Karhunen-Loève Transform: An Exercise in Simple Image-Processing Parallel Pipelines[*]

M. Fleury, A. C. Downton and A. F. Clark

Department of Electronic Systems Engineering

University of Essex, Wivenhoe Park, Colchester, CO4 3SQ, U.K.

`tel: +44 - 1206 - 872712`

`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 new processing model, employing a pipeline of processor farms, are described. A simple exemplar, the Karhunen-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. In the manner of co-design, 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 implementation.

Keywords: Karhunen-Loève transform, parallel pipeline, multi-spectral images, co-design

[*]A shortened version of this paper appeared in the Third International Euro-Par Conference, Passau, Germany, August 1997

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. Provided that $n \gg p$, where n is the number of sub-images farmed out – for instance an image row, and p is the number of processors, then load-balancing can be made semi-automatic by having returning processed work form an implicit request for more data. This method will work even when there is no *a priori* knowledge of the sub-image workload distribution. Using separability and/or linearity, it is also possible to decompose other algorithms, such as orthogonal transforms [1], 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 [2], 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 [3], the advantages of the pipeline for practical computation *vis-à-vis* the hypercube were already formulated in relation to IP, the authors advocating a re-partitionable pipeline.

By laying emphasis on the data-farm one does in effect use the same tuned program for as many low-level IP algorithms as is possible, analogously to the way commercial databases present the same structure and support software across a variety of hardware. Needless to say, the data-farm paradigm is a deadlock-free design methodology [4] and moreover each farm within a pipeline preserves locality of reference, which in [5] is shown to be vital for effective use of low-dimensional direct networks.

After partition of a large sequential program written in a structured manner each stage of the pipeline can be parallelized by means of a data-farm. The program code length might be *circa* 10,000 lines in length and unfamiliar to those charged with the parallelization. In [6], there is a new system design and development methodology for decomposing multi-algorithm applications into pipelines of processor farms, i.e. parallel pipelines, which has been successfully employed [7, 8, 9, 10, 11] in real-time embedded applications.[1] However, the same idea works for single algorithms and moreover the idea, and its limitations are illustrated in a simple form when directed to single algorithms.

In this paper, the basic concept is applied to a Karhunen-Loève transform (KLT), which is generally engaged in batch mode, for example to process multi-spectral satellite imagery. As a parallel processor, we have used a network of microprocessors running the VxWorks real-time operating system [12]. Though the VxWorks-based system is mainly intended for direct application of real-time systems such as in robotics, we have also found it has a potential for algorithm prototyping as a parallel pipeline can be mapped to the system. VxWorks system overhead is low: it is single-user; the thread structure is not superimposed on top of heavy-weight processes; and event response times are optimized. Algorithm prototyping is a first step in co-design, whereby a suitable hardware/software mix for a target embedded systems is decided upon [13].

In the paper, various early suggestions for target implementation of the KLT are made in order to reach real-time performance. Only the computational model is considered. The precise number of VLSI components making up a parallel pipeline is not determined. The TI C80 or MVP [14], in effect a four worker data-farm on one integrated circuit (IC), as a VLSI solution matches the computational model, except that only the 'farmer' processor on the IC has a floating-point unit. Systolic arrays could be substituted as the computational elements for two of the component data farms, alleviating a problem on general-purpose machines of insufficient memory bandwidth.

The VxWorks system is thread-based with rapid context switching. We indicate other thread-based environments which could also serve as a means of prototyping parallel pipelines. By exploiting the commonality between the environments it is possible to specify an abstract machine which has wide applicability. Notice that similar ideas in which the Communicating

[1]These embedded systems can be characterised as soft and data-dominated. Hard real-time systems have safety-critical constraints and are generally control-dominated.

Sequential Processes (CSP) [15] framework is enlarged are also current in hard real-time systems [16]. The transputer, which is now of course an elderly microprocessor but does implement the CSP framework, has not been superceded in terms of provision for context switching though many DSP cores are superior in computational terms.

The prototype parallel pipeline was transferred to a distributed-memory parallel computer, the Transtech Paramid [17], in order to make performance tests in an isolated environment. The Paramid is an i860-based machine with transputers acting as communication coprocessors to the i860s.

Section 2 introduces in a practical way the VxWorks operating system and compares it in passing to two other thread-based systems. Section 3 reviews various Karhunen-Loève computational algorithms. In Section 4, the algorithm used for the parallel implementation is detailed, including computational complexity. Possible implementations suitable for a library of parallel routines are also indicated in Section 4. Different pipelined arrangements for working in batch mode are the subject of Section 5. Some techniques transferable from the relatively simple KLT algorithm to multi-algorithm applications are additionally mentioned in Section 5. Section 6 details performance results from tests on the Paramid. Finally, Section 7 summarizes and draws some conclusions.

2 The VxWorks Real-Time Kernel

VxWorks is a Unix-like single-user operating system (O.S.) for real-time development work. The KLT program modules were written in 'C', cross-compiled on a PC running the NextStep O.S. (based on the Mach micro-kernel) and loaded and linked on attached 68030K boards.[2] The 68030 microprocessor [18], has an instruction set with test-and-set and compare-and-swap, suitable for memory access control. The 68030K boards are linked by an Ethernet LAN, and VxWorks provides a source-compatible BSD 4.3 socket API for using the network. In Fig. 1, two alternative configurations for the present VxWorks system are shown. Each

[2]Versions of VxWorks are available for a range of more recent microprocessors.

program module consists of one or more tasks, which can be spawned as required. Remote spawning was accomplished by writing an iterative server.

VxWorks is representative of an approaching consensus in facilities for parallel processing shared for example by transputer-based machines using Inmos parallel 'C' [19] and by a SPARC-based workstation network with the light-weight process (LWP) library, that we have currently ported our pipelines to. The Java programming language, which also has elementary support for multi-threading represents a future implementation vehicle. The facilities consist of:

- support for low-overhead context switching, though switching is as yet based on control or communication blockage and not on cache conflict;
- a local inter-thread communication mechanism, which can be extended to the external network; and
- an access-control mechanism to regulate write contention to static memory. Counting semaphores, which are low-overhead, are constructed for access-control using existing control primitives.

