High-Speed Calculation for Tissue Characterization of Coronary Plaque by Employing Parallel Computing Techniques

Takanori Koga, Shota Furukawa, Eiji Uchino and Noriaki Suetake

***Abstract*—In recent years, remarkable progress can be seen in the field of computer-aided medical diagnosis of ischemic coronary arterial disease. Intravascular ultrasound (IVUS)-based tissue characterization of coronary plaque is a significant topic in this field. The authors have proposed the multiple k-nearest neighbor (MkNN) classifier for the tissue characterization of coronary plaque in an IVUS B-mode image. Although its characterization performance was highly evaluated, the calculation speed was too slow to use actually in medical practice. The purpose of this study is to accelerate the speed of MkNN classifier aiming for it to be actually used in the medical practice. Recently, some parallel computing techniques on central processing unit (CPU) or on graphics processing unit (GPU) have come into general usage. Especially, the general purpose computation technique on Graphics Processing Unit (GPGPU) has got into the limelight recently. In this study, the calculation speeds of the MkNN classifier are evaluated for cases of various implementations using the parallel computing techniques. By employing GPGPU technique, it has been confirmed that its speed has been drastically accelerated enough for the practical use.**

***Index Terms*—Acute Coronary Syndromes (ACS), Intravascular Ultrasound (IVUS) Method, Multiple k-Nearest Neighbor, Parallel Computing, Pattern Classification, Pixel Classification, Tissue Characterization, GPGPU**

I. Introduction

An intravascular ultrasound (IVUS) method [1] is a tomographic imaging technology, which is often used for the diagnosis of acute coronary syndromes (ACS) [2], [3] in the field of cardiology. The IVUS method provides thousands of two-dimensional cross-sectional images of plaque in coronary artery. A medical doctor diagnoses the coronary plaque by carefully observing the B-mode images [4], [5], which are constructed by the intravascular ultrasound signals. For supporting the interpretation of the B-mode images and estimation of the structure of coronary plaque[6], some computer-aided diagnosis methods had been proposed so far [7], [8], [9], [10], [11], [12].

In order to realize a precise tissue characterization of coronary plaque to support the medical doctors, we have proposed the multiple k-nearest neighbor (MkNN) classifier so far [13], [14], [15]. The MkNN classifier classifies coronary plaque pixel by pixel. The MkNN classifier considers the spatial continuity of distribution of the data, not only in the feature space but also in the observation space, and thus performs a fine classification even if the distributions of data for each class overlap with each other in the feature space.

However, the MkNN classifier takes too much calculation time for classification. This is because the calculation process of the MkNN classifier is described by deeply-nested iteration structure in the program code. Therefore, the speed-up of the MkNN classifier is a supreme order to make it used in the medical practice.

Recently, some parallel computing techniques on the central processing unit (CPU) or on the graphics processing unit (GPU) have been used in the various fields [16]. The Open Multi-Processing (OpenMP) [17], [18] application program interface (API) makes multi-platform shared-memory parallel programming easy on a variety of single and multi-CPU computer with C/C++ or Fortran language. On the other hand, especially, the general purpose computation technique on Graphics Processing Unit (GPGPU) [19], [20] has got into the limelight in various fields of science and technology [21], [22]. The calculation performance of the latest GPU is extremely higher than that of CPU. Moreover, by using the Compute Unified Device Architecture (CUDA) library supplied by NVIDIA Corporation [23], it is easy to develop a general purpose program processed on GPU with C language.

In this paper, in order to realize a speed-up of the MkNN classifier [13], [14], [15], it is implemented by employing GPGPU technique. First, we verify its performance using a benchmark problem by comparison to other implementations of parallel computing. We then apply it to the real tissue characterization problem of coronary plaque in the IVUS image. This is our main concern.

II. Intravascular Ultrasound (IVUS) Method

Intravascular ultrasound (IVUS) method [1], [4] is one of the medical imaging techniques. It allows the application of ultrasound technology to observe from inside the blood vessels

Manuscript received April 8, 2011.

This work was supported in part by the strategic program for promoting research of Yamaguchi University, 2010. Many thanks are due to Dr. R. Kubota for his helpful assistance.

T. Koga is with the Department of Computer Science and Electronic Engineering, Tokuyama College of Technology, Gakuendai, Shunan 745–8585, JAPAN, Phone and Fax:+81-834-29-6319; Email: koga@tokuyama.ac.jp

S. Furukawa, E. Uchino and N. Suetake are with the Graduate School of Science and Engineering, Yamaguchi University, 1677-1 Yoshida, Yamaguchi 753-8512, JAPAN. Phone and Fax:+81-834-29-6319; Email: {n016vc@, uchino@, suetake@sci.}yamaguchi-u.ac.jp

E. Uchino is the Chair of Fuzzy Logic Systems Institute (FLSI), 680-41 Kawazu, Iizuka 820-0067, JAPAN

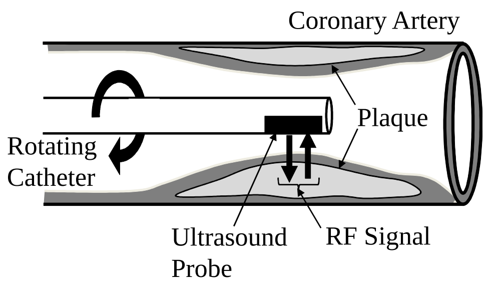


Fig. 1. Overview image of the intravascular ultrasound (IVUS) method.

