PARTIAL LEAST SQUARES ON GRAPHICAL PROCESSOR FOR EFFICIENT PATTERN RECOGNITION

Balaji Vasan Srinivasan, William Robson Schwartz, Ramani Duraiswami, Larry Davis

Department of Computer Science, University of Maryland, College Park, MD, USA.
`[balajiv,schwartz,ramani,lsd]@umiacs.umd.edu`

ABSTRACT

Partial least squares (PLS) methods have recently been used for many pattern recognition problems in computer vision. Here, PLS is primarily used as a supervised dimensionality reduction tool to obtain effective feature combinations for better learning. However, application of PLS to large datasets is hindered by its higher computational cost. We propose an approach to accelerate the classical PLS algorithm on graphical processors to obtain the same performance at a reduced cost. Although, PLS modeling is practically an offline training process, accelerating it helps large scale modeling. The proposed acceleration is shown to perform well and it yields upto $\sim$ 30X speedup, It is applied on standard datasets in human detection and face recognition.

***Index Terms*—** partial least squares, supervised dimensionality reduction, graphical processors, human detection, face recognition

1. INTRODUCTION

With improved sensors, the amount of data available in many computer vision problems has increased dramatically in the recent years. Further recent research has led to the use of new feature descriptors that capture the data characteristics better. With the availability of tall fat datasets (large number (N) of very high dimensional (d) features) for learnng, algorithms using the data often scale poorly on large datasets. One way to mitigate this is to project the features to a low-dimensional subspace, and learn from data in this subspace, thus improving the algorithmic scaling. Partial least squares are one such dimensionality reduction approach that learn this subspace such that the classes are well separated. They have recently gained popularity as a supervised dimensionality technique in vision applications because of its effective performance.

Partial least squares [1] are a wide class of methods for modeling relations between sets of observed variables by means of latent variables. It includes regression, classification and dimensionality reduction. The underlying assumption in PLS is that the observed data is generated by a system/process which is driven by a small number of latent variables. Projection of the observed data to the latent structure was developed and enhanced by Wold and his collaborators [2, 3]. PLS has received a great amount of attention in the field of chemometrics [4], and its success there has led to its application in other scientific areas like bioinformatics and image processing [5, 6].

Although originally proposed as a regression technique, PLS can be used for class-aware dimensionality reduction [7]. PLS based dimensionality reduction techniques have recently been used in computer vision to effectively combine several low level features for enhancing detection and recognition. Typically in these applications the number of samples N is much less than the number of features d, ($N << d$). For example, Schwartz et al. [8] combine histogram of oriented gradients (HOG), color frequency and co-occurence features (tens of thousand of features) for ~ 1000 samples using PLS to a low dimensional (typically ~ 20) space and use a standard classifier in this subspace for efficient human detection. Inspite of the good performances of PLS, it has a computational cost of O(Nd) and does not scale well with N and d. Although in the vision algorithms, PLS modeling is an offline training step, accelerating it is paramount to enable large scale modeling.

In this paper, we accelerate PLS [9] on a graphical processor. The paper is organized as follows: in Section 2, we first introduce PLS formulation and draw its parallels with PCA and LDA. We then describe the NIPALS algorithm for PLS and its computation bottlenecks. We introduce the GPU-based strategies for PLS-acceleration in Section 3 and illustrate the resulting performance improvement in human detection and face recognition in Section 4.

2. PARTIAL LEAST SQUARES

PLS [1] is a latent variable based modeling technique that builds relationships between X, a matrix of features and Y, a matrix of the corresponding response variable. While a detailed analysis of PLS can be found in [1], we provide a brief introduction here.

Let $x \in R^d$ denote the d-dimensional feature space (independent variable) and $y \in R^f$ be the corresponding response variable (could be the 1-dimensional class-label or f dimensional dependent variables). Given the independent and dependent variable pairs $\{x_i, y_i\}, i = 1, \ldots, N$ ($x \in R^d, y \in R^f$), PLS aims at the modeling the relationship between x and y by using latent structures. Denoting X as a $N \times d$ matrix of independent data points x_i and Y as a $N \times f$ matrix of y_j, PLS decomposes them as below,

$$X = TP^T + F, \quad (1)$$
$$Y = UQ^T + G, \quad (2)$$

where T and U are $N \times p$ ($p < d$) with p latent vectors, P ($d \times p$) and Q ($f \times p$) are loading vectors and F ($N \times d$) and G ($N \times f$) are residual vectors. PLS, in its classical form, is based on the *nonlinear iterative partial least squares (NIPALS) algorithm*, which constructs a set of weight vectors w such that,