With these facilities we find that we can construct a processing model [20] which amounts to a relaxed version of the widely-disseminated CSP model.

The main augmentations to the CSP model are: since low-level IP is data-intensive, shared memory is required to avoid excessive memory-to-memory transfers; and local buffers are provided to guarantee that communication latency is masked given sufficient computation granularity. We also employ a non-deterministic operator, CSP's alternation, for the explicit purpose of efficient de-multiplexing. For example, in VxWorks alternation was simulated by the socket API `select` call, whereas alternation is microcoded on the transputer. Notice that this use of alternation does not prevent limited compiler rescheduling of communication calls including message aggregation [21].

In both the Sun workstation and the VxWorks implementations, CSP's channel can be effectively simulated by socket message-transmitting calls.[3] In fact, we specialise this concept by always including a message tag. This allows intervening buffering software to be transparent to the form of the succeeding message and therefore the software is reusable. By using streamed communication for tag and message, there is no set-up overhead for tag messages (or multiple components of one message). On occasions where disparate elements form one message, the socket vectored mode of communication will underlie the channel call. However, again our aim is to provide a single communication primitive whatever the underlying mechanism.

In our LWP library implementation, the LWP `rendezvous` call could directly model the CSP channel when confined to intra-processor communication. Likewise, in VxWorks a `message queue` primitive is available which when single-spaced fulfils the same purpose. A software wrapper is provided to make it appear that the channel is used for intra- and inter-processor communication.

The features of the common model can be captured in a high-level template. The data-farm template which we have implemented across socket-based platforms is shown in Fig. 2. Scheduler threads, which provide round-robin scheduling on a priority-based system, were not required on the VxWorks implementation. Optional I/O handling threads in the farmer can be included though system buffering may already occur.

3 The Karhunen-Loève Transform

The nomenclature of the Karhunen-Loève Transform (KLT) [22, 23] is confused [24]. In statistics, the KLT is reserved for a transform that acts on any data set, while the term Principal Components Algorithm (PCA) is reserved for zero-meaned data. However, in this paper the term KLT refers to a transform acting on zero-meaned image data.

The KLT differs from other common orthogonal transform algorithms, such as the Fourier

[3]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. Notice that the 68030 supports bus-control signals.

transform, in two respects:

- it is data-dependent; and
- it is applied to an image ensemble.

In statistics, the columns of the matrix to be transformed represent realisations of a stochastic process. Therefore, it is legitimate to employ the PCA to reduce the dimensionality of the data. In image processing, each image can be viewed as a single realisation of a stochastic process. Therefore, the transform should act on a sample set of images, from a possibly infinite population of images.

The KLT has a number of features [25] which occur by virtue of the rotation of the data representation basis vectors (Fig. 3). Amongst the features relevant to the computation of a KLT are:

- The KLT transform achieves optimal data compression in the mean-square error sense.[4]
- The KLT projects the data onto a basis that results in complete decorrelation, though only if the data are first zero-meaned. Notice that the decorrelation is of statistical significance and does not correspond necessarily to a semantic decomposition.
- If the data are of high dimensionality, by reason of properties one and two it is possible to reduce the dimensionality.
- For some finite stationary Markov order-one processes with known boundary conditions — many natural scenes acquired by an appropriate sensor — the basis vectors are *a priori* harmonic sinusoidals and hence a fast algorithm (the FFT-like sine transform) is available [26]. Another route to fast implementation is by neural nets employing Hebbian learning [27].

[4]That is $\epsilon(k) = E[(x-\hat{x})^T(x-\hat{x})]$ is a minimum, where $\hat{x}$ is the representation of x truncated to k terms. E is the mathematical expectation operator. The minimal orthonormal basis set is found by the method of Lagrangian multipliers, using the orthonormality of the basis vectors as the constraint.

However, the lack of a general fast algorithm, because the covariance matrix eigenvectors must be found in every case, makes it pressing to find a suitable parallel decomposition.

The KLT is employed in multi-spectral analysis of satellite-gathered images [28] through the spectral signature of imaged regions. Significant data reductions are achieved in the storage of satellite images if the multi-spectral set are transformed to KLT space. The dimensionality in this application is relatively low.

The KLT has also been applied to sets of face images [29]. A candidate face, once normalized and transformed, can be matched by a suitable distance metric (e.g. Mahanalobis) to a database of faces stored in KLT space. A reformulation of the KLT algorithm is utilized for the face recognition application, whereby the rows of each face image are stacked to form one vector per-image. In [30], a way of reducing the computational complexity of the reformulation is demonstrated. In fact, the reformulation apparently is equivalent to the algorithm developed in Section 4. Unfortunately, the alternative KLT algorithm is not as clearly parallelizable as the algorithm of Section 4 because of the long vectors required.

The face database and other databases are usually of high dimensionality. In this case, an iterative solution may be necessary [31]. The iterative solution relies on keeping the state of the KLT space which does not suit a data-farming programming paradigm.

It is apparent from Fig. 3 that the SNR will be improved by a KLT if additive Gaussian noise is present, resulting from incoherent sensors, as in multi-spectral scanning. There is a variant of the KLT [32] suitable for coping with multiplicative noise such as speckle noise in multifrequency synthetic aperture radar. Finally, noise-dominated image sets may be analysed through the low-component images.

4 KLT Parallelization

Consider a sample set of real-valued images from an ensemble of images. For example, these might be the same scene at different wavelengths or a collection of related images at the same wavelength. 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 $\vec{x}_k = (x_{ij}^0, x_{ij}^1, \ldots, x_{ij}^{D-1})$ for $k = 0, 1, \ldots, MN-1$, $i = 0, 1, \ldots, M-1$ and $j = 0, 1, \ldots, N-1$. Calculate the sample mean vector:

$$\vec{m}_x = \frac{1}{MN} \sum_{k=0}^{MN-1} \vec{x}_k \ . \tag{1}$$

Use a computational formula to create the sample covariance matrix:

$$[C_x] = \frac{1}{MN} \left(\sum_{k=0}^{MN-1} \vec{x}_k \vec{x}_k^T \right) - \vec{m}_x \vec{m}_x^T \ , \tag{2}$$

with superscript T representing the transpose. Equation (2) is appropriate if the image ensemble is formed by a stochastic process that is wide-sense stationary in time. Form the eigenvector set:

$$[C_x]\vec{u}_k = \lambda_k \vec{u}_k, \ k = 0, 1, \ldots D-1 \ , \tag{3}$$

where $\{\vec{u}_k\}$ are the eigenvectors with associated eigenvalue set $\{\lambda_k\}$. $[C_x]$ is symmetric and non-negative definite, which implies that the $\{\vec{u}_k\}$ exist and are orthogonal. In fact, the eigenvectors are orthonormal and therefore form a well-behaved coordinate basis. The associated eigenvalues are nonnegative. In any expansion of a data set onto the axis, the eigenvalues index the variance of the data set about the associated eigenvector.[5]

