Fast Parallel Processing using GPU in computing L1-PCA bases

Nobuhiro Funatsu
Kurume National College of Technology
1-1-1, Komorino, Kurume-shi, Fukuoka 830-8555 Japan
Tel: +81-942-35-9392 Fax: +81-942-35-9398
E-mail: s44236nf@std.kurume-nct.ac.jp

Yoshimitsu Kuroki
Kurume National College of Technology
1-1-1, Komorino, Kurume-shi, Fukuoka 830-8555 Japan
Tel: +81-942-35-9392 Fax: +81-942-35-9398
E-mail: kuroki@kurume-nct.ac.jp

***Abstract*—In data-analysis problems with a large number of dimensions, the principal component analysis based on L2-norm (L2-PCA) is one of the most popular methods, but L2-PCA is sensitive to outliers. Unlike L2-PCA, PCA-L1 is robust to outliers because it utilizes the L1-norm, which is less sensitive to outliers; therefore, some studies have shown the superiority of PCA-L1 to L2-PCA [2][3]. However, PCA-L1 requires enormous computational cost to obtain the bases, because PCA-L1 employs an iterative algorithm, and initial bases are eigenvectors of autocorrelation matrix. The autocorrelation matrix in the PCA-L1 needs to be recalculated for the each basis besides. In previous works [3], the authors proposed a fast PCA-L1 algorithm providing identical bases in terms of theoretical approach, and decreased computational time roughly to a quarter. This paper attempts to accelerate the computation of the L1-PCA bases using GPU.**

I. Introduction

In data-analysis problems with a large number of dimension, principal component analysis (PCA) is one of the most popular methods. PCA finds orthonormal bases, and given multivariable data are projected onto the subspace spanned by the bases. Various methods in order to obtain the bases are proposed, and the most popular method is the PCA based on L2-norm (L2-PCA). Some readers may recall Eigenface in the study fields of face recognition as one of the famous application of L2-PCA [1]. Projection values of data onto the bases in L2-PCA have the greatest number of variance. Although L2-PCA has been successful for many problems, the influence of outliers on obtaining the principal bases is significant due to the L2-norm criterion. The influence seems to be reduced by the PCA based on L1-norm (L1-PCA). Unlike L2-PCA, L1-PCA is robust to outliers because it utilizes the L1-norm, which is less sensitive to outliers. However, it is difficult to calculate exact solutions of L1-PCA. To solve this problem, Kwak proposes a feasible scheme employing a substitute formula based on L1-norm, designated as PCA-L1 [2], to obtain principal bases. The bases obtained by PCA-L1 are invariant to rotations. The detail of PCA-L1 is described in Section 2; PCA-L1 employs an iterative algorithm to compute each bases, and requires to calculate an eigenvector of autocorrelation matrix as an initial vector. The autocorrelation matrix is computed by the projected data onto the orthogonal complement of already calculated bases, and needs to be recalculated for each basis repeatedly. The authors decreased the PCA-L1 bases computation concerning the relation between the current autocorrelation matrix and the previous one. This paper expedites the bases computation by utilizing GPU.

The rest of this paper is organized as follows. In Section 2, outline of PCA-L1 is described. Section 3 expresses our fast PCA-L1 algorithm from theoretical approach. Section 4 introduces CUDA, a software environment of GPGPU. The performance of implementation on GPU is compared with the previous methods in Section 5, and we conclude in Section 6 with short summary.

II. PCA-L1 algorithm

Let $\boldsymbol{X} = [\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n] \in R^{d \times n}$ be the given data, where n and d denote the number of samples and the dimension of the original input space, respectively. Without loss of generality, $\boldsymbol{x}_i (i = 1, 2, \ldots, n)$ are assumed to have zero means.

