Karhünen-Loève Transform: An Exercise in Simple Image-Processing Parallel Pipelines

Martin Fleury, Andy C. Downton and Adrian F. Clark

Department of Electronic Systems Engineering, University of Essex, Wivenhoe Park,
Colchester, CO4 4SQ, U.K
tel: +44 - 1206 - 872795
fax: +44 - 1206 - 872900
e-mail fleum@essex.ac.uk

Abstract. Practical parallelizations of multi-phased low-level image-processing algorithms may require working in batch mode. The features of a common processing model, employing a pipeline of processor farms, are described. A simple exemplar, the Karhünen-Loève transform, is prototyped on a network of processors running a real-time operating system. The design trade-offs for this and similar algorithms are indicated, when a general solution is sought. Eventual implementation on large- and fine- grained hardware is considered. The chosen exemplar is shown to have some features, such as strict sequencing and unbalanced processing phases, which militate against a comfortable parallelization.

1 Introduction

Many low-level image-processing (IP) algorithms, such as spatial filters, are completely localized in their data references. If adjacent image data are overlapped at boundaries then at a small additional cost a data-farming programming paradigm can be employed, in which the only communication is between worker process and data farmer. Using separability and/or linearity, it is also possible to decompose other algorithms, such as orthogonal transforms, rather than employ a global access pattern. If these latter algorithms are viewed as single-image library functions then a difficulty commonly arises because it is necessary to centralize between the data-farming phases. However, since IP is often in batch mode, it only requires a slight shift in perspective towards continuous data flows in order to realise that effective parallelizations may occur if a pipeline is the normal form of processing.

In this paper, this basic concept is applied to a Karhünen-Loève transform (KLT), which is generally engaged in batch mode. As a development environment, we have used a network of microprocessors running the VxWorks real-time operating system [1]. Compared to other distributed environments, the VxWorks-based system is attractive for algorithm prototyping: because it is an isolated environment, because the thread structure is not superimposed on top of heavy-weight processes, and because event response times are optimized. Since the VxWorks system is not the final target parallel system, we are interested in the computational complexity rather than the performance on VxWorks.

2 The VxWorks Real-Time Kernel

VxWorks is intended as a Unix-like single-user operating system (O.S.) for real-time development work. The KLT program modules were written in 'C' and cross-compiled on a PC running the NextStep O.S. Once compiled, the modules are loaded and linked on a set of 68030 boards, each hosted by a PC. The 68030 processors are connected by an Ethernet LAN segment. For inter-processor communication, VxWorks includes a source-compatible BSD socket API.[1] Each program module consists of one or more threads, each of which can be spawned as required. Remote spawning is accomplished by providing an iterative server, which responds to requests over the LAN from the central farmer module.

Priority-based scheduling of threads is employed on worker modules, giving priority to communication-handling threads. Messages are stored in circular buffers, with access by the communication threads regulated through semaphores.[2] A `message queue` primitive is available in VxWorks which allows the application thread to `rendezvous` with any communication thread. On the farmer module, the `select` socket call is employed in a way that de-multiplexes work requests in a fair manner. A number of features enable the software structure to be reused, e.g. tagged messages, the selection of a streamed communication mode and a strict interface to application functions.

3 The Karhünen-Loève Transform

The KLT [2] is most widely used in applications such as multi-spectral analysis of satellite-gathered images [3] through the resulting spectral signature of imaged regions. Significant data reductions are also achieved in the storage of satellite images if the multi-spectral set are transformed to KLT space. The KLT has recently been applied to the recognition of facial images [4]. Notice that the size of the image set is potentially much larger in the latter two applications.

Consider a sample set of real-valued images from an ensemble of images. Create vectors with the equivalent pixel taken from each of the images, i.e. if there are D images each of size $M \times N$ then form the column vectors $\mathbf{x_k} = (x^0_{ij}, x^1_{ij}, \ldots, x^{D-1}_{ij})^T$ for $k = 0, 1, \ldots, MN-1$, $i = 0, 1, \ldots, M-1$ and $j = 0, 1, \ldots, N-1$ (with superscript T representing the transpose). Calculate the sample mean vector: $\mathbf{m_x} = \frac{1}{M}\sum_{k=0}^{MN-1} \mathbf{x_k}$. Use a computational formula to create the sample covariance matrix: $[\mathbf{C_x}] = \frac{1}{M}\left(\sum_{k=0}^{MN-1} \mathbf{x_k x_k^T}\right) - \mathbf{m_x m_x^T}$. Form the eigenvector set: $[\mathbf{C_x}]\mathbf{u_k} = \lambda_k \mathbf{u_k}$, $k = 0, 1, \ldots D-1$, where $\{\mathbf{u_k}\}$ are the eigenvectors with associated eigenvalue set $\{\lambda_k\}$. The KLT kernel is a unitary matrix, $[\mathbf{V}]$, whose columns, $\{\mathbf{u_k}\}$ (arranged in descending order of eigenvalue amplitude), are used to transform each zero-meaned vector: $\mathbf{y_k} = [\mathbf{V}]^T(\mathbf{x_k} - \mathbf{m_x})$.