The KLT kernel is a unitary matrix, $[V]$, whose columns, $\{\vec{u_k}\}$ (arranged in descending order of eigenvalue amplitude), are used to transform each zero-meaned vector:

$$\vec{y}_k = [V]^T (\vec{x}_k - \vec{m}_x) \ k = 0, 1, \ldots, MN-1. \tag{4}$$

The properties of $[V]$ can serve as a check on the correct working of the algorithm.

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.

[5]This property arises from $\epsilon(k) = \sum_{j=k+1}^{\infty} \lambda_j$.

- Calculate the set of outer products and sum, $\sum_{k=0}^{MN-1} \vec{x}_k \vec{x}_k^T$, in $O(MND^2)$ time.
- Form $\vec{m}_x \vec{m}_x^T$; subtract matrices to find $[C_x]$; and find the eigenvectors of $[C_x]$. The eigenvector calculation is $O(D^3)$.
- Convert the $\{\vec{x}_k\}$ to zero-mean form in $O(MND)$.
- Form the $\{\vec{y}_k\}$ by $O(MND^2)$ operations.

Since the covariance matrix is, for the chosen multi-spectral application, too small to justify parallelization, the total parallelizable complexity is

$$O(MND) + O(MND^2) \tag{5}$$

Consider the KLT as applied to a single image in one-off mode. One way to parallelize the steps leading to (4) would be to send a cross-section through the images to each process, selecting the cross-section on the basis of image strips. The geometry is shown in Fig. 4. 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:

$$[T_i] = \frac{1}{MN} \sum_{k=0}^{(MN-1)/n} \vec{x}_k^i (\vec{x}_k^i)^T, \ i = 1, 2, \dots n \ , \tag{6}$$

for n strips. In a second phase, the farmer can find $[\vec{V}]$ from $[\vec{C_x}]$, which is now broadcast so that for each strip the calculation of $\{\vec{y}_k^i\}$ can go ahead.
However, the duplication of sub-image distribution (once for the partial sums and once to compute the transform) is inefficient.

A possibility is to retain the data that are farmed out in the first phase at the worker processes. On a transputer-based 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 wider potential.

An alternative static load-balancing scheme is to exchange partial results amongst the worker processes so that the calculation of matrix $[V]$ can be replicated on each worker process. A suitable exchange algorithm for large-grained machines can be adapted from [33], if the processors can be organized in a uni-ring.

5 A Pipeline Decomposition

The design trade-offs between pipeline throughput, latency and algorithm decomposition for practical parallelizations are explored in [6] and this section makes use of the conceptual framework. If a continuous-flow pipeline for KLTing image sets were to use temporal parallelism, by simply sending an image-set to each worker processor, then the communication and/or buffering overhead would be large given that it would not be easy to overlap communication with computation. An idealized, pipeline timing sequence of a decomposed KLT is given in Fig. 5. The covariance and transform threads are in fact the data-farmers which in theory should use sufficient worker tasks to balance the time required to find the eigenvectors of the covariance matrix. In a preliminary implementation of this elementary pipeline on the VxWorks-based system, both farmers were placed on the same processor (Fig. 6) since the same data are needed for forming the covariance matrix and for transforming to eigenspace.[6]

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 VLSI implementation this implies a total buffer size of 5 Mbytes and upwards would be needed for (say) 10 images of 512×512 size. Additionally, if one makes a best-case estimation of the desired number of processors on the two farms to achieve a balance, based on (5), the number of images in a set and the number of processors is impossibly large (Fig. 7). For a VLSI implementation, the calculation

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

of the eigenvectors cannot be implemented by simple circuitry as can the farmed stages and is likely to reduce this scaling in a favourable direction. A suitable VLSI processor may combine a RISC core for the eigenvector calculation with an array for regular calculations. Such a processor, with array and RISC on the same die, is implemented in [34]. The array can work in systolic or SIMD mode.

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. 8. 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 $O(MND) + O(MND^2)$ 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. 8 is relevant as the basis of a VLSI scheme, possibly through a systolic array. For a large-grained parallelization, the arrangement of Fig. 6 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 is the same it is now easily possible to scale the throughput in an incremental fashion.

The pipeline makes for a clearly defined development sequence, going from a sequential version to a two-stage implementation (when the transform is omitted or performed sequentially), to the final three stage implementation. Note that a somewhat more efficient purely sequential version occurs if the image set are zero-meaned before forming the covariance matrix, rather than separately reduce the covariance matrix to zero-mean form as occurs in the parallel version.

A form of parallel accounting is possible if the messages are instrumented by means of logical clock event-stamps [35]. The trace is 'double-entry' since the correct messages are matched by examining the message lengths. In Fig. 9, showing a space-time display snapshot from the `ParaGraph` visualizer [36], one worker has been used in each farm, with farmer one as processor 0, farmer 2 as processor 4, worker farm one as processor 1, worker farm two as processor 2 and eigenvector processor 3. The message event-trace records the end of the first image set processing, the hand-over of covariance and mean parameters, the commencement of processing by the first farm on the second image and the second farm's processing of the first image. Clearly, no time gap information is given but this is perhaps mostly relevant to the target system [37].