In L2-PCA, one tries to find an $\boldsymbol{W}^*$ which is $m(< d)$ dimensional linear subspace. The $\boldsymbol{W}^*$ is the solution of the following dual problem:

$$\boldsymbol{W}^* = \underset{\boldsymbol{W}}{\operatorname{argmax}} \parallel \boldsymbol{W}^T \boldsymbol{S} \boldsymbol{W} \parallel_2 = \underset{\boldsymbol{W}}{\operatorname{argmax}} \parallel \boldsymbol{W}^T \boldsymbol{X} \parallel_2, \quad (1)$$
$$\text{Subject to} \quad \boldsymbol{W}^T \boldsymbol{W} = \boldsymbol{I}_m,$$

where $\boldsymbol{W} \in R^{d \times m}$ is the projection matrix whose columns $\boldsymbol{w}_k (k = 1, 2, \ldots, m)$ constitute the bases of the m-dimensional linear subspace (feature space), $\boldsymbol{S} = \boldsymbol{X}^T \boldsymbol{X}$ is the autocorrelation matrix of $\boldsymbol{X}$, $\boldsymbol{I}_m$ is the $m \times m$ identity matrix, and $\| \cdot \|_2$ denotes the L2-norm of a matrix or a vector. Since the methods based on the L2-norm are sensitive to outliers, we use the methods based on the L1-norm which is robust to outliers than the L2-norm.

In PCA-L1, one tries to find an $\boldsymbol{W}^*$ which is $m(< d)$ dimensional linear subspace. The $\boldsymbol{W}^*$ is the solution of the following dual problem:

$$\boldsymbol{W}^* = \underset{\boldsymbol{W}}{\operatorname{argmax}} \parallel \boldsymbol{W}^T \boldsymbol{X} \parallel_1, \quad (2)$$
$$\text{Subject to} \quad \boldsymbol{W}^T \boldsymbol{W} = \boldsymbol{I}_m.$$

Here, the constraint $\boldsymbol{W}^T \boldsymbol{W} = \boldsymbol{I}_m$ ensures the orthonormality of the projection matrix. The solution of (2) is invariant to

rotations because the maximization is done on the subspace and it is expected to be more robust to outliers than the L2 solution.

As a downside, finding a global solution of (2) for $m > 1$ is very difficult. To ameliorate this problem, Kwak simplify (2) into series of $m = 1$ problems using a greedy search method. If we set $m = 1$, (2) becomes the following optimization problem:

$$\boldsymbol{w}^* = \underset{\boldsymbol{w}}{\operatorname{argmax}} \parallel \boldsymbol{w}^T \boldsymbol{X} \parallel_1 = \sum_{i=1}^{n} \left| \boldsymbol{w}^T \boldsymbol{x}_i \right|, \quad \text{Subject to} \quad \parallel \boldsymbol{w} \parallel_2 = 1. \tag{3}$$

In the following, an algorithm to solve (3) is presented.

Algorithm: PCA-L1 ($m = 1$)

1) Initialization:
   Pick any $\boldsymbol{w}(t = 0)$. Set $\boldsymbol{w}(0) \leftarrow \boldsymbol{w}(0) / \parallel \boldsymbol{w}(0) \parallel_2$.
2) Polarity check:
   For all $i \in \{1, \ldots, n\}$, if $\boldsymbol{w}^T \boldsymbol{x}_i < 0, p_i(t) = -1$, otherwise $p_i(t) = 1$.
3) Flipping and maximization:
   Set $t \leftarrow t + 1$ and $\boldsymbol{w}(t) = \sum_{i=1}^{n} p_i(t-1)\boldsymbol{x}_i$. Set $\boldsymbol{w}(t) \leftarrow \boldsymbol{w}(t) / \parallel \boldsymbol{w}(t) \parallel_2$.
4) Convergence check:
   a) if $\boldsymbol{w}(t) \neq \boldsymbol{w}(t-1)$, go to Step 2.
   b) Else if there exists i such that $\boldsymbol{w}^T \boldsymbol{x}_i = 0$, set $\boldsymbol{w}(t) \leftarrow (\boldsymbol{w}(t) + \Delta\boldsymbol{w}) / \parallel \boldsymbol{w}(t) + \Delta\boldsymbol{w} \parallel_2$ and go to Step 2. Here $\Delta\boldsymbol{w}$ is a small nonzero random vector.
   c) Otherwise, set $\boldsymbol{w}^* = \boldsymbol{w}(t)$ and stop.