$$\max[cov(t_i, u_i)]^2 = \max_w [cov(Xw, y)]^2. \quad (3)$$

where t_i and u_i are the i^{th} columns of $\mathbf{T}$ and $\mathbf{U}$ respectively. cov indicates the covariance defined by,

$$cov(t, u) = E\left[(t - E[t])(u - E[u])^T\right], \quad (4)$$

Table 1. *NIPALS algorithm*

```
Nonlinear Iterative PArtial Least Squares (NIPALS)

Given: $N \times d$ feature samples $X$ and
       $N \times f$ response variable $Y$

 1) Initialize vector $u$,
      if $Y$ is $1-$dimensional, assign $u = Y$,
      else $u =$ a random $N \times 1$ vector
 2) Iterate to convergence:
    $a)$ $w = X^T u/(u^T u)$
    $b)$ $\|w\| \rightarrow 1$
    $c)$ $t = Xw$
    $d)$ $c = Y^T t/(t^T t)$
    $e)$ $\|c\| \rightarrow 1$
    $f)$ $u = Yc$
  3) $p = X^T t$
  4) Deflate $X : X \leftarrow X - tp^T$ and $Y : Y \leftarrow Y - tc^T$
  If more projection vectors are required, go to step 2
```

E being the expectation.

The NIPALS algorithm is shown in Table. 1 and comprises two main steps; in the first step, the weight w is evaluated according to Eq. 3. Once w is obtained, NIPALS performs a deflation of the X and Y matrices, which is a rank-1 update such that any information captured by w is removed from X and Y. If the desired latent space dimension is not achieved, the algorithm returns to the first step to evaluate a new w.

It can be shown that the weight w in NIPALS corresponds to the first eigenvector of the following eigenvalue problem and the NIPALS is just a mirror of the popular power iterations for finding the dominant eigenvectors,

$$[X^T yy^T X]w = \lambda w. \tag{5}$$

Because the rank of the above system is limited by the number of samples N, $N < d$ yields a few dominant eigenvectors and hence PLS works best in this scenario.

The NIPALS algorithm (Table. 1) involves a number of linear algebra operations on the feature matrix X and response variable Y. The asymptotic space and time complexity is O(dN) (assuming $f \leq d$). It is not possible to do away with O(dN) space requirement because this is required to store the feature matrix. However, the time complexity can be addressed with efficient parallelization strategy using graphical processors.

2.1. PLS, LDA and PCA

Principal component analysis (PCA) and Linear Discriminant Analysis (LDA) are two widely used dimensionality reduction techniques in machine learning. We compare PLS with PCA and LDA based on their optimization criteria.

PCA projects the features onto a direction of maximal variance (the principal directions),

$$max_{|w|=1}[var(Xw)]. \tag{6}$$

This is equivalent to solving the following eigen problem,

$$[X^T X]w = \lambda w. \tag{7}$$

PCA does not use any information about the response variables and hence is an unsupervised dimensionality reduction.

LDA finds projection w that maximizes the interclass variance while minimizing the intraclass variance in the projected space. If y spans c classes, LDA results in a $c-1$ dimensional subspace. Unlike PCA, LDA uses the response variable information to determine the projection space. Let

$$\mathbf{S}_b = \sum_{c=1}^{C} (\mu^c - \mu)^T (\mu^c - \mu), \tag{8}$$

denote the *between-class variance* where μ^c is the mean of the feature vectors from class c and μ is the mean of the feature vectors from all classes. Let

$$\mathbf{S}_w = \sum_{c=1}^{C} \sum_{i=1}^{n_c} (x_c^j - \mu^c)^T (x_c^j - \mu^c) \tag{9}$$

denote the *within-class* variance, where x_c^j is the the j^{th} feature sample in class c. LDA solves the following eigen problem,

$$[S_w^{-1} S_b]w = \lambda w. \tag{10}$$

LDA is preferred over PCA for classification tasks because of its supervised nature that enables a unique projection for each classification task. However, the size of the LDA projection space is bounded by $c-1$, which can limit the projected space's representation ability. When handling a few samples of thousands of features ($N < d$), the variance matrices in LDA are not of full rank and the weight vectors cannot be extracted.

PLS is also a supervised dimensionality reduction technique, as seen from Eq. 3. It has further been shown that for normalized response variables (y), the eigen problem of Eq. 5 reduces [10] to,

$$[S_b]w = \lambda w, \tag{11}$$

which is equivalent to maximizing the between class-variance. Rosipal et al. [10] further note that for a 2-class problem, the first eigenvector of Eq. 11 (PLS) and Eq. 10 (LDA) are identical. However, PLS does not have the $c-1$ limit in the projection space dimension, and can extract further projecting directions beyond $c-1$. Also, unlike LDA, PLS is well-suited for data with $N << d$.