6 Results

The Paramid multicomputer employed has eight processing modules each with two little-endian processors, an i860 and a transputer, communicating through overlapped memory [19]. The transputer has link valency four, each link sustaining a raw 20Mb/s. When used in this manner the i860 supports a single process. The i860, run at 50MHz, is a superscalar design with internal Harvard architecture, and four-way vector units [38] resulting in a theoretical peak 200Mflop/s, though sustained performance can be a small fraction of that figure, 25.6Mflop/s using a standard in-cache benchmark [39]. The Portland optimising compiler was utilised to take advantage of the i860s features. Timings record the arithmetic mean from five image sets passing through the pipeline, with clock resolution ~ 0.001s. The i860s have 16Mbytes RAM which meant that a set of ten square images, size 256 single precision floating point pixels, would fit into main memory, and the equivalent set size 512 would not.

Table 1 records timings for two pipelines. To discount I/O times, the same image set, loaded into main memory, was reused. The Paramid normally loads images via a SCSI link which would create an I/O bottleneck. Local buffers to store three image lines were placed at the workers. In the first pipeline, each farmer occupied its own processor, two workers were employed in each farm, and the eigencalculator was also placed on a separate i860. In order to increase the size of the farms to three workers, the eigenvector calculations were switched to the transputer associated with the first farmer. However, the second pipeline showed an appreciable drop in performance. Equivalent times were recorded (not shown in the table) when the eigenvector calculations were shifted to the second farmer's transputer. The difficulty of improving the throughput illustrates the need to consider special-purpose hardware.

7 Conclusion

The Karhunen-Loève Transform (KLT) has been prototyped on a pipeline of parallel processor farms. The KLT is an exemplar of a generalised approach. In the exemplar, various algorithm analysis techniques are shown in simplified form. The pipeline employs reusable and instrumented data-farm software modules. The design exploits a commonality between recent parallel environments. The initial implementation of the KLT is upon a real-time Unix-like operating system kernel, VxWorks. 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 was carried out under EPSRC research contract GR/K40277 'Parallel software tools for embedded signal-processing applications' as part of the EPSRC Portable Software Tools for Parallel Architectures directed programme.

References

[1] M. Fleury and A. F. Clark. Parallelizing a set of 2D frequency transforms in a flexible manner. *IEE Proceedings Part I (Vision, Image and Signal Processing)*, 145(1):65–72, February 1997.

[2] E. R. Davies. Image processing—its milieu, its nature, and constraints on the design of special architectures for its implementation. In M. J. B. Duff, editor, *Computing Structures for Image Processing*, pages 57–76. Academic Press, London, 1983.

[3] M. H. Sunwoo and J. K. Aggarwal. Flexibly coupled multiprocessors for image processing. In F. A. Briggs, editor, *International Conference on Parallel Processing*, volume 1, pages 452–461. Pennsylvannia State University, 1988.

[4] P. H. Welch, G. R. R. Justo, and C. Willcock. High-level paradigms for deadlock-free high-performance systems. In *Transputer Applications and Systems '93*, pages 981–1004. IOS, Amsterdam, 1993.

[5] A. Agarwal. Limits on interconnection network performance. *IEEE Transactions on Parallel and Distributed Systems*, 2(4):398–412, October 1991.

[6] A. C. Downton, R. W. S. Tregidgo, and A. Çuhadar. Top-down structured parallelisation of embedded image-processing applications. *IEE Proceedings, Part I (Vision, Image and Signal Processing)*, 141(6):431–437, December 1994.

[7] A. Çuhadar and A. C. Downton. Structured parallel design for embedded vision systems: An application case study. In *Proceedings of IPA'95 IEE International Conference on Image Processing and Its Applications*, pages 712–716, July 1995. IEE Conference Publication No. 410.

[8] A. C. Downton. Speed-up trend analysis for H.261 and model-based image coding algorithms using a parallel-pipeline model. *Signal Processing: Image Communications*, 7:489–502, 1995.

[9] H. P. Sava, M. Fleury, A. C. Downton, and A. F. Clark. A case study in pipeline processor farming: Parallelising the H.263 encoder. In *UK Parallel '96*, pages 196–205. Springer, London, 1996.

[10] A. Çuhadar, D. Sampson, and A. Downton. A scalable parallel approach to vector quantization. *Real-Time Imaging*, 2:241–247, 1996.

[11] M. Fleury, A. C. Downton, and A. F. Clark. Pipelined parallelization of face recognition. *Machine Vision and Applications*, 1997. Submitted for publication.

[12] Wind River Systems, Inc., 1010, Atlantic Avenue, Almeda, CA. *VxWorks Programmer's Guide*, 1993. Version 5.1.

[13] M. Edwards and J. Forrest. A practical hardware architecture to support software acceleration. *Microprocessors and Microsystems*, 20:167–174, 1996.

[14] K. Balmer, N. Ing-Simmons, P. Moyse, I. Robertson, J. Keay, M. Hammes, E. Oakland, R. Simpson, G. Barr, and D. Roskell. A single chip multimedia video processor. In *IEEE Custom Integrated Circuits Conference*, pages 91–94, 1994.

[15] C. A. R. Hoare. *Communicating Sequential Processes.* Prentice-Hall, Englewood Cliffs, NJ, 1989.

[16] E. Verhulst. Non-sequential processing: Bridging the semantic gap left by the von Neumann architecture. In *IEEE Workshop on Signal Processing Systems*, pages 35–49, 1997.

[17] Transtech Parallel Systems Ltd., 17-19 Manor Court Yard, Hughenden Ave., High Wycombe, Bucks., UK. *The Paramid User's Guide*, 1993.

[18] D. Tabak. *Multiprocessors.* Prentice-Hall, Englewood Cliffs, NJ, 1990.

[19] M. Fleury, H. P. Sava, A. C. Downton, and A. F. Clark. Designing and instrumenting a software template for embedded parallel systems. In *UK PARALLEL '96*, pages 163–180. Springer, London, 1996.

[20] M. Fleury, H. Sava, A. C. Downton, and A. F. Clark. A real-time parallel image-processing model. In $6^{\underline{th}}$ *International Conference on Image Processing and its Applications*, volume 1, pages 174–178, 1997. IEE Conference Publication No. 443.

[21] G. Barrett. Rescheduling communications. In J. R. Davy and P. M. Dew, editors, *Abstract Machine Models for Highly Parallel Computers*, pages 281–294. O.U.P., Oxford, UK, 1995.

