while it was about 100 times faster with a large image. It is well known that

GPU is effective when it uses large data. Therefore, the proposed method is very effective, because large and numerous training image sets tend to be used in real problems.
Next, we experimented to explain the influence of transporting data and synchronous time between CPU and GPU. Figure 11 shows details of the processing time. "GPU-ACTIT (inefficiently)" loads and reads images whenever it calculates the fitness of an individual. Loading and reading images influence the performance. As a result, the proposed method essentially performed the process that costs large computing time on GPU.

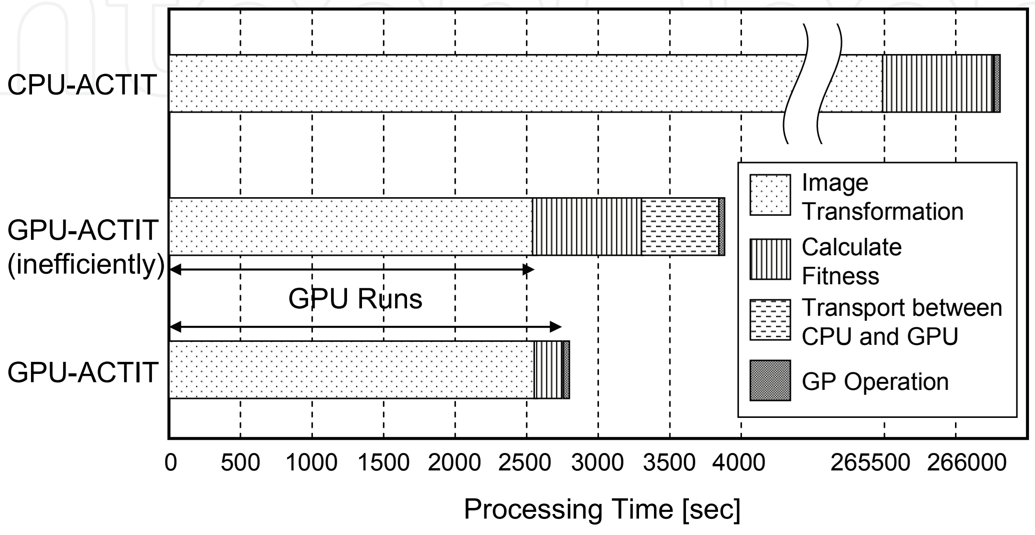


Fig. 11. Details of processing time.

2. Parallel of GPU-ACTIT: We compared the optimization speed of Multi-GPUs-ACTIT with one GPU-ACTIT. Parallel models used were master–slave, Parallel-MGG, and Parallel-MGG with waiting buffer. The number of GPUs was 1–4. Image size was only 512 × 512.

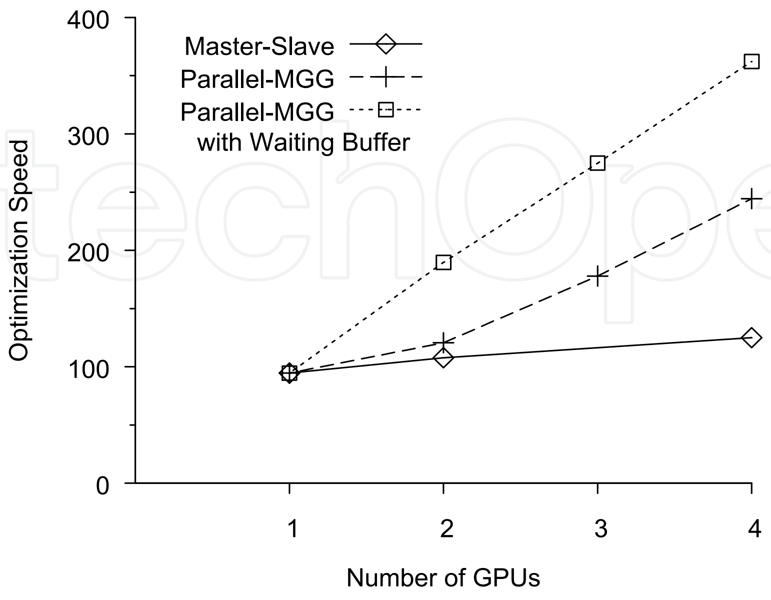


Fig. 12. Experimental results of Multi-GPUs-ACTIT.

| Number of GPUs | CPU-ACTIT | Master-slave | Parallel-MGG | Parallel-MGG with Waiting Buffer |
|---|---|---|---|---|
| 1 | 1.0 | 94.5 (1.0) | 94.5 (1.0) | 94.5 (1.0) |
| 2 | - | 107.7 (1.1) | 120.6 (1.3) | 189.7 (2.0) |
| 3 | - | - | 177.9 (1.9) | 274.9 (2.9) |
| 4 | - | 125.0 (1.3) | 244.4 (2.6) | 362.2 (3.8) |

Table 4. Details of experimental results of Multi-GPUs-ACTIT.