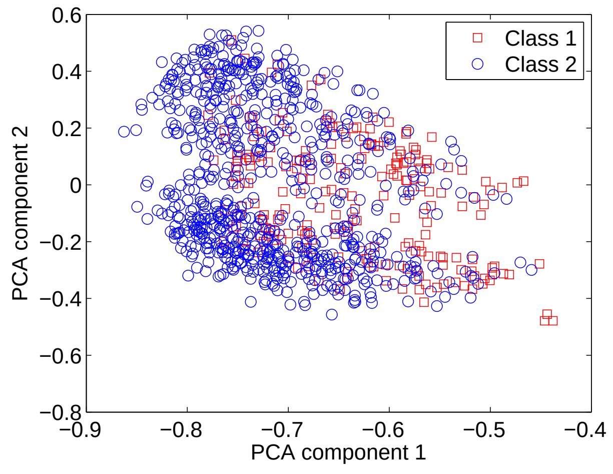
We compared the projections of LDA, PLS and PCA for classifying two random objects in the Caltech-101 database [11] based on the SIFT features. The resulting projections that were learned from the entire samples in the class are shown in Fig. 1. It can be seen that while the projections of PCA are not discriminating for the first 2 components, PLS and LDA exhibit more discrimination between classes.

In order to illustrate the performance for $N << d$, we chose only 25 samples in the two classes, and the resulting projections for PCA and PLS are shown in Fig. 2. The LDA in this case failed because of a rank-deficit variance matrix. PLS is again more discriminating than PCA.

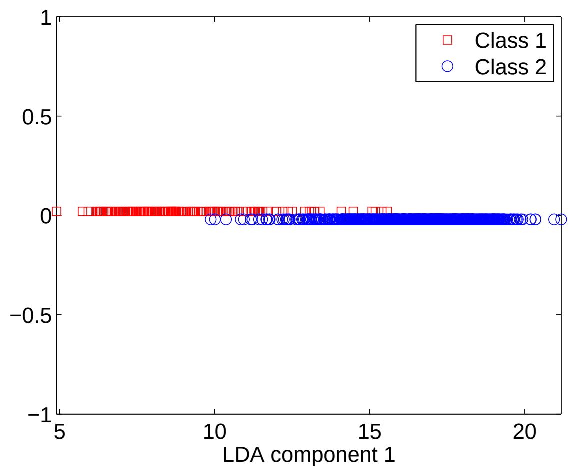
From these examples, the PLS and LDA spaces are clearly more discriminating than PCA-space. Further, when the number of features exceed the number of samples, PLS projections work the best among the 3, making it a better choice than LDA/PCA for $d >> N$. it may be a problem when you have to built multiple models, like for face recognition.

3. GRAPHICAL PROCESSORS

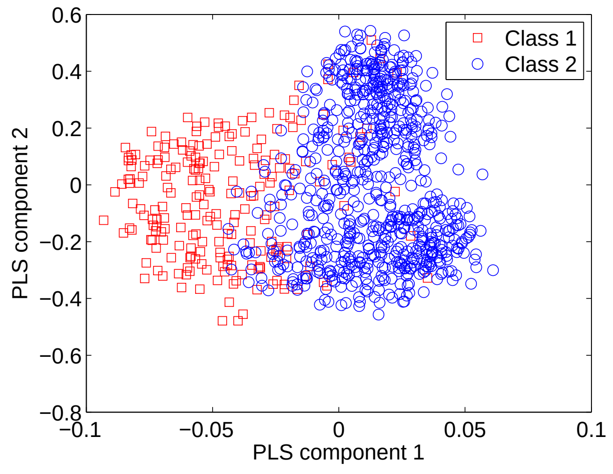
Computer chip-makers are no longer able to easily improve the speed of processors, with the result that computer architectures of the future will have more cores, rather than more capable faster cores. This