[22] K. Karhunen. Ueber lineare methoden in der wahrscheinlichtskeitsrechnung. *Annals Acad. Sci. FennicæSeries A.I*, 37, 1947.

[23] M. M. Loève. Fonctions aleatories de seconde ordre. In P. Levy, editor, *Process Stochastiques et Movement Brownien.* Hermann, Paris, 1948.

[24] J. J. Gerbrands. On the relationship between SVD, KLT and PCA. *Pattern Recognition*, 14(1-6):375–381, 1981.

[25] P. A. Devijver and J. Kittler. *Pattern Recognition: A Statistical Approach.* Prentice-Hall, London, 1982.

[26] A. K. Jain. A fast Karhunen-Loeve transform for a class of random processes. *IEEE Transactions on Communications*, 24:1023–1029, September 1976.

[27] E. Oja. Principal components, minor components, and linear neural networks. *Neural Networks*, 5:927–935, 1992.

[28] 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.

[29] 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.

[30] H. Murakami and B. V. K. V. Kumar. Efficient calculation of primary images from a set of images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 4(5):511–515, September 1982.

[31] P. J. Vermeulen and D. P. Casasent. Karhunen-Loève techniques for optimal processing of time-sequential imagery. *Optical Engineering*, 30(4):415–423, April 1991.

[32] J-S. Lee and K. Hoppel. Principal components transformation of multifrequency polarimetric SAR imagery. *IEEE Transactions on Geoscience and Remote Sensing*, 30(4):686–696, 1992.

[33] L. Sousa, J. Burrios, A. Costa, and M. Picdate. Parallel image processing for transputer-based systems. In *IEE International Conference on Image Processing and its Applications*, pages 33–36, 1992.

[34] R. B. Yates, N. A. Thacker, S. J. Evans, S. N. Walker, and P. A. Ivey. An array processor for general purpose digital image compression. *IEEE Transactions of Solid-State Circuits*, 30(3):244–249, March 1995.

[35] M. Raynal and M. Singhal. Capturing causality in distributed systems. *IEEE Computer*, pages 49–56, 1996.

[36] M. T. Heath and J. A. Etheridge. Visualizing the performance of parallel programs. *IEEE Software*, 8(5):29–39, 1991.

[37] M. Fleury, A. C. Downton, A. F. Clark, and H. P. Sava. The design of a clock synchronization sub-system for parallel embedded systems. *IEE Proceedings Part I (Computers and Digital Techniques)*, 144(2):65–73, March 1997.

[38] M. Atkins. Performance and the i860 microprocessor. *IEEE Micro*, pages 24–78, October 1991.

[39] N. Sarvan, R. Durrant, M. Fleury, A. C. Downton, and A. F. Clark. Analysis prediction template toolkit (aptt) for real-time image processing. In *IEE International Conference on Image Processing and its Applications, IPA'99*, 1999. Accepted for publication.

Pipeline:	(1)		(2)	
Set size	128	256	128	256
4	0.73	2.58	1.93	4.88
5	0.88	3.19	2.06	5.37
6	1.02	3.59	2.14	5.81
7	1.12	4.13	2.27	6.24
8	1.28	4.80	2.40	6.73
9	1.39	5.20	2.53	7.23
10	1.52	5.74	2.64	7.70