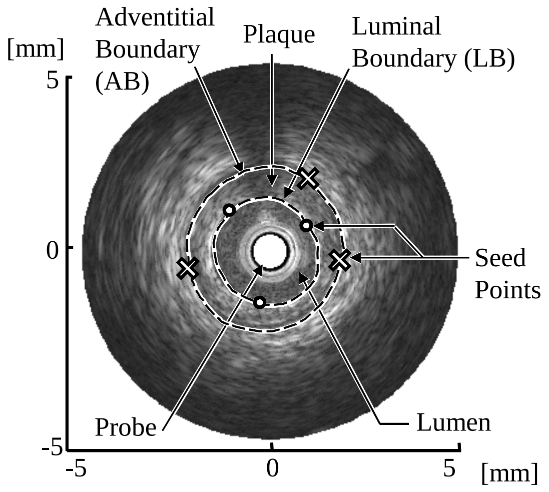


Fig. 2. An example of B-mode image obtained by the IVUS method. In the image, 4mm corresponds to 2,048 pixels.

out through the surrounding blood column, visualizing the coronary plaque in vivo.

In the IVUS method, a specially designed thin catheter with the ultimately-miniaturized ultrasound probe attached to its distal end is used (see Fig. 1). To visualize the coronary arteries and plaques in vivo, angiographic techniques are used. A medical doctor steers the guidewire with very thin diameter from outside the body through angiography catheters into the coronary artery to be visualized. The ultrasound catheter tip is slid in over the guidewire and positioned, using angiography techniques so that the tip is at the farthest away position to be visualized. The guidewire is kept stationary and the ultrasound catheter tip is slid backwards, usually under motorized control.

The proximal end of the catheter is connected to a computerized ultrasound equipment. The ultrasound waves are emitted from the ultrasound probe, which are in the 40MHz range in this study, and the catheter also receives the reflected signal from the plaque tissue. It is sampled at 400 MHz and stored in the computerized ultrasound equipment.

An IVUS image is constructed of the amplitude information of the received radiofrequency (RF) signal. In order to visualize the inside of a coronary artery, the sampled RF signal is first transformed into an 8-bit luminal intensity signal by taking the absolute value of a signal, then by taking the envelope of a signal, and finally by taking its logarithmic value.

The luminal intensity signals in all radial directions are then formed to obtain a tomographic cross sectional image of a coronary artery as shown in Fig. 2. The IVUS image of Fig. 2 is called "B-mode image." A B-mode image displays a real time ultrasound cross-sectional image of a thin section of a blood vessel currently surrounding a catheter probe.

In this study, the B-mode image is constructed with 2,048 pixels in depth, and 256 lines in radial direction. Hence, the resolution of distance and angle are 1.95μm/pixel and 1.41°/line, respectively.

In the qualitative assessment of coronary plaque, the area of coronary plaque surrounded by the following two boundaries is extracted in the IVUS B-mode image as shown in Fig. 2. One is a luminal boundary (LB) between the lumen and the plaque, and the other is an adventitial boundary (AB) between the plaque and the vascular wall. Then the area of coronary plaque is characterized by using the diagnosis methods in order to analyze its composition and structure.

III. MkNN Classifier

Fig. 3 shows the pixel classification procedure of the multiple k-nearest neighbor (MkNN) classifier [13], [14], [15] for an example of a two-class classification problem. In Fig. 3, the observation space is an observed image, and the feature vector space consists of the feature vectors obtained at each pixel of the observed image. In general, the feature vectors are calculated usually, *e.g.*, by principal component analysis of an image, Fourier transformation, and so on.

In this paper, for the tissue characterization of coronary plaque, the feature vectors are calculated by the Fourier transformation of radio frequency (RF) signal, which constitutes the IVUS B-mode image [4], [5]. The training feature vectors (prototypes) are fed to the MkNN classifier in the same manner as to the ordinary kNN classifier.

That is, in the MkNN classifier, the classification is carried out for the feature vector obtained at each pixel $\boldsymbol{p} = (x_1, x_2)$ of interest (POI) in the observation space as shown in Fig. 3. In this figure, note that k represents "the number of neighboring pixels $\boldsymbol{p}_m$ of $\boldsymbol{p}$ (POI) in the observation space," and k' represents "the number of training feature vectors $\boldsymbol{u}_m$ in the feature space corresponding to pixels $\boldsymbol{p}_m$ in the observation space."

The substantial difference between the ordinary kNN classifier and the MkNN classifier is as follows. In the MkNN classifier, not only the training feature vectors corresponding to the pixel of interest $\boldsymbol{p}$ (the pixel to be classified) but also those of neighboring pixels $\boldsymbol{p}_m$ of $\boldsymbol{p}$ in the observation space (*i.e.*, on the image) are used for the classification. That is, in the MkNN classifier, $(k \times k')$-nearest training feature vectors are selected for the training of $\boldsymbol{p}$ (POI). Finally, the majority decision considering the class labels of all the selected training feature vectors is made in order to classify $\boldsymbol{p}$ (POI). On the other hand, the ordinary kNN classifier only uses the class