(a) First 2 PCA dimensions

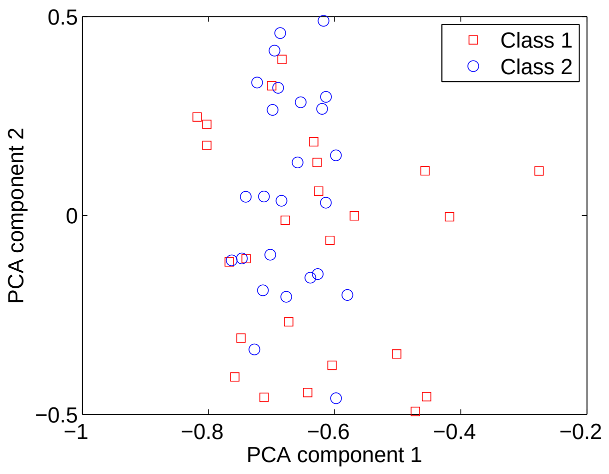


(b) LDA projection

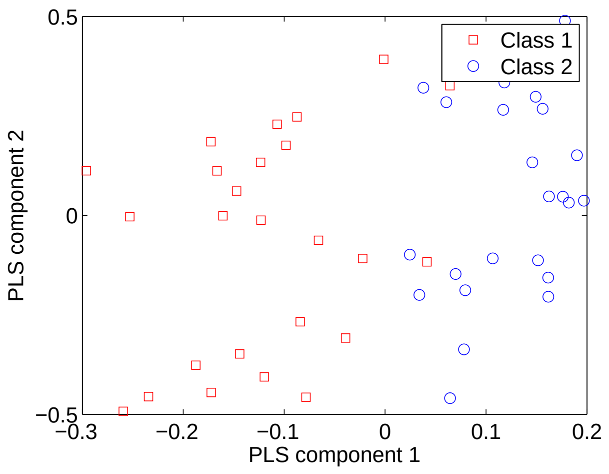


(c) First 2 PLS dimensions

Fig. 1. *PCA, LDA and PLS projections on* 2 *randomly chosen objects in Catltech*-101 *[11] dataset*



(a) First 2 PCA dimensions with 25 samples per class



(b) First 2 PLS dimensions with 25 samples per class

Fig. 2. *First two dimensions of PCA and PLS projections using only* 25 *samples per class*

era of multicore computing requires that algorithms be adapted to the data parallel architecture. A particularly capable set of data parallel processors are the graphical processors, which have evolved into highly capable compute coprocessors. A graphical processing unit (GPU) is a highly parallel, multi-threaded, multi-core processor with tremendous computational horsepower. In 2008, while the fastest Intel CPU could achieve only $\sim$ 50 Gflops speed theoretically, GPUs could achieve $\sim$ 950 Gflops on actual benchmarks [12]. Fig. 3 shows the relative growth in the speeds of NVIDIA GPUs and Intel CPUs as of 2008 (similar numbers are reported for AMD/ATI CPUs and GPUs). The recently announced FERMI architecture significantly improves these benchmarks. Moreover, GPUs power utilization per flop is an order of magnitude better. GPUs are particularly well-suited for data parallel computation and are designed as a single-program-multiple-data (SPMD) architecture with very high arithmetic intensity (ratio of arithmetic operation to memory operations). However, the GPU does not have the functionalities of a CPU like task-scheduling. Therefore, it can efficiently be used to assist the CPU in its operation rather than replace it.

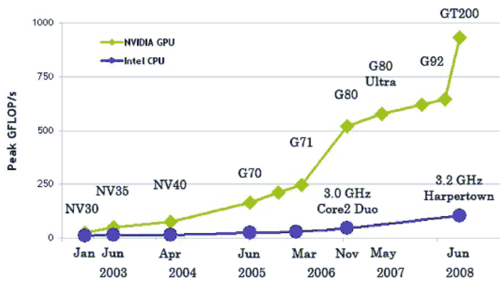


Fig. 3. *Growth in the CPU and GPU speeds over the last* 6 *years on benchmarks (Image from [12])*

In 2007, NVIDIA introduced *Compute Unified Device Architecture (CUDA)*[12], a parallel programming model that leverages the parallel compute engine in NVIDIA GPUs to solve general purpose computational problems. With CUDA, GPUs can be seen as a bunch of parallel co-processor that can assist the main processor in its computations. The OpenCL initiative seeks to provide a similar non-proprietary API for general purpose GPU computing. NVIDIA have further released CUBLAS [9], a CUDA based BLAS (Basic Linear Algebra Subprograms) which includes accelerated BLAS1, BLAS2 and BLAS3 routines in single and double precisions, although double precision on GPU is an order of magniturde slower than single precision.

3.1. GPU for accelerating NIPALS

CUBLAS is an implementation of BLAS (Basic Linear Algebra Subprograms) on top of the NVIDIA CUDA driver. The library is self-contained at the API level, that is, no direct interaction with the CUDA driver is necessary. CUBLAS attaches to a single GPU and does not auto-parallelize across multiple GPUs. The basic model by which applications use the CUBLAS library is to create matrix and vector objects in GPU memory space, fill them with data, call a sequence of CUBLAS functions, and, finally, upload the results from GPU memory space back to the host. To accomplish this, CUBLAS provides helper functions for creating and destroying objects in GPU space, and for writing data to and retrieving data from these objects. CUBLAS offers best speedup for BLAS2 (matrix-vector operations) and BLAS3 (matrix-matrix operations) operations.