[1] The VxWorks system is available in a tightly-coupled variant, by means of processors linked by a VME bus, but again sockets form the principal communication mode.

[2] The 68030 has two compare-and-swap instructions as well as support for cache management.

4 KLT Parallelization

The time complexity of the operations is analysed as follows, where no distinction is made between a multiplication and an add operation: form the mean vector with $O(MND)$ element-wise operations; calculate the set of outer products and sum, $\sum_{k=0}^{MN-1}\mathbf{x_k}\mathbf{x_k^T}$, in $O(MND^2)$ time; form $\mathbf{m_x}\mathbf{m_x^T}$; subtract matrices to find $[\mathbf{C_x}]$; and find the eigenvectors of $[\mathbf{C_x}]$ (the eigenvector calculation is $O(D^3)$); convert the $\{\mathbf{x_k}\}$ to zero-mean form in $O(MND)$; and form the $\{\mathbf{y_k}\}$ by $O(MND^2)$ operations. Since the covariance matrix is generally too small to justify parallelization, the total parallelizable complexity is $O(MND+MND^2)$, i.e. the eigenvectors are found sequentially.

Consider the KLT as applied to a single image in one-off mode. One parallelization method would be to send a cross-section through the images to each process, selecting the cross-section on the basis of image strips. In a first phase, the mean vector of each cross-section image strip is found and returned to a central farmer along with a partial vector sum, forming the strip matrix: $[\mathbf{T_i}] = \frac{1}{MN}\sum_{k=0}^{(MN-1)/n}\mathbf{x_k^i}(\mathbf{x_k^i})^T$, $i = 1, 2, \ldots n$, for n strips. In a second phase, the farmer can find $[\mathbf{V}]$ from $[\mathbf{C_x}]$, which is now broadcast so that for each strip the calculation of $\{\mathbf{y_k^i}\}$ can go ahead. However, the duplication of sub-image distribution (once for the partial sums and once to compute the transform) is inefficient even though the duplication is strictly necessary if demand-based data-farming is employed.

A possibility is to retain the data that are farmed out in the first phase at the worker processes. On a system with store-and-forward communication the first farming phase will have established an efficient distribution of the workload given the characteristics of the network. Therefore, the second phase will already have approximately the correct workload distribution. This is not a solution on a shared network of workstations as processor load and network load is time dependent. The solution is also not a general one since other two-phased low-level IP algorithms do not usually use the same data in both phases, though the time complexity can be similar. The method of finding a workload distribution by a demand-based method and then re-using the distribution for a static load-balance in subsequent independent runs may have general potential.

An alternative static load-balancing scheme is to exchange partial results amongst the worker processes so that the calculation of matrix $\mathbf{V}$ can be replicated on each worker process. A suitable exchange algorithm for large-grained machines is available if the processors can be organized in a uni-ring. A second method of performing a KLT is to consider each image as a single vector formed by stacking rows. In [5], this scheme, whatever its merits for particular applications, is shown to have the same time complexity but to be less flexible in regard to load-balancing.

5 Pipeline Decompositions

A simple pipeline can be formed by the sequence: covariance, eigenmatrix and image-transform modules. In a preliminary implementation of this pipeline on the VxWorks-based system, both farmers were placed on the same processor (Fig. 1) since the same data are needed for forming the covariance matrix and for transforming to eigenspace.[3] In principle, double buffering of image sets allows loading of one image set to proceed while the previous image set is transformed. However, for a target VLSI implementation this implies a total buffer size of 5 Mbytes and upwards would be needed for (say) 10 images of 512 × 512 size.