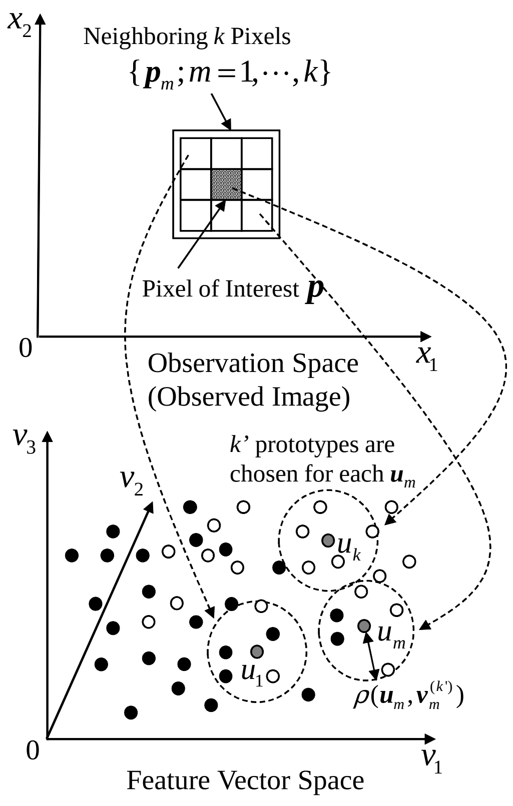

x_2
Neighboring k Pixels
$\{\boldsymbol{p}_m; m=1,\cdots,k\}$
Pixel of Interest $\boldsymbol{p}$
0
x_1
Observation Space
(Observed Image)
v_3
k' prototypes are chosen for each $\boldsymbol{u}_m$
v_2
u_k
u_m
u_1
$\rho(\boldsymbol{u}_m, \boldsymbol{v}_m^{(k')})$
0
v_1
Feature Vector Space

Fig. 3. Overview of the MkNN classifier.

labels of k'-nearest prototypes corresponding to $\boldsymbol{p}$ (POI) in the majority decision making.

The following is a brief description of the procedure of the MkNN classifier.

Step1: Select the neighboring pixels $\{\boldsymbol{p}_m; m=1,\cdots,k\}$ of $\boldsymbol{p}$ (POI) in the observation space. Calculate $\boldsymbol{u}_m$ for $\boldsymbol{p}_m$.

Step2: Suppose that a set of N pairs $\{(\boldsymbol{v}_n, i_n); n = 1,\cdots,N\}$ are given, where $\boldsymbol{v}_n$ is a training feature vector which belongs to class $i_n \in \{1,\cdots,C\}$. C is the number of classes. Calculate the Euclidean distances $\|\boldsymbol{v}_n - \boldsymbol{u}_m\|$ between each $\boldsymbol{u}_m$ and all the training feature vectors $\{\boldsymbol{v}_n; n = 1,\cdots,N\}$.

Step3: The training feature vectors $\boldsymbol{v}_n$ which satisfy the following condition:

$$\|\boldsymbol{v}_n - \boldsymbol{u}_m\| \le \rho\left(\boldsymbol{u}_m, \boldsymbol{v}_m^{(k')}\right), \qquad (1)$$

are selected for each $\boldsymbol{u}_m$, where $\boldsymbol{v}_m^{(k')}$ is the k'-th nearest training feature vector around $\boldsymbol{u}_m$. $\rho(\boldsymbol{u}_m, \boldsymbol{v}_m^{(k')})$ is an Euclidean distance between $\boldsymbol{u}_m$ and $\boldsymbol{v}_m^{(k')}$ in the feature space.

Step4: The feature vector for pixel $\boldsymbol{p}$ in the observation space is classified into class c as follows:

$$c = \arg\max_j \sum_{m=1}^{k} \sum_{n=1}^{N} U(\boldsymbol{u}_m, \boldsymbol{v}_n, i_n, j), \qquad (2)$$

$$U = \begin{cases} 1, & \|\boldsymbol{v}_n - \boldsymbol{u}_m\| \le \rho\left(\boldsymbol{u}_m, \boldsymbol{v}_m^{(k')}\right) \text{ and } i_n = j \\ 0, & otherwise. \end{cases} \qquad (3)$$

Eq. (3) is a function for a majority decision making of a class label.

In the MkNN classifier, the spatial continuities both in the observation and feature spaces are utilized in the procedure of **Step3**. That is, in the MkNN classifier, $(k \times k')$-nearest training feature vectors are selected for each feature vector corresponding to $\boldsymbol{p}$ (POI) based on the spatial relationship between $\boldsymbol{p}$ and $\boldsymbol{p}_m$ in the observation space. This means that the characteristic of MkNN is to utilize the information on the spatial continuities in both the observation and feature spaces. The MkNN classifier thus realizes a fine pixel classification even for the B-mode image with heavy noises and/or measuring errors [13], [14], [15].

IV. Parallel Computation on CPU/GPU

In this study, for acceleration of the calculation of the MkNN classifier, some parallel computing implementations are compared to each other. The overviews of parallel computing on CPU or on GPU are described in the following subsections.

A. Parallel Computation on CPU

The parallel computing models using multi CPUs or a many-core CPU are categorized as a distributed memory type, a shared-memory type or a hybrid distributed shared-memory type.

In the distributed memory type, there is typically a processor, a memory, and some form of interconnection that allows programs on each processor to interact with each other. Message Passing Interface (MPI) is a de-facto standard in the development of parallel computing program for the distributed memory system. MPI is a library for data and message transfer, which can be used in C/C++ and Fortran program. In the programming with MPI, memory management and data transferring process are to be described obviously. This is somewhat troublesome, and portability of the program becomes low.

On the other hand, in computer hardware, shared memory refers to a (typically) large block of random access memory that can be accessed by several different central processing units (CPUs) in a multiple-processor computer system. For the parallel computing on CPU, the open multi-programming (OpenMP) [17], [18] or the portable operating system interface (POSIX) thread called Pthread, are broadly used.