The key difference between an efficient algorithm on a sequential processor and a graphics processor is that the former requires to have as less computation as possible while the latter needs to minimize memory access to and from the global memory. An efficient GPU algorithm should ensure a minimal transfer of data from the host memory to the GPU memory. NIPALS has several BLAS1, BLAS2 and BLAS3 tasks. Therefore, the best computational performance would result if BLAS2 and BLAS3 are performed on GPU and BLAS1 on CPU. But, this would result in several data movements back and forth between CPU and GPU, and can cost heavily in access times. Therefore, in our GPU-based NIPALS we perform all blas operations on the GPU. Such a strategy would be advantageous because the BLAS2 and BLAS3 speedups are significant and the savings on the memory transfer times is big enough to weigh over BLAS1 disadvantages. Accordingly the schematic for the GPU based implementation is shown in Table 2.

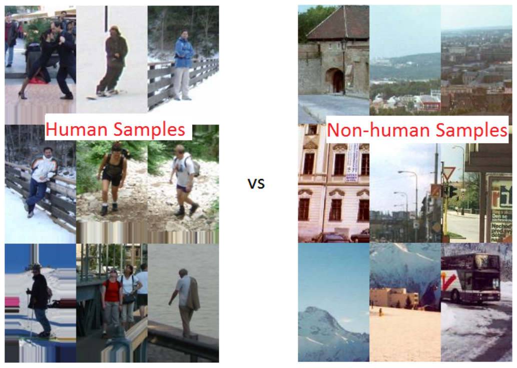


Fig. 4. *Human samples vs Nonhuman samples from INRIA Person database [13]*

4. EXPERIMENTS

We experimented on PLS based human detetion and face recognition to illustrate the GPU speedup and its utility. In all our experiments, we used a Quad core Intel Xeon processor for the CPU operations and the 240-core Tesla C1060 for the GPU operations. The CPU and GPU codes were written in C++ with Matlab linkages and the mean absolute error between the CPU and GPU based PLS-modeling in all runs was $\sim 10^{-6}$.

4.1. Human Detection

Detection of humans in videos and images is a task of fundamental importance in computer vision in order to be able to provide information for higher level processing steps. It has been observed that combination of feature descriptors provide improvements on detection rates [8]. Such feature augmentation leads to a very high dimensional feature space that may not be handled correctly by some machine learning methods. Schwartz et al. [8] propose the use of PLS to reduce the dimensionality of the feature space.

The method proposed by Schwartz et al. [8] works as follows. First, a PLS model is learned to discriminate between human and non-human samples (Fig. 4), with the resulting low dimensional representation being used to estimate the parameters of a quadratic classifier. Once the model is available, a testing sample has its feature descriptors projected onto the model and the low dimensional representation is used to classify it as either human or non-human. The method was tested on several human detection datasets and it was able to improve detection results comparing to other state-of-art methods.

In this experiment, we emulate the experiments in [8] by extracting only the Histogram of Oriented Gradients (HOG), instead of the full feature set done in [8], from the INRIA person database [13] and building the PLS model for human/non-human training datasets. There were 2416 positive training samples and 9744 negative training examples. We extract 98928 HOG features from each of these samples, thus resulting in a tall-fat dataset which will be our test bed for performance analysis.

First, we illustrate the effect of the number of PLS components and the computational time. Fig. 5(a). For low number of PLS components, the eigenvalue problem is well-conditioned and there-

Table 2. *NIPAL algorithm with CUBLAS*

```
GPU based NIPALS for partial least squares

Given: $N \times d$ Feature samples $X$ and response variable $Y$

 1) Allocate GPU memory for $X$ and $y$ and transfer data to GPU [cublasAlloc,cublasSetVector]
 2) Iterate to convergence:
    $a)$ $w = X^T u/(u^T u)$; $\|w\| \rightarrow 1$: [cublasSgemv,cublasSscal,cublasSnrm2]
    $c)$ $t = Xw$: [cublasSgemv]
    $d)$ $c = Y^T t/(t^T t)$; $\|c\| \rightarrow 1$: [cublasSgemv,cublasSscal,cublasSnrm2]
    $f)$ $u = Yc$: [cublasSgemv]
 3) $p = X^T t$: [cublasSgemv]
 4) Deflate $X : X \leftarrow X - tp^T$ and $Y : Y \leftarrow Y - tc^T$: [cublasSgemv]
 If more projection vectors are required, go to step 2
```

fore the speedup obtained is not significant. However, as the number of PLS components is increased, the eigen system becomes ill-conditioned, resulting in increased computations and thus significant speedup. In a practical experiment, the number of PLS components is determined by cross-validation after building a PLS model with at least 10 to 20 components.