The projection vector $\boldsymbol{w}$ converges to $\boldsymbol{w}^*$, which is a local maximum point of $\sum_{i=1}^{n} \left| \boldsymbol{w}^T \boldsymbol{x}_i \right|$, and there is a possibility that it may not be global solution. The first principal base of L2-PCA is used as an initial vector $\boldsymbol{w}(0)$ and presents a greedy search algorithm for $m > 1$. The greedy search algorithm is as follows:

Greedy search algorithm

1) For all $i \in \{1, \ldots, n\}$, $\boldsymbol{x}_i \leftarrow \boldsymbol{x}_i - \bar{\boldsymbol{x}}$, where $\boldsymbol{x}_i$ is the image for study, and $\bar{\boldsymbol{x}} = \frac{1}{n} \sum_{i=1}^{n} \boldsymbol{x}_i$
2) Set a data-set $\boldsymbol{X} = (\boldsymbol{x}_1 \ldots \boldsymbol{x}_n)$.
3) Compute a autocorrelation matrix $\boldsymbol{S} = \boldsymbol{X}^T \boldsymbol{X}$.
4) Processing from 5) to 7) is repeated m times.
5) The first principal base of L2-PCA is assumed to be the initial value and a principal base $\boldsymbol{w}$ is calculated by the PCA-L1 algorithm.
6) Set $\boldsymbol{X} \leftarrow \boldsymbol{X} - \boldsymbol{w}(\boldsymbol{w}^T \boldsymbol{X})$. By the processing, $\boldsymbol{X}$ is projected into orthogonal complementary space of $\boldsymbol{w}$.
7) Compute $\boldsymbol{S} = \boldsymbol{X}^T \boldsymbol{X}$.

We should notice that the solution of the greedy algorithm is not necessarily the optimal solution of (2). However, it is expected to provide a set of good projections that maximizes L1 dispersion.

III. FAST ALGORITHM

The algorithms shown in the previous section require much execution time to calculate the new data set and the autocorrelation matrix as the procedure 6) and 7) in the greedy search algorithm. Then, this section describes a fast method of the processing. Now, we would like to focus explanation for the computation of the mth autocorrelation matrix. At this time, it is assumed that the $m-1$th data-set:

$$\begin{aligned} \boldsymbol{X} &= \begin{bmatrix} \boldsymbol{x}_1 & \boldsymbol{x}_2 & \ldots & \boldsymbol{x}_n \end{bmatrix} \\ &= \begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1n} \\ x_{21} & x_{22} & \cdots & x_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ x_{d1} & x_{d2} & \cdots & x_{dn} \end{bmatrix}, \end{aligned} \tag{4}$$

autocorrelation matrix:

$$\begin{aligned} \boldsymbol{S} &= \boldsymbol{X}^T \boldsymbol{X} \\ &= \begin{bmatrix} \sum_{k=1}^{d} x_{k1}^2 & \sum_{k=1}^{d} x_{k1}x_{k2} & \cdots & \sum_{k=1}^{d} x_{k1}x_{kn} \\ \sum_{k=1}^{d} x_{k2}x_{k1} & \sum_{k=1}^{d} x_{k2}^2 & \cdots & \sum_{k=1}^{d} x_{k2}x_{kn} \\ \vdots & \vdots & \ddots & \vdots \\ \sum_{k=1}^{d} x_{kn}x_{k1} & \sum_{k=1}^{d} x_{kn}x_{k2} & \cdots & \sum_{k=1}^{d} x_{kn}^2 \end{bmatrix}, \end{aligned} \tag{5}$$

and the principal base $\boldsymbol{w}$ are given. To calculate the mth principal base, first, the data-set $\boldsymbol{X}$ is projected into orthogonal complementary space of the vector $\boldsymbol{w}$. The new data-set $\boldsymbol{X}'$ is given by

$$\begin{aligned} \boldsymbol{X}' &= \begin{bmatrix} \boldsymbol{x}'_1 & \ldots & \boldsymbol{x}'_n \end{bmatrix} \\ &= \begin{bmatrix} \boldsymbol{x}_1 - (\boldsymbol{w}^T \boldsymbol{x}_1)\boldsymbol{w} & \ldots & \boldsymbol{x}_n - (\boldsymbol{w}^T \boldsymbol{x}_n)\boldsymbol{w} \end{bmatrix} \\ &= \begin{bmatrix} \boldsymbol{x}_1 - \alpha_1 \boldsymbol{w} & \ldots & \boldsymbol{x}_n - \alpha_n \boldsymbol{w} \end{bmatrix}, \end{aligned} \tag{6}$$