Especially, OpenMP is a de-facto standard for shared-memory programming and most people in scientific field use OpenMP in case of shared memory parallelization. OpenMP is a set of compiler directives, callable runtime library routines and environmental variables that extend C/C++ or Fortran to express shared-memory parallelism. In practical use of OpenMP, scalable parallel computing on CPU is easily realized by inserting some compiler directives into key points

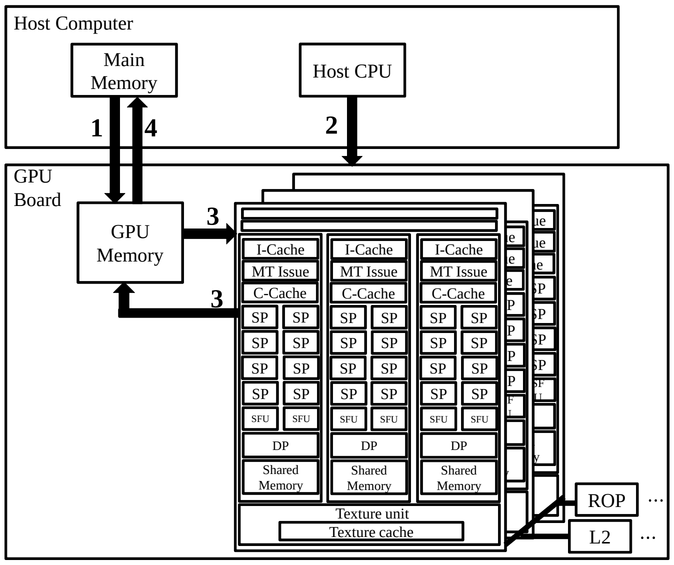


Fig. 4. Architecture of GPU computing board and execution process of GPGPU.

and combining some functions in a non-parallelized program code. In addition, highly portable software for shared-memory system can be realized by using OpenMP.

For these reasons, we employ OpenMP for the parallel computing implementation on CPU for comparisons to GPGPU.

B. Parallel Computation on GPU

To meet the strong demand for real-time, high-definition 3D graphics and multimedia experiences, the programmable graphics processing unit (GPU) has evolved into a massively parallel, multithreaded, many-core processor. The latest GPUs have tens of hundreds of fragment processors, and higher memory bandwidths than regular CPUs.

However, GPUs are designed specifically for graphics. Therefore, they are critically-constrained in terms of operations and programming. Due to such characteristic, GPUs are only effective at problems that can be solved using stream processing. Furthermore, the hardware can only be used in specific purposes.

Under such constraints, the idea behind general-purpose computing on graphics processing units (GPGPU) [19], [20] is to use GPUs to accelerate selected computations in applications that are traditionally handled by CPUs. It is made possible by the addition of programmable stages and higher precision arithmetic to the rendering pipelines, which allows software developers to use stream processing on non-graphics data.

In the field of GPGPU, Compute Unified Device Architecture (CUDA) [23], which is a parallel computing architecture developed by NVIDIA, is frequently used. By using CUDA, developers can access to the virtual instruction set and memory of the parallel computational elements in GPUs. In general, in the programming using CUDA, programmers write an application with two portions of code, functions to be executed on the CPU host (Host program) and functions to be executed on the GPU device (GPU kernel function), by using C language. Fig. 4 briefly shows the following sequence of steps involved in a typical CUDA kernel invocation.

TABLE I
SPECIFICATION OF TESLA C1060 GPU COMPUTING PROCESSOR BOARD USED IN THE EXPERIMENTS.

Peak Processing Performance	933 [GFLOPs]
Number of Processor Cores	240
Clock Speed	1,296 [MHz]
Memory Size	4 [GB]
Memory I/O	GDDR3 512 [bit]
Memory Clock	800 [MHz]

1. Copy the data from main memory to GPU memory.
2. CPU instructs GPU to start execution of the kernel.
3. GPU gets the data from GPU memory and execute the kernel in parallrel processing.
4. Calculation results are copied to main memory.

GPUs can only process independent vertices and fragments for graphics, but can process many of them in parallel. This is especially effective when the vertices or fragments are to be processed in the same manner. In this regard, GPUs are stream processors, which can operate in parallel by running a single kernel on many records in a stream at once. Furthermore, it is important for GPGPU applications to have high arithmetic intensity. Otherwise, the memory access latency will limit computational speed-up. Ideal GPGPU applications have massive data sets, high parallelism, and minimal dependency between data elements.

The pixel classification by the MkNN classifier is suitable for the parallel processing on GPU by using GPGPU technique. This is because the pixel classification process can be independently calculated pixel by pixel and the above-mentioned matters are satisfied.

V. EXPERIMENTS

A. Experimental Conditions

In order to compare the computational costs for various implementations, the following nine types of implementations are achieved. Below is a brief explanation of each implementation with an abbreviation label of each implementation.

- MATLAB: Program described by MATLAB script
- MATLAB+PCT: Program described by MATLAB script using Parallel Computing Toolbox (execution in parallel on CPU)
- MATLAB+PCT+BLAS: Program described by MATLAB script using Parallel Computing Toolbox, in which Basic Linear Algebra Subprograms (BLAS) library like distance calculation is performed (execution in parallel on CPU)
- C: Program described by C language (execution in parallel on CPU)
- C+OpenMP: Program described by C language with OpenMP (execution in parallel on CPU)