Next, we built the PLS model using the entire 98928 features, but chose a random subset of training samples. The corresponding computational performance is shown in Fig. 5(b). Initially, the data transfer time is dominant, hence GPU-based NIPALS is slower than the CPU-based NIPALS. However, as the sample size increases, there is an improved performance.

In the final experiment, we built the PLS model using a randomly chosen subset of features to study the effect of dimension in speedup. The resulting performance is shown in Fig. 5(c), and a speedup of $\sim$ 30X is obtained against the direct version.

Fig. 6 shows the speedup of our GPU-based NIPALS for various sample and feature sizes. Although there is considerable speedup for lower datasize/dimension, significant speedup is obtained for large datasize/dimension indicating its utility for large datasets.

Schwartz et al. [8] perform a cross-validation analysis to select a subset of the negative training examples. The PLS model is built initially using 5000 samples and this is used to classify the remaining negative examples. The misclassified examples are now added to the training set, and the experiment is repeated for a fixed number of trials. We repeated this experiment using the HOG features and observed that while the CPU based approach takes over an hour, GPU-based approach took only $10 - minutes$ for a 5-iteration trial. Here, we have reported the time taken for PLS modeling and cross validation only, and have not included the feature extraction time. This shows that using a GPU-based approach would reduce the training time by $6 - 7$ folds, which can be utilized for further algorithmic sophistication. Figure 7 shows the detection error trade-offs obtained using 1126 positive testing samples and by shifting the detection windows by 8 pixels in the negative testing images, all of which are available in the dataset, similar to the approach in [8].

4.2. Face Recognition

The problem of face recognition has received significant attention over the years due to its importance to applications such as surveillance. One of the tasks of face recognition is called identification, in which for a given probe face, its goal is to match unknown faces (probe samples) against a gallery of known people. One of recent approaches that deal with face identification using PLS is proposed in Schwartz et al. [14]. It uses a one-against-all classification scheme (Fig. 8). This scheme requires one PLS model per subject; therefore, due to the availability of large amounts of data, it is important that these models are built quickly.

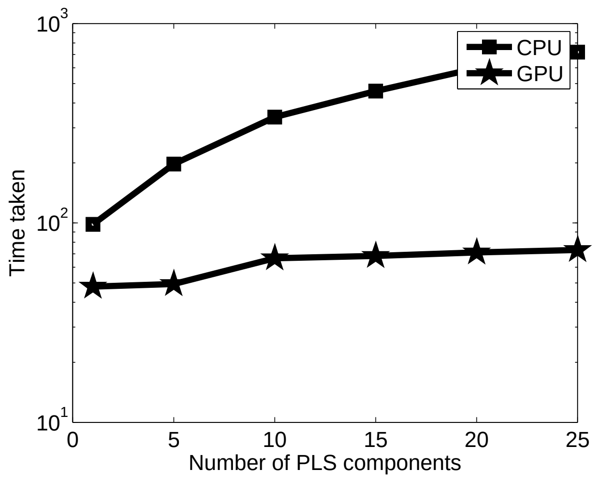
Specifically, the one-against-all classification scheme works as follows. When the PLS model is built for the i^{th} subject, the samples of the remaining subjects in the gallery are used as counter-examples. Therefore, PLS method estimates which features are better suitable to discriminate between the i^{th} subject and the remaining ones. Finally, after all models are built, when a probe sample is presented, it is project onto each model and the best match is the one with highest regression response.

In this experiment we compare the computational cost between CPU and GPU-based approaches for building the models for subjects in the gallery of the FERET dataset [15]. This dataset consists of 1196 subjects in the gallery (with one sample per subject), and four probe sets: fb (1995 images taken with different facial expressions), fc (194 images taken under different lighting conditions), dup1 (722 images taken at a later date), and dup2 (234 images taken at least one year apart). The recognition rates obtained by Schwartz et al. [14] are shown in Fig. 9.

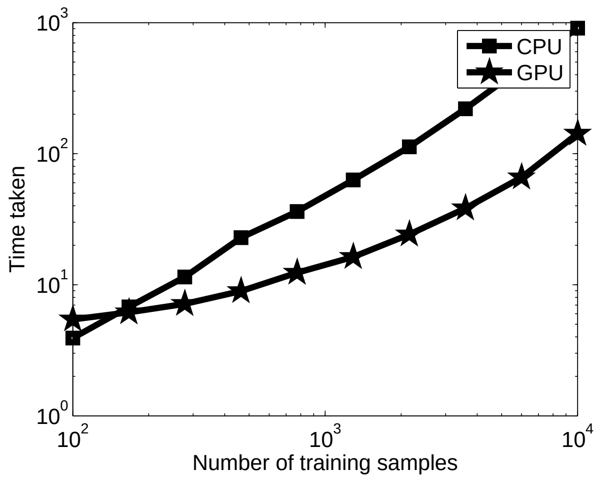
For each subject, GPU-based approach takes only 2.516 ± 0.068s to extract 20 PLS factors, whereas the CPU approach takes 17.06 ± 0.802s. Total time to process the 1196 subjects in the database was only 50 minutes with the GPU against a $\sim$ 6-hour CPU processing. Here again, we have reported the PLS modeling time only, without considering the feature extraction time. The improved computational performance allows for easier addition of new face models and addition of negative examples for improving an existing model.