where α_i is a inner product between $\boldsymbol{w}$ and $\boldsymbol{x}_i$.
Second, a new autocorrelation matrix $\boldsymbol{S}'$ is calculated form $\boldsymbol{X}'$. $\boldsymbol{S}'$ is given by

$$\begin{aligned} \boldsymbol{S}' &= \boldsymbol{X}'^T \boldsymbol{X}' \\ &= \begin{bmatrix} \sum_{k=1}^{d} x'^2_{k1} & \sum_{k=1}^{d} x'_{k1}x'_{k2} & \cdots & \sum_{k=1}^{d} x'_{k1}x'_{kn} \\ \sum_{k=1}^{d} x'_{k2}x'_{k1} & \sum_{k=1}^{d} x'^2_{k2} & \cdots & \sum_{k=1}^{d} x'_{k2}x'_{kn} \\ \vdots & \vdots & \ddots & \vdots \\ \sum_{k=1}^{d} x'_{kn}x'_{k1} & \sum_{k=1}^{d} x'_{kn}x'_{k2} & \cdots & \sum_{k=1}^{d} x'^2_{kn} \end{bmatrix}. \end{aligned} \tag{7}$$

Based on (7), we can derive

$$s'_{ij} = \sum_{k=1}^{d} x'_{ki} x'_{kj} \tag{8}$$

$$= \sum_{k=1}^{d} (x_{ki} - \alpha_i w_k)(x_{kj} - \alpha_j w_k) \tag{9}$$

$$= \sum_{k=1}^{d} (x_{ki} x_{kj}) - 2\alpha_i \alpha_j + \sum_{k=1}^{d} \alpha_i \alpha_j w_k^2 \tag{10}$$

$$= s_{ij} + \alpha_i \alpha_j (\sum_{k=1}^{d} w_k^2 - 2), \tag{11}$$

where s'_{ij} is the (i, j) element in $\boldsymbol{S}'$. Therefore, to calculate $\boldsymbol{S}'$, we only have to add $\alpha_i \alpha_j (\sum_{k=1}^{d} w_k^2 - 2)$ to s_{ij}. This method achieved to decrease computational time almost to a quarter, and the obtained bases are identical to the ones of PCA-L1.

IV. GPGPU

Because the greedy search algorithm shown in the previous section is almost independent processing, a parallel processing seems to contribute to speed up the implementation. Accordingly, we focus on General-purpose computing on GPU (GPGPU). GPU has a lot of simple computing units; and consequently, GPU enables higher parallelization than CPU. GPGPU is expected to exceed CPU's performance in various problem-solving.

CUDA (Compute Unified Device Architecture) [5] is a free GPGPU programing tool provided by NVIDIA. It is based on C and C++, and we can hence make a GPGPU program easily. The source code consists of host codes and device codes which describe operations on CPU and GPU, respectively. The host codes set the number of threads and call kernel functions described in the device codes.

NVIDIA also develops a linear algebra library named CUBLAS, which is a BLAS (Basic Linear Algebra Library) optimized to CUDA.

In the implementation on GPU, each thread is assigned to each element of matrices or vectors.

V. Experimental results

To evaluate the efficacy of GPGPU, we adopt the fast PCA-L1 algorithm on CPU and GPU to face recognition. In the face recognition, face images are divided into two groups; for study and for test. The images are considered to be column vectors by the raster scan order, and study vectors are embedded in a data-set matrix $\boldsymbol{X}$. A low-dimensional linear subspace of $\boldsymbol{X}$ is calculated. In the identification of a person, the study image having the smallest distance from the test image in the subspace is considered to be the most suitable person.

A Yale Face Database [4] is used for the experiment. The database consists of 65 gray-scale images of 10 individuals. And the size of images is 170×120 pixels. Fig.1 shows a part of the images. Specifications of PC are shown as follows: OS: Windows XP, CPU: Intel Core 2 DUO(R) 2.33 GHz, Memory: 2GB, and GPU: Geforce GTX 280. Table I shows specifics the GPU.