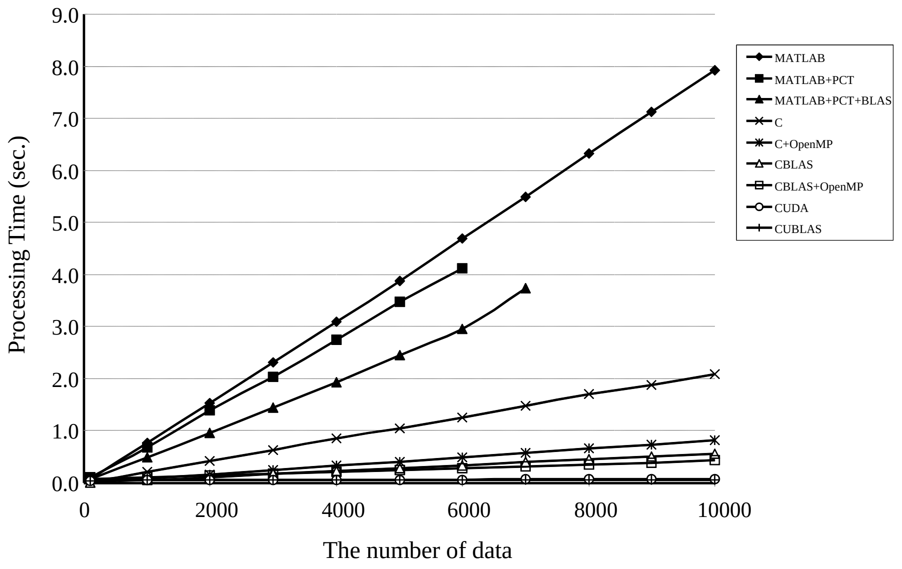


Fig. 5. Computational costs versus the number of data (pixels) to be classified. Experimental conditions: $k = 9$ (3×3), and $k' = 9$.

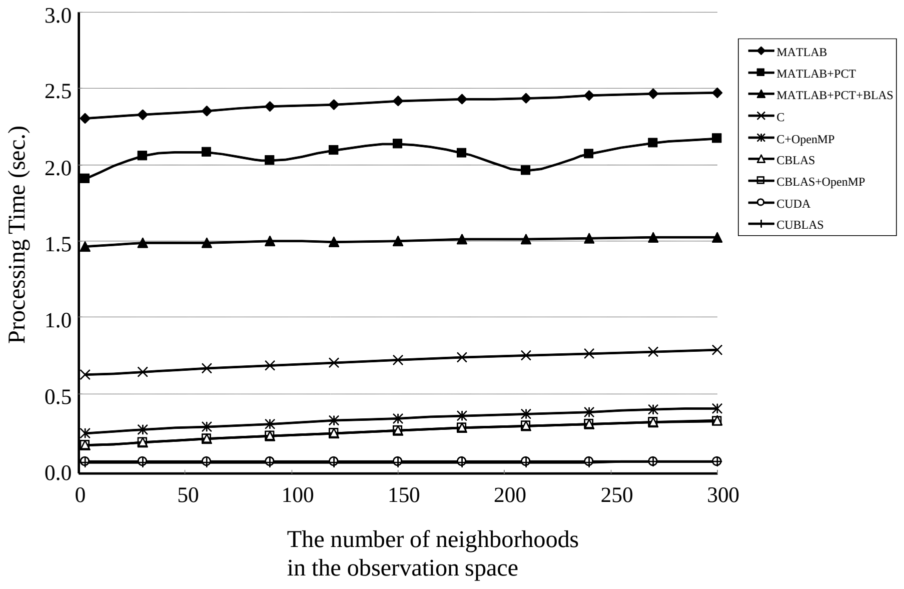


Fig. 6. Computational costs versus the number of neighboring data (pixels) in the data observation space to be classified. Experimental conditions: the number of data to be classified is 3,000, and $k' = 9$.

- CBLAS: Program described by C language and BLAS library (execution in parallel on CPU)
- CBLAS+OpenMP: CBLAS: Program described by C language, BLAS library and OpenMP (execution in parallel on CPU)
- CUDA: Program described by C language and CUDA (execution in parallel on GPU)
- CUBLAS: Program described by C language, CUDA, and CUBLAS (execution in parallel on GPU)

MATLAB is a numerical computation software produced by Mathworks. MATLAB's Parallel Computing Toolbox (PCT) is a library to produce functions for parallel computing on multi-core CPU. BLAS is a library for linear algebraic processing with respect to vectors and matrices. CUBLAS is an optimized BLAS library for GPGPU.

Note that Basic Linear Algebra Subprograms (BLAS) library like distance calculation in the MATLAB+PCT+BLAS makes the following reduction in terms of the distance calculation of Eq.(1). The Euclidian distance between a feature vector $\boldsymbol{u}_m$ and a training vector $\boldsymbol{v}_n$ is calculated by $\|\boldsymbol{v}_n - \boldsymbol{u}_m\|^2 = \boldsymbol{v}_n^T\boldsymbol{v}_n - 2\boldsymbol{u}_m^T\boldsymbol{v}_n + \boldsymbol{u}_m^T\boldsymbol{u}_m$. Here, $\boldsymbol{u}_m^T\boldsymbol{u}_m$ can be neglected as an unnecessary term in the evaluation of distance. Thus, the