5. CONCLUSION

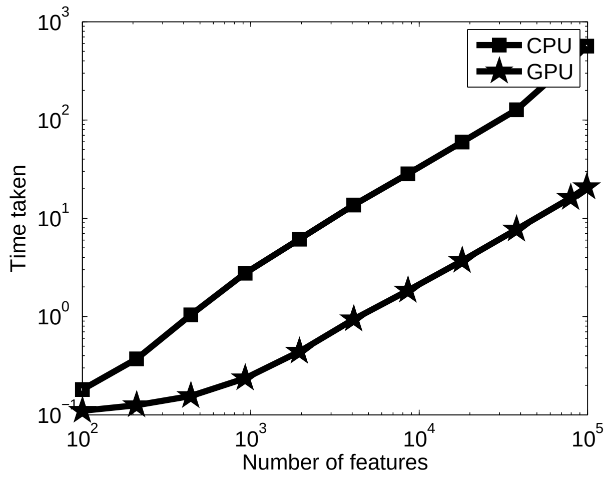
In this paper, we have accelerated partial least squares based dimensionality reduction on a graphical processor. Although, in many cases this is an offline task, we have illustrated the utility of the acceleration for tall and fat datasets to enable faster modeling and the speedups are promising. With newer FERMI architecture improving the FLOPS (FLoating-point OPerations per Second) further, the speedups can only improve and become increasingly advantageous. The core algorithm illustrated here will soon be released as an open source.



(a) Across varying PLS components



(b) Across varying sample sizes



(c) Across varying feature dimensions

Fig. 5. *Performance of GPU-based NIPALS under various conditions*

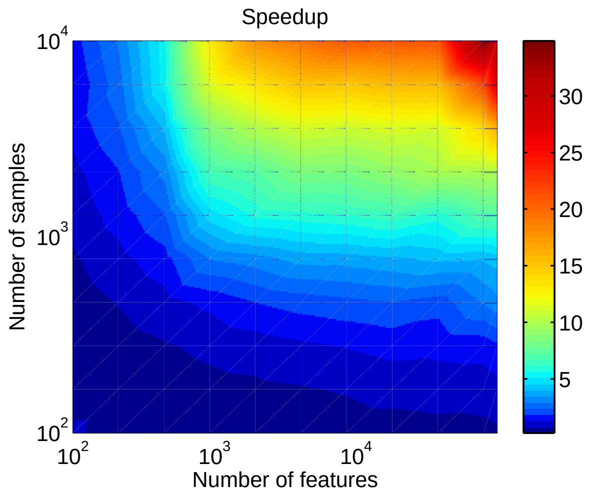


Fig. 6. *Speedup obtained by the GPU-based NIPALS for various sample sizes of different feature dimensions*

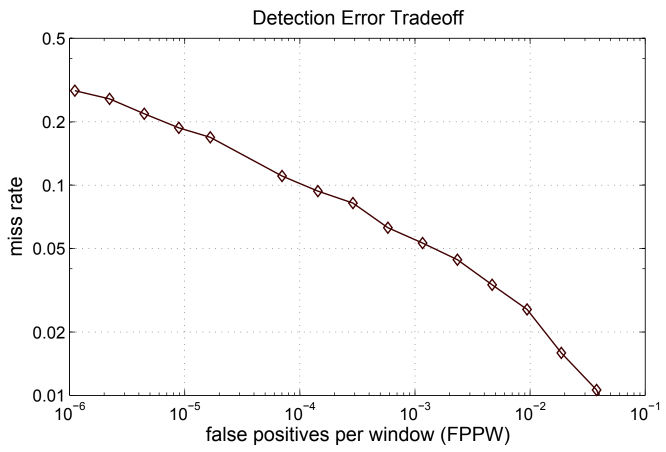


Fig. 7. *Detection error tradeoff for HOG features*

6. REFERENCES

[1] R. Rosipal and N. Krmer, "Overview and recent advances in partial least squares," in *in Subspace, Latent Structure and Feature Selection Techniques, Lecture Notes in Computer Science*. 2006, pp. 34–51, Springer.

[2] H. Wold, "Soft modeling: the basic design and some extensions," *Systems under indirect observation*, vol. 2, pp. 589–591, 1982.