Figure 12 and Table 4 show experimental results. The horizontal axis denotes the number of GPUs, and the vertical axis denotes optimization speed. In Fig. 12 and Table 4, values are based on the optimization speed of 1.0 for CPU-ACTIT using 512 × 512 images. Parenthetic values are based on the optimization speed of 1.0 for one GPU-ACTIT using 512 × 512 images.

The optimization of four GPUs-ACTIT was about 3.8 times faster than that of one GPU-ACTIT in the proposed parallel model. The optimization of four GPUs-ACTIT was about 360 times faster than that of CPU-ACTIT in the proposed parallel model. We experimentally showed that the proposed parallel method is efficient.

## 5. Conclusions

We employed GPU to ACTIT for the purpose of reducing optimization time. The proposed method essentially performed the process that costs large computing time on GPU. Moreover, we proposed an efficient parallel model and instructed GPU-ACTIT to perform in parallel using multiple GPUs for fast processing. The optimization of the proposed method was several hundred times faster than that of the ordinary ACTIT. We experimentally showed that the proposed method is effective.

In future work, we plan to implement complex filters for ACTIT using only CPU that can be implemented on GPU. We propose image-processing algorithms that are effective for GPGPU. We also aim to construct a fast evolutionary image-processing system.

## 6. References

Aoki, S. & Nagao, T. (1999). ACTIT; Automatic Construction of Tree-structural Image Transformation, *The Institute of Image Information and Television Engineers*, Vol.53, No.6, pp.888-894.

Buck, I.; Foley, T.; Horn, D.; Sugerman, J.; Fatahalian, K.; Houston, M. & Hanrahan, P. (2004). Brook for GPUs: Stream Computing on Graphics Hardware, *SIGGRAPH 2004*.

Erick Cantu-Paz. (1998). A survey of parallel genetic algorithms, *Calculateurs Paralleles,* Vol.10, No.2.

Fernando, R. & Kilgard, M. J. (2003). *The Cg Tutorial: The Definitive Guide to Programmable Real-Time Graphics,* the Addison Wesley, ISBN978-0321194961.

Fernando, R. (2004). *GPU Gems: Programming Techniques, Tips, and Tricks for Real-Time Graphics,* the Addison Wesley, ISBN978-0321228321.

Fung, J.; Mann, S. & Aimone, C. (2005). OpenVIDIA: Parallel GPU Computer Vision, *Proceedings of the ACM Multimedia 2005,* pp.849-852, Nov.6-11, Singapore.

GoldBerg, D. E. (1989). *Genetic Algorithms in Search, Optimization & Machine Learning,* the Addison Wesley, ISBN978-0201157673.

Holland, J. H. (1975, 1992). *Adaptation in Natural and Artificial Systems,* the Univ. Michigan Press, ISBN978-0472084609, MIT Press, ISBN978-0262581110.

Koza, J. R. (1992). *Genetic Programming on the Programming of Computers by Means of Natural Selection,* MIT Press, ISBN978-0262111706.

Mura, H.; Ando, J. & Nagao, T. (2006). A research on fast tree-structural image transformation using PC cluster, *The Institute of Image Information and Television Engineers Technical Report*, Vol.30, No.17, pp.87-88, Japan.

Nagao, T. (2002). *Evolutionary Image Processing,* Shokodo, ISBN978-4785690632, Japan.

Nakano, Y. & Nagao, T. (2006). Automatic Extraction of Internal Organs Region from 3D PET Image Data using 3D-ACTIT, *International Workshop on Advanced Image Technology 2006,* Jan.10, Okinawa, Japan.

Owens, J. D.; Luebke, D.; Govindaraju, N.; Harris, M.; Kruger, J.; Lefohn, A. E. & Purcell, T. J. (2005). A Survey of General-Purpose Computation on Graphics Hardware, *EUROGRAPHICS 2005*.

Pharr, M. & Fernando, R. (2005). *GPU Gems 2: Programming Techniques for High-Performance Graphics and General-Purpose Computation,* the Addison Wesley, ISBN978-0321335593.

Sato, H.; Ono, I. & Kobayashi, S. (1997). A New Generation Alternation Model of Genetic Algorithms and its Assessment, *The Japanese Society for Artificial Intelligence*, Vol.12, No.5, pp.734-744.

Tanese, R. (1989). Distributed Genetic Algorithms, *Proc. 3rd International Conference on Genetic Algorithms,* pp.434-439.