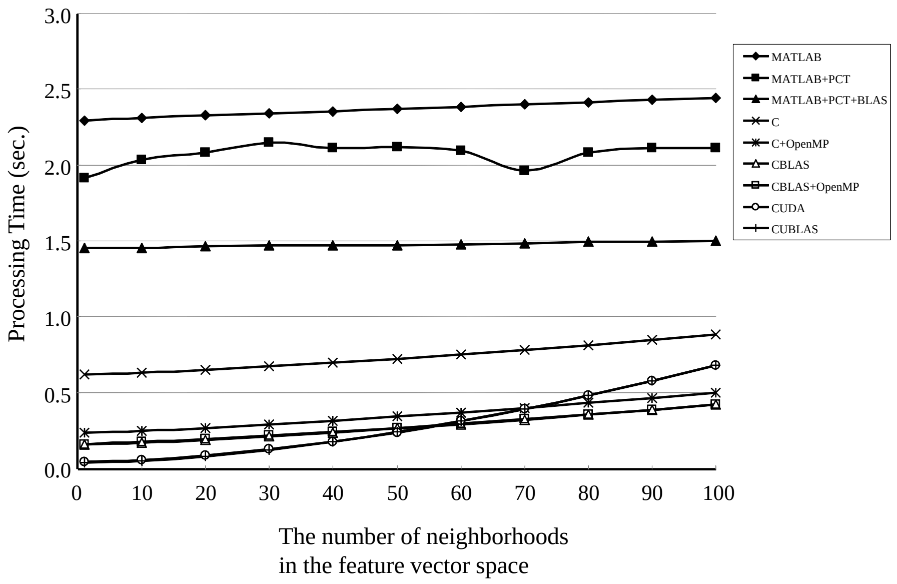


Fig. 7. Computational costs versus the number of neighboring vectors in the feature vector space. Experimental conditions: the number of data to be classified is 3,000, and $k = 9 (3 \times 3)$.

Euclidian distance between the feature vector and the training vector can be obtained only by calculating $\boldsymbol{v}_n^T \boldsymbol{v}_n - 2\boldsymbol{u}_m^T \boldsymbol{v}_n$. In the program code of the MATLAB+PCT+BLAS, this leads to a big reduction of calculation.

The computer used in the experiments has Intel Core i7 920 2.67GHz (with 4 cores) CPU and NVIDIA Tesla C1060 GPU computing processor board. The specification of Tesla C1060 computing processor board is described in Table I. Operating system (OS) is Microsoft Windows XP 32-bit. The GPU kernel function is described by CUDA. MATLAB's version is 2009b. Compiler of C language and its developing environment is Microsoft Visual Studio 2008 Express Edition with Windows SDK for Windows Server 2008 and .NET Framework 3.5. In the programs with OpenMP, the number of threads is set to be 8. In the programs with CUDA, the number of threads is set to be 240.

The program codes for GPGPU using CUDA library are constituted by a host program and a GPU kernel function. The host program consists of the program routines for data transfer to GPU and its control routines. The GPU kernel function consists of the program routines to be processed on the GPU. The data to be processed on GPU is transferred at once from the main memory of host machine to GPU memory in order to avoid data transfer latency. In addition, in programming of MkNN classifier, paper [24] is referred, which describes an implementation of the ordinary kNN classifier by using GPGPU technique.

All the programs are fully tuned with respect to program description for the parallel processing. For example, various points such as the position of the parallel for statement in PCT and OpenMP, data partitioning in GPU, and so on are empirically tuned in order to make the performance of each implementation best. This is because the performance of the parallel computation is greatly influenced by those factors.

B. Performance Validation by Using Randomly-Generated Data Set

For performance validation of GPGPU, an artificially-generated random data set is used. The feature vector is 16-dimensional real-valued vector. The number of classes is 11. The number of the training feature vectors is 3,000. The training feature vectors are sampled from 11 classes of data set with equal probability.

Under the above conditions, the computational costs are evaluated for the following 3 cases.

- Case 1: the number of data (pixels) to be classified is changed
- Case 2: the number of neighboring data (pixels) in the data observation space is changed
- Case 3: the number of neighboring vectors in the feature vector space is changed

The computational cost is evaluated by an average time of 100 iterations for each case. The parameters of the MkNN classifier are shown in the captions of Figs. 5, 6 and 7, respectively.

Fig. 5 shows the experimental results for case 1 versus the number of data (pixels) to be classified. In general, the computational cost monotonically increases with the number of data. On the contrary, in cases with CUDA and CUBLAS, the computational cost is very small independent of the number of data. The program using CUBLAS is a little bit faster than that using CUDA.

In the experimental results of MATLAB+PCT and MATLAB+PCT+BLAS, there are missings of result around 6,000 and 7,000 of the number of data, respectively. This is because a memory allocation was impossible due to a hardware restriction. It is thought that these are caused by the internal processes of memory allocation for parallel processing. That is, this comes from too huge memory area was tried to be allocated automatically for the parallel processing by PCT.

Fig. 6 shows the experimental results for case 2 versus the number of neighboring data (pixels) in the data observation space. It can be seen that the computational cost is independent of the neighboring data. Those results imply that each implementation method can process the MkNN classifier in approximately the same calculation time with the ordinary kNN classifier. It means that the MkNN classifier is an effective classifier because it can use the information in the data observation space in approximately the similar calculation time of the ordinary kNN classifier.

Fig. 7 shows the experimental results for case 3 versus the number of neighboring vectors in the feature vector space. The computational cost is in general independent of the neighboring data. The programs using CUDA and CUBLAS are a little bit influenced by the number of neighboring data. This is because the sorting algorithm is involved in the MkNN classifier to obtain the neighboring order in the feature vector space, which is not suitable for parallel processing. The similar phenomena have been reported in [24].

In cases of parallel computing on CPU, the performance of the programs with PCT or OpenMP has shown better than that of the program without using parallel computing. On the other hand, the programs parallely-executed on GPU with GPGPU technique has shown overwhelming performance. In addition, it was confirmed that the classification results were not affected in all the experiments with Tesla C1060, although Tesla C1060 computing processor board only supports single precision.

By the results of these experiments, the superiority and basic characteristics of the MkNN classifier on GPU has been verified.

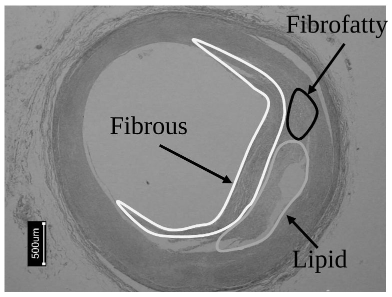
C. Real Application to Tissue Characterization of Coronary Plaque

Here we confirm the performance of the GPGPU technique applied to the tissue characterization of coronary plaque in the real IVUS image of a patient.