[3] S. Wold, A. Ruhe, H. Wold, and W. J. Dunn, "The collinearity problem in linear regression. the partial least squares (PLS) approach to generalized inverses," *SIAM Journal on Scientific and Statistical Computing*, vol. 5, no. 3, pp. 735–743, 1984.

[4] S. Wold, M. Sjöström, and L. Eriksson, "PLS-regression: a basic tool of chemometrics," *Chemometrics and Intelligent Laboratory Systems*, vol. 58, no. 2, pp. 109–130, October 2001.

[5] D.V. Nguyen, "Tumor classification by partial least squares using microarray gene expression data," *Bioinformatics*, vol. 18, pp. 39–50(12), January 2002.

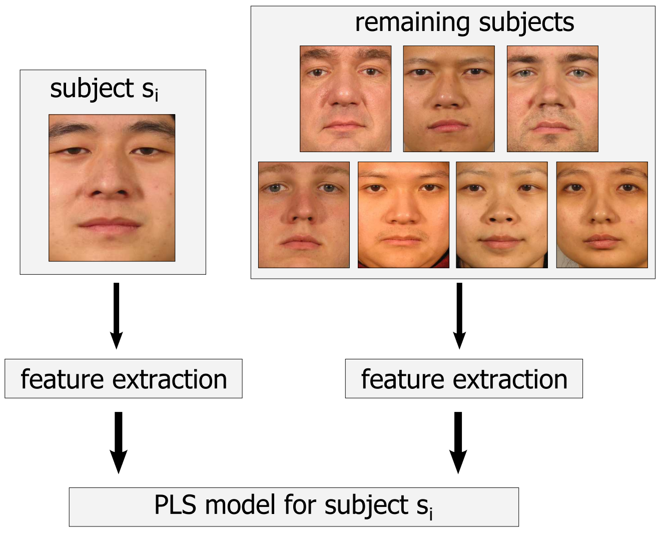


Fig. 8. *One against all face recognition scheme used in [14] for the construction of PLS model*

[6] K.J. Worsley, "An overview and some new developments in the statistical analysis of PET and fMRI data," *Human Brain Mapping*, vol. 5, pp. 254–258, 1997.

[7] R. Teofilo, J. Martins, and M. Ferreira, "Sorting variables by using informative vectors as a strategy for feature selection in multivariate regression," in *Journal of Chemometrics*, vol. 23, pp. 32–48.

[8] W.R. Schwartz, A. Kembhavi, D. Harwood, and L.S. Davis, "Human detection using partial least squares analysis," in *International Conference on Computer Vision*, 2009.

[9] NVIDIA, *NVIDIA CUDA CUBLAS Library 2.1*, 2008.

[10] R. Rosipal and L.J. Trejo, "Kernel PLS-SVC for linear and nonlinear classification," in *In Proceedings of the twentieth International Conference on Machine Learning (ICML-2003*, 2003, pp. 640–647.

[11] L. Fei-Fei, R. Fergus, and P. Perona, "Learning generative visual models from few training examples: An incremental Bayesian approach tested on 101 object categories," 2004, p. 178.

[12] NVIDIA, *NVIDIA CUDA Programming Guide 2.0*, 2008.

[13] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," in *International Conference on Computer Vision & Pattern Recognition*, June 2005, vol. 2, pp. 886–893.

[14] W.R. Schwartz, H. Guo, and L.S. Davis, "A robust and scalable approach to face identification," in *European Conference on Computer Vision*, 2010.

[15] J.P. Phillips, H Moon, S.A. Rizvi, and P.J. Rauss, "The FERET evaluation methodology for face-recognition algorithms," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 22, no. 10, pp. 1090–1104, 2000.

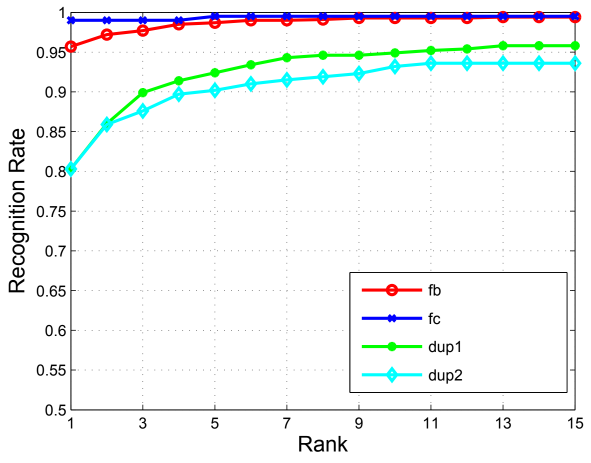


Fig. 9. *Recognition rates based on Schwartz et al. [14] on FERET database [15]*