Table 1: Timings (s) for parallel pipelines on a Paramid

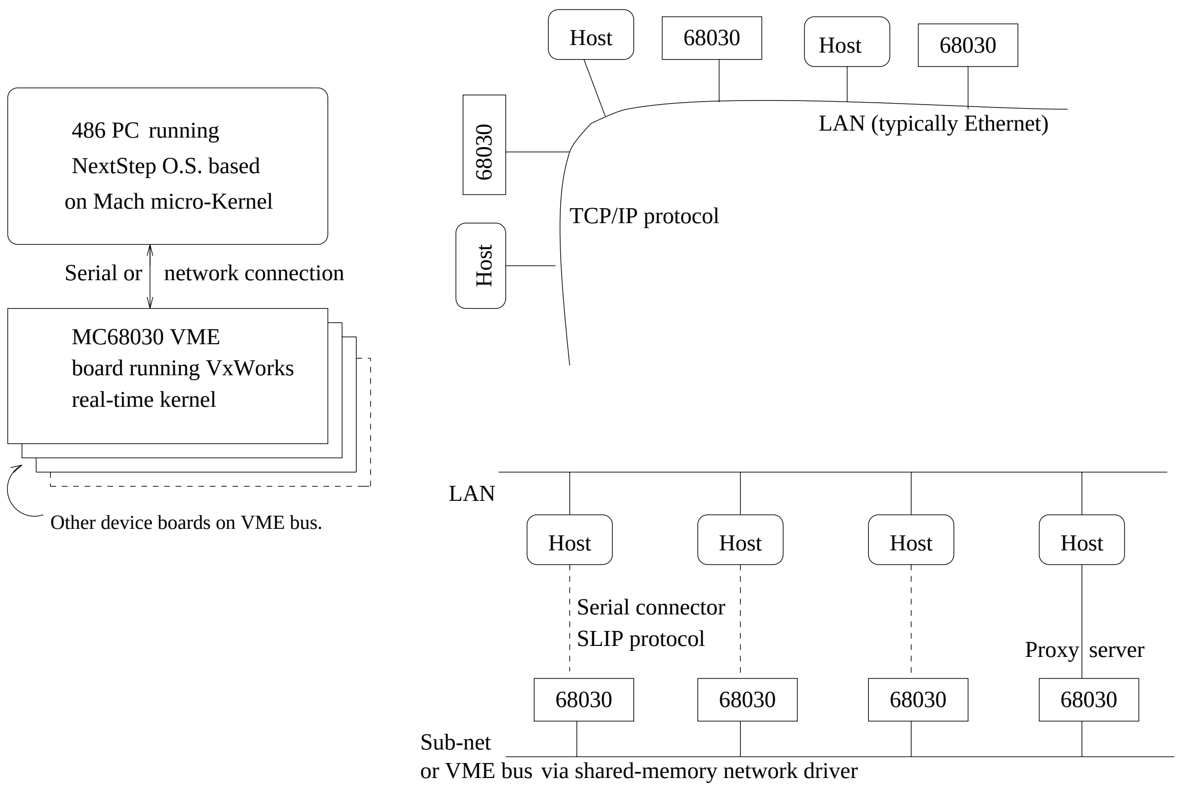

Figure 1: Alternative VxWorks Configurations

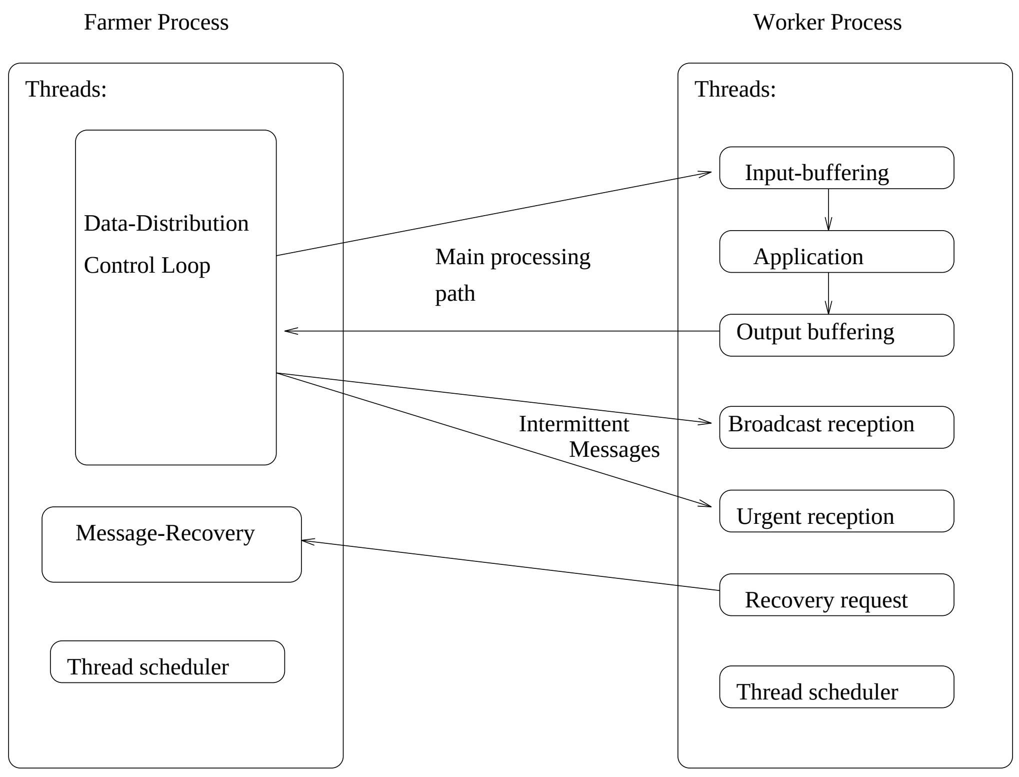

Figure 2: Data-farm Template Outline

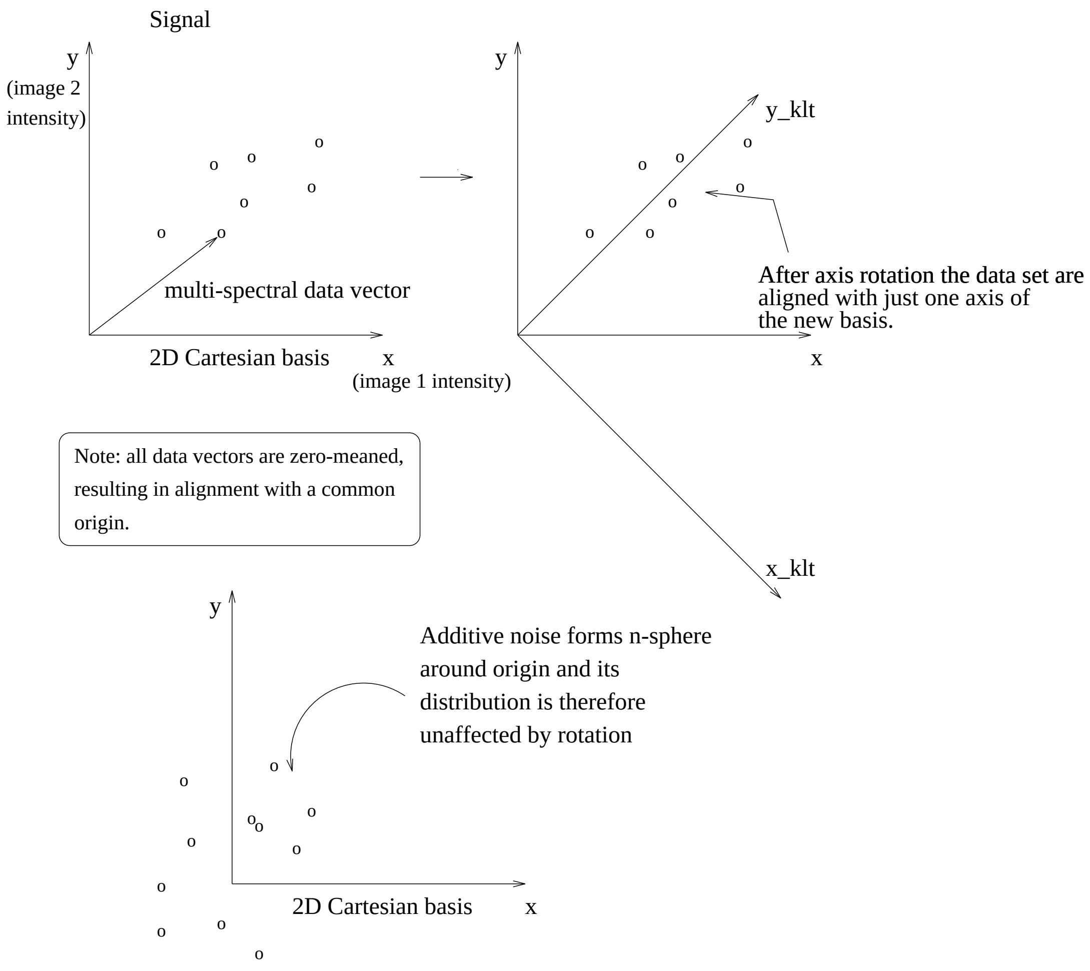
Signal
y
(image 2 intensity)
multi-spectral data vector
2D Cartesian basis
x
(image 1 intensity)
y
y_klt
After axis rotation the data set are aligned with just one axis of the new basis.
x
x_klt
Note: all data vectors are zero-meaned, resulting in alignment with a common origin.
y
Additive noise forms n-sphere around origin and its distribution is therefore unaffected by rotation
2D Cartesian basis
x