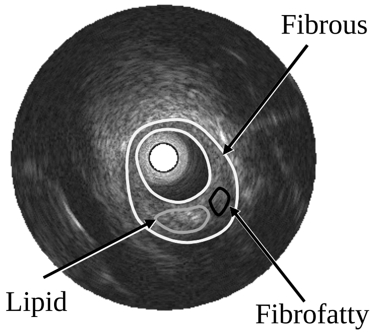
Fig. 8(a) is a microscopic image of stained tissue of test data. Fig. 8(b) is a desirable characterization result for the test data, which is obtained by medical doctor’s interpretation. The region of interest (ROI) surrounded by two white lines in Fig. 8(c) is classified into lipid tissue, fibrofatty tissue, and fibrous tissue pixel by pixel. In the experiments, the feature vector obtained at each pixel of a B-mode image is a power spectrum calculated by the short-time discrete FFT of a radio frequency (RF) signal in radial direction [13], [14], [15].

The short-time discrete FFT was carried out by shifting the window of a size of 32 pixels in depth direction of a radial line. The feature vector is 17-dimensional real-valued vector concerning spectrum. The feature vectors are normalized in the range of $[0, 1]$. The number of class is 3. The number of the training feature vectors is 195. The training feature vectors are sampled from 3 classes of data sets with equal probability.

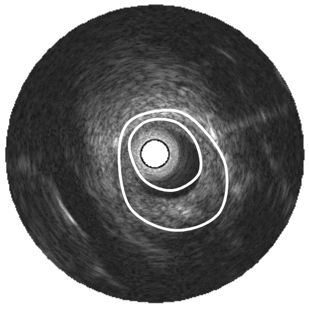
In the experiments, approximately 70,000 samples (pixels) per IVUS B-mode image are classified. In case of the MkNN



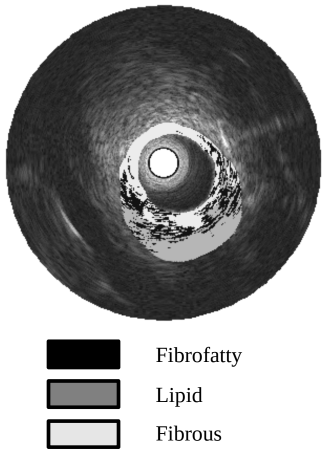
(a)



(b)



(c)



(d)

Fig. 8. Tissue characterization of coronary plaque in IVUS B-mode image. (a) Microscopic image of stained tissue of test data (b) Desirable characterization result for test data. (c) The area surrounded by two white lines is the region of interest (ROI) to be characterized. (d) Tissue characterization results by the MkNN classifier.

TABLE II
COMPUTATIONAL COST FOR EACH IMPLEMENTATION IN CLASSIFICATION OF IVUS DATA.

Implementation method	Calculation time (sec.)
MATLAB	13.12
MATLAB+PCT	6.00
MATLAB+PCT+BLAS	4.56
C	1.24
C+OpenMP	0.73
CBLAS	0.53
CBLAS+OpenMP	0.38
CUDA	0.07

classifier, the number of neighboring data in the feature vector space k' is set to be 9, and the number of neighborhood data in the observation space k is set to be 9 (3×3).

Fig. 8(d) shows the characterization result by the MkNN classifier. Table II shows the comparison of processing time by each implementation. In this regard, experimental result by CUBLAS could not be obtained partly because memory allocation was impossible due to a hardware restriction. The calculation time employing GPGPU (CUDA) is drastically reduced, which is good enough for a medical practice.

It was also confirmed as in the simulation experiments using artificial data set in the previous subsection that the classification results were not affected in all the experiments with Tesla C1060, although Tesla C1060 computing processor board only supports single precision.

With those experiments, tissue characterization of coronary plaque by the MkNN classifier has proposed in [13], [14], [15] reached almost near to the practical use.

VI. CONCLUSION

In this paper, we have implemented the MkNN classifier, a tissue characterization algorithm for coronary plaque, by using some parallel computing techniques. It has been shown that the calculation speed of the MkNN classifier had been drastically accelerated by using GPGPU technique. The possibility of the practical use of the tissue characterization by MkNN classifier has been greatly increased.

Future work is to further reduce the computing time by using GPU clusters aiming at the real practical use soon.

REFERENCES

[1] J. B. Hodgson, S. P. Graham, A. D. Savakus, S. G. Dame, D. N. Stephens, P. S. Dhillon, D. Brands, H. Sheehan, and M. J. Eberle, "Clinical percutaneous imaging of coronary anatomy using an over-the-wire ultrasound catheter system," *Int. J. Cardiac Imaging*, vol. 4, pp. 187–193, 1989.

[2] E. Falk, "Why do plaques rupture?," *Circulation*, vol. 86, pp. 30–42, 1992.

[3] E. Falk, P. K. Shah, and V. Fuster, "Coronary plaque disruption," *Circulation*, vol. 92, no. 3, pp. 657–671, 1995.

[4] B. N. Potkin, A. L. Bartorelli, J. M. Gessert, R. F. Neville, Y. Almagor, W. C. Roberts, and M. B. Leon, "Coronary artery imaging with intravascular high-frequency ultrasound," *Circulation*, vol. 81, pp. 1575–1585, 1990.