Fig. 1. A part of images used to experiment

TABLE I
SPECIFICATION OF GPU

	GeForce GTX 280
Global memory size	1GB
A nomber of computing units	240
Shared memory size	16KB
Clock rate	1.30 GHz

In the experiment, we measure the processing time under the following conditions: the number of extracted features is fixed to 20, and the number of the images for study is varied from 50 to 100. For the purpose of comparison, we measure CPU's processing time on equal terms with GPGPU. Table II shows the experimental results. As shown in table II, GPGPU's computing time is faster than CPU's one. It seems that GPGPU is more effective as the number of study images increase. Furthermore, we obtain the same calculation result of extracted features, and we consequently get the same recognition rate.

Next, we compare the time required to finish each process of GPGPU to CPU. we fix the number of images for study to 100, and measure the computing time of following each process.

(1) Move data-set $\boldsymbol{X}$ from a main memory to a global memory on GPU.
(2) Set $\boldsymbol{x}_i \leftarrow \boldsymbol{x}_i - \bar{\boldsymbol{x}}$ for all $i \in \{1, \ldots, n\}$,.
(3) Compute an autocorrelation matrix $\boldsymbol{S} = \boldsymbol{X}^T \boldsymbol{X}$.
(4) Solve a eigenvalue problem to obtain a initial vector of PCA-L1 algorithm.
(5) Execute the PCA-L1 algorithm to get a principal bases.
(6) Compute a new data-set $\boldsymbol{X}'$ and a new autocorrelation matrix $\boldsymbol{S}'$

Table III shows the results. Table III tells that the execution time of any processing is shortened. In particular, the computing time of the autocorrelation matrix is shortened greatly since it is high degree of parallelization.

VI. Conclusion

In this paper, we conducted the fast PCA-L1 algorithm on CUDA. The effect was verified by applying the algorithm to the dimension reduction of Eigenface in the facial recognition. In the PCA-L1 algorithm, it is necessary to solve the

TABLE II
THE COMPUTING TIME OF GPGPU AND CPU WITH RESPECT TO THE NUMBER OF STUDY IMAGES

The number of images for study	Computing time [s]		rate [%]
	GPGPU	CPU	
50	0.97	1.67	42.05
60	1.02	2.09	51.48
70	1.27	2.78	54.48
80	1.30	3.19	59.22
90	1.33	3.80	65.00
100	1.55	4.59	66.32

TABLE III
THE COMPUTING TIME OF GPU AND CPU FOR EACH PROCESS

Process No.	Computing time [s]		rate [%]
	GPGPU	CPU	
(1)	0.0069	-	-
(2)	0.032	0.047	31.55
(3)	0.011	1.04	98.97
(4)	0.0040	0.016	75.51
(5)	0.012	0.074	83.93
(6)	0.0088	0.032	72.51

eigenvalue problems of the number of extracted bases, and it requires heavy computational cost. GPGPU achieves faster performance than CPU, in addition, it keeps recognition rate.

In future works, we would like to apply a larger dimensions data-set to GPGPU.

REFERENCES

[1] P.N.Belhumeur, J.P.Hespanha, and D.J.Kriegman, "Eigenfaces vs. fisherfaces: recognition using class specific Linear projection," IEEE Trans. on PAMI, vol.19, no.7, pp.711-720, 1997.

[2] Nojun Kwak, "Principal component analysis baced on L1-norm maximization," IEEE Trans. on PAMI, vol.30, no.9, pp.1672-1680, 2008.

[3] N.Funatsu, and Y,Kuroki. "Fast Method of Principal Component Analysis Based on L1-Norm Maximization Algorithm," Asia-Pacific Signal and Information Processing Association Annual Summit and Conference 2009.

[4] A.S.Georghiades, P.N.Belhumeur, and D.J.Kriegman, "From few to many: Illumination cone models for face recognition under variable lighting and pose," IEEE Transactions on Pattern Analysis and Machine Intelligence, vol.23, no.6, pp.643-660, 2001.

[5] NVIDIA, "CUDA Zone – The resource for CUDA developers," http://www.nvidia.co.jp/object/cuda_home_jp .html, accessed Sept. 15,2010.