Figure 3: The Effect of the KLT Change of Basis on Signal and Noise (Additive)

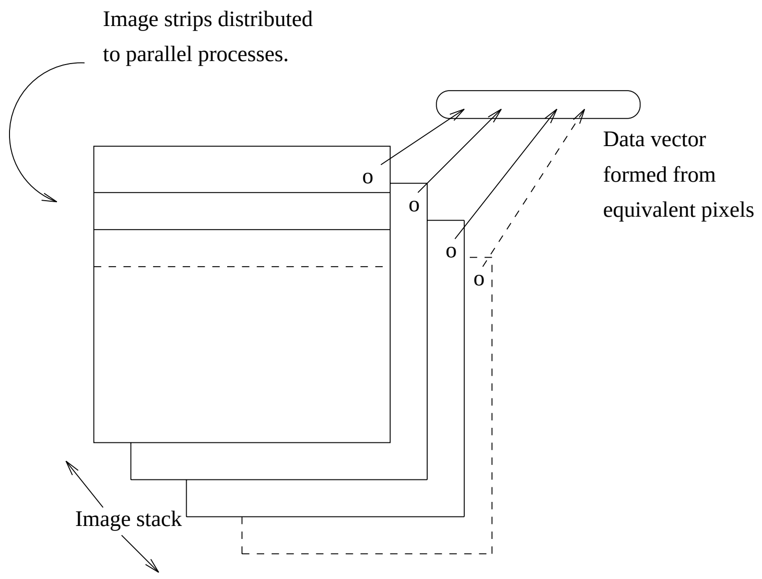

Figure 4: Decomposition of the KLT Processing

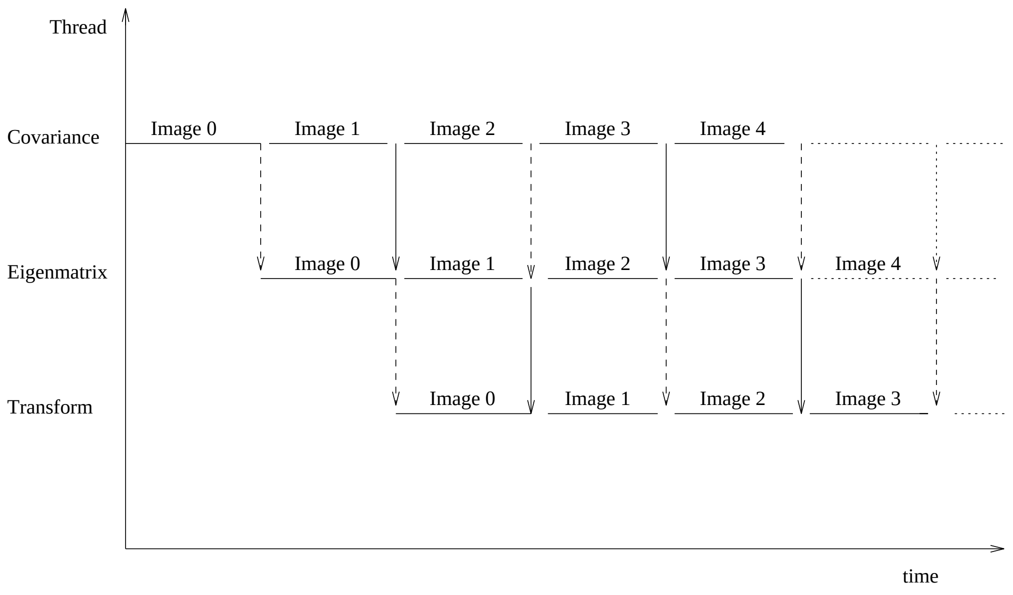

Figure 5: Ideal KLT Pipeline Timing

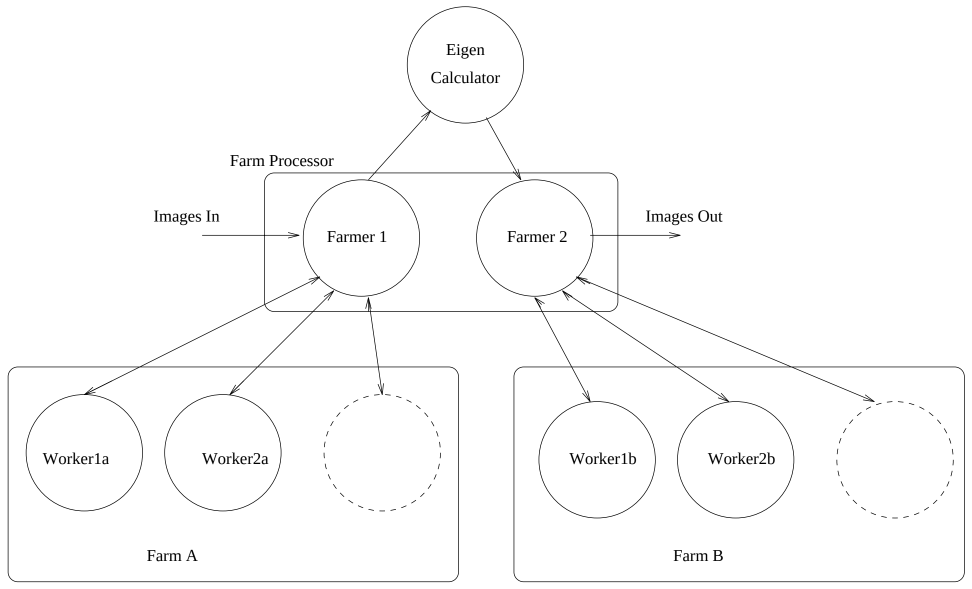

Figure 6: KLT Pipeline Partitioning