[5] J. D. Klingensmith, D. G. Vince, B. D. Kuban, R. Shekhar, E. M. Tuzcu, S. E. Nissen, and J. F. Cornhill, "Assessment of coronary compensatory enlargement by three-dimensional intravascular ultrasound," *Int. J. Cardiac Imaging*, vol. 16, pp. 87–98, 2000.

[6] G. J. Friedrich, N. Y. Moes, V. A. Muhlberger, C. Gabl, G. Mikuz, D. Hausmann, P. J. Fitzgerald, and P. G. Yock, "Detection of intralesional calcium by intracoronary ultrasound depends on the histlogic pattern," *Am. Heart J.*, vol. 128, pp. 435–441, 1994.

[7] D. T. Linker, A. Klevan, Å. Grø nningsæther, P. G. Yock, and Bj. A. J. Angelsen, "Tissue characterization with intra-arterial ultrasound: Special promise and problems," *Int. J. Cardiac Imaging*, vol. 6, pp. 255–263, 1991.

[8] M. Sonka, X. Zhang, M. Siebes, M. S. Bissing, S. C. DeJong, S. M. Collins, and C. R. McKay, "Segmentation of intravascular ultrasound images: A knowledge-based approach," *IEEE Trans. Med. Img.*, vol. 14, pp. 719–732, 1995.

[9] S. J. Nicholls, E. M. Tuzcu, I. Sipahi, P. Schoenhagen, and S. E. Nissen, "Intravascular ultrasound in cardiovascular medicine," *Circulation*, vol. 114, pp. 54–59, 2006.

[10] R. Kubota, M. Kunihiro, N. Suetake, E. Uchino, G. Hashimoto, T. Hiro, and M. Matsuzaki, "An intravascular ultrasound-based tissue characterization using shift-invariant features extracted by adaptive subspace SOM," *Int. J. of Biology and Biomedical Engineering*, vol. 2, no. 2, pp. 79–88, 2008.

[11] M. Kawasaki, H. Takatsu, T. Noda, K. Sano, Y. Ito, K. Hayakawa, K. Tsuchiya, M. Arai, K. Nishigaki, G. Takemura, S. Minatoguchi, T. Fujiwara, and H. Fujiwara, "In vivo quantitative tissue characterization of human coronary arterial plaques by use of integrated backscatter intravascular ultrasound and comparison with angioscopic findings," *Circulation*, vol. 105, pp. 2487–2492, 2002.

[12] M. Kawasaki, B. E. Bouma, J. Bressner, S. L. Houser, S. K. Nadkarni, B. D. MacNeill, I. K. Jang, H. Fujiwara, and G. J. Tearney, "Diagnostic accuracy of optical coherence tomography and integrated backscatter intravascular ultrasound images for tissue characterization of human coronary plaques," *J. Am. Coll. Cardiol.*, vol. 48, pp. 81–88, 2006.

[13] R. Kubota, M. Kunihiro, N. Suetake, E. Uchino, G. Hashimoto, T. Hiro, and M. Matsuzaki, "Intravascular ultrasound-based tissue classification of coronary plaque into fibrosis or lipid by k-nearest neighbor method," *Proc. of Int. Conf. on Soft Computing and Human Sci. -New Horizon beyond the 20th Anniversary of BMFSA*, pp. 93–96, 2007.

[14] R. Kubota, N. Suetake, and E. Uchino, "Hierarchical k-nearest neighbor classification using feature and observation space information," *IEICE Electronics Express*, vol. 5, no. 3, pp. 114–119, 2008.

[15] E. Uchino, N. Suetake, R. Kubota, T. Koga, G. Hashimoto, T. Hiro, and M. Matsuzaki, "An ROC performance validation of hierarchical k-nearest neighbor classifier applied to tissue characterization using IVUS-RF signal," *Proc. of 2009 Int. Workshop on Nonlinear Circuits and Signal Processing*, pp. 333–336, 2009.

[16] D. Wippig and B. Klauer, "GPU-based translation-invariant 2D discrete wavelet transform for image processing," *Int. J. of Comp.*, vol. 5, no. 2, pp. 226–234, 2011.

[17] M. J. Quinn, *Parallel Programming in C with MPI and OpenMP*, McGraw Hill Higher Education, 2003.

[18] D. J. Kuck, B. Chapman, G. Jost, and R. van der Pas, *Using OpenMP: Portable Shared Memory Parallel Programming (Scientific and Engineering Computation)*, The MIT Press, 2007.

[19] M. Pharr and R. Fernando, *GPU Gems 2: Programming Techniques for High-Performance Graphics and General-Purpose Computation*, Addison-Wesley Professional, 2005.

[20] D. B. Kirk and W. W. Hwu, *Programming Massively Parallel Processors: A Hands-on Approach (Applications of GPU Computing Series)*, Morgan Kaufmann, 2010.

[21] A. Victor, M. A. D. Bogdan, M. Florica, M. Anca, and E. Alexandru, "GPGPU for cheaper 3D MMO servers," *Proc. of the 9th Int. Conf. on Telecommunications and Informatics*, pp. 238–243, 2010.

[22] A. Moldoveanu, F. Moldoveanu, and V. Asavei, "More scalability at lower costs - server architecture for massive multiplayer 3D virtual spaces powered by GPGPU," *Int. J. of Computers and Communications*, pp. 117–126, 2007.

[23] J. Sanders and E. Kandrot, *CUDA by Example: An Introduction to General-Purpose GPU Programming*, Addison-Wesley Professional, 2010.

[24] V. Garcia, E. Debreuve, and M. Barlaud, "Fast k nearest neighbor search using GPU," *Proc. of the CVPR Workshop on Comp. Vision on GPU*, pp. 1–6, 2008.