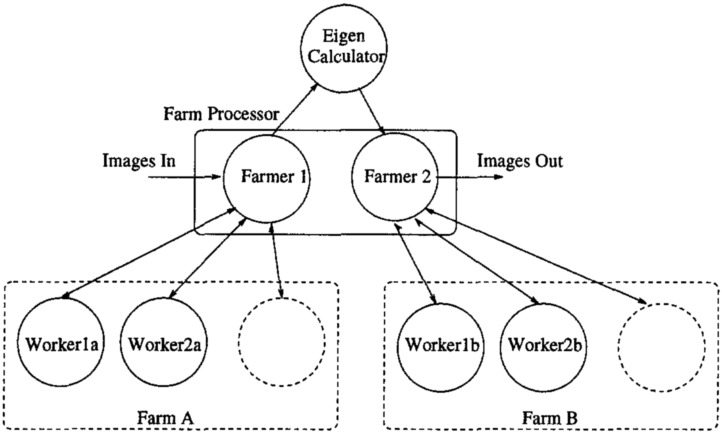

Fig. 1. KLT Pipeline Partitioning

The first pipeline stage can be further partitioned into calculation of the mean vector and calculation of the outer products, since the two calculations are independent and therefore can take place in parallel. Additionally, the second stage calculations can be split further between reducing the image set to zero-mean form and transforming the image set, though these calculations are not independent. However, the reduction to zero-mean form is independent of the eigenvector calculation and could take place in parallel with that calculation. These partitioning possibilities are shown in Fig. 2. Assume that the two farms in the first pipeline partition can be operated in parallel, by means of two farmers on the same processor feeding from a common buffer. Since the maximum time complexity of each stage of the new pipeline is reduced from $O(MND + MND^2)$ to $O(MND^2)$, then the number of processors on any one farm that will reduce pipeline throughput is reduced. However, the bandwidth requirements are increased. Since both the components of the second partition are dependent on the completion of all the calculations on the first partition, the pipeline traversal latency will not be reduced by decomposing the image into smaller components.

The pipeline of Fig. 2 is relevant as the basis of a VLSI scheme, possibly through a systolic array. For a large-grained parallelization, the arrangement of Fig. 1 but merging the eigenvector calculation into the work of the second farmer is practical. The scheduling regime on the processor hosting the two farmers is round-robin for fairness. Since the time complexity of both stages of the pipeline

[3] A worker module can also be placed on the same processor to soak up any spare processing capacity.

is the same it is now easily possible to scale the throughput in an incremental fashion.

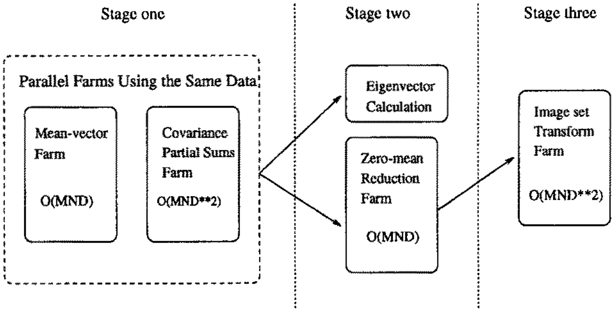

Fig. 2. Alternative Partitioning of the KLT Pipeline

6 Conclusion

The Karhünen-Loève Transform has been prototyped on a pipeline of parallel processor farms. A two-phased single farm arrangement is described, in one mode of which the initial workload distribution, arrived at by a demand-based method, is reused in a static load-balance. Two different pipeline decompositions are explored. To achieve a completely balanced pipeline for all but the largest of jobs will be prohibitive in practical terms. The simpler of the two pipelines is appropriate for large-grained applications, whereas a further decomposition may be relevant to fine-grained VLSI implementations. The strict sequencing in the KLT algorithm prevents attempts to improve the pipeline traversal latency.

Acknowledgement

This work is being carried out under EPSRC research contract GR/K40277 ‘Portable software tools for embedded signal processing applications’ as part of the EPSRC Portable Software Tools for Parallel Architectures directed programme.

References

1. Wind River Systems, Inc., 1010, Atlantic Avenue, Almeda, CA. *VxWorks Programmer's Guide*, 1993. Version 5.1.
2. J. J. Gerbrands. On the relationship between SVD, KLT and PCA. *Pattern Recognition*, 14(1-6):375–381, 1981.
3. P. J. Ready and P. A. Wintz. Information extraction, SNR improvement and data compression in multispectral imagery. *IEEE Trans. on Comms.*, 31(10):1123–1130, 1973.
4. M. Kirby and L. Sirovich. Application of the Karhunen-Loève procedure for the characterization of human faces. *IEEE Trans. PAMI*, 12(1):103–108, 1990.
5. M. Fleury. *Efficient Parallel Image-Processing Software.* PhD thesis, University of Essex, 1996.