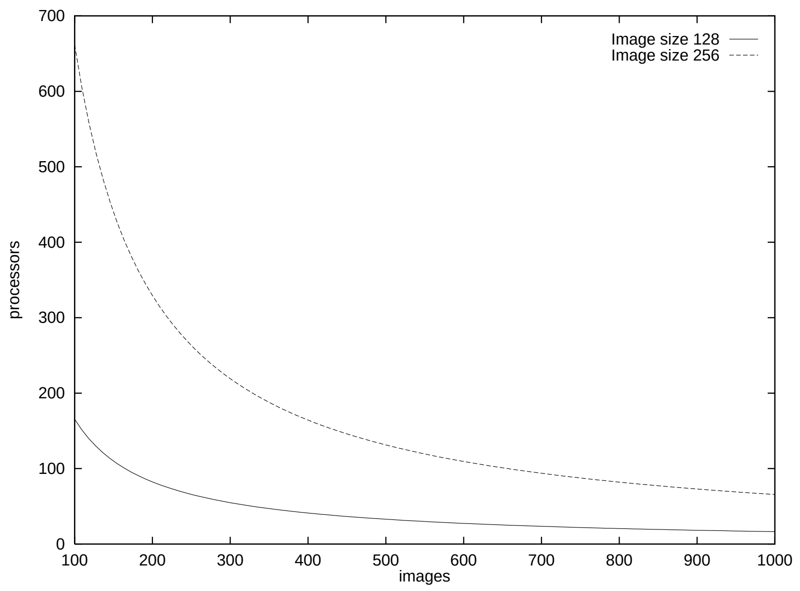

Figure 7: Scaling of Workload and Processors, for Square Images

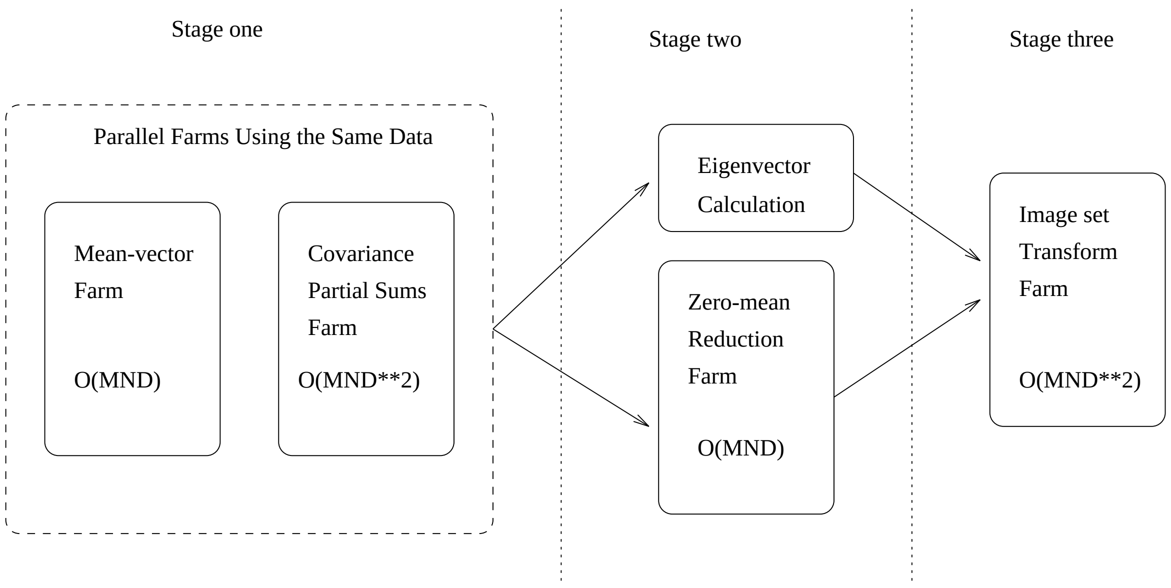

Figure 8: Alternative Partitioning of the KLT Pipeline

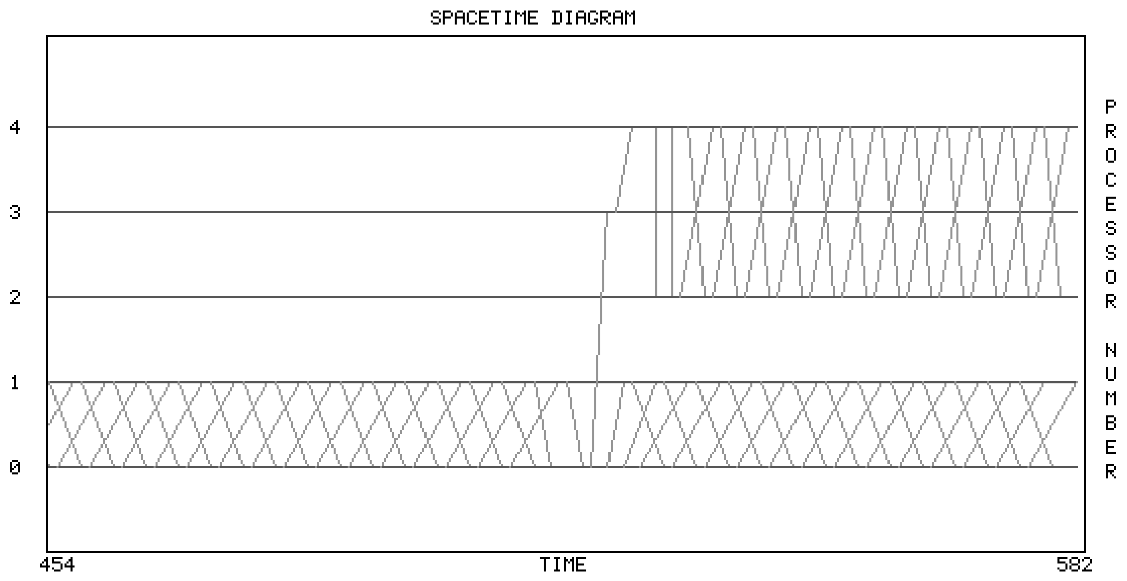

Figure 9: Logical Clock Trace of KLT Pipeline