Development of a Fine-grained Parallel Karhunen-Loève Transform

M. Fleury, R.P. Self, and A. C. Downton
Department of Electronic Systems Engineering
University of Essex
Wivenhoe Park
Colchester, UK
Tel.: +44-(0)1206-87217, Fax.: +44-(0)1206-872900
fleum@essex.ac.uk

Abstract

A high-performance Karhunen-Loève transform for multi-spectral imagery suitable for remote sensing applications has been prototyped on a platform FPGA, through hardware compilation onto a PC-based development board. Performance estimates suggest that the design will outperform implementation on a high-end microprocessor, given due attention to I/O (Input/Output). Detailed analysis of the design steps taken to produce a successful prototype are given. A design that addresses the issue of data bandwidth is included. General conclusions are reached for the utility of this architecture and design method for fine-grained parallel processing.

1 Introduction

The discrete time Karhunen-Loève Transform (KLT) [1,2] is a staple image-processing algorithm usually applied to a highly correlated image ensemble (set of related images). Unfortunately, the KLT kernel is data dependent with the result that no fast algorithm exists as the transform coefficients are not known in advance. An alternative way to improve processing speed is to introduce parallelism. Again for the KLT there is an impediment because, though two of the three processing stages can be readily parallelised, the middle stage cannot. Furthermore a processing architecture with a high data bandwidth is required, as all images in an ensemble are input to find the transform kernel. The first phase of the transform can be accomplished by integer arithmetic, whereas the second and third stages require floating point. The algorithm is deterministic with processing completely spatially localised. In other words, the inherent parallelism in the KLT is fine-grained.

In this paper, we prototype the KLT on an FPGA (Field Programmable Gate Array), and show that the current generation of FPGAs can effectively compete with a high-performance microprocessor, when processing image ensembles of size $6 \times 512 \times 512$ grey-scale pixels. In developing the KLT prototype engine, hardware compilation rather than a traditional Hardware Description Language (HDL) has been utilised, allowing the design to be approached as an exercise in parallel computing rather than hardware layout. The design is pipelined with some internal parallelism within a pipeline stage, and is data parallel across the image ensemble width. At a general level, the design method described in this paper addresses some of the concerns of those interested in parallel algorithms, who are faced with a relatively novel fine-grained parallel architecture, but whose background is largely in software. Though the KLT is a staple image-processing algorithm it also involves matrix manipulations characteristic of scientific programming in general.

One of the first KLT parallelisations [3] appears to have been on an SIMD (Single Instruction stream, Multiple Data stream) DAP (Distributed Array Processor) [4], which acted as a co-processor or rather intelligent memory to an ICL 2900 mainframe. In the ICL version, the DAP consisted of a 64×64 array of bit processors, each associated with a memory cell. On the DAP, I/O itself takes place in parallel across the bit planes of the memory. Unfortunately, bit-serial SIMD machines have not found sufficient applications in the marketplace, and hence, there is only one manufacturer known to us still supporting these fine-grained machines.

However, the advent of platform FPGAs, previously used only used as glue logic [5], has transformed that situation because there is now a mass-produced device with significant computational power that also supports fine-grained hardware parallelism. On these devices, the I/O bandwidth is an important feature. The Virtex-I from Xilinx [6], has 512 I/O pads to access 1 million logic gates, while the later version, the Virtex-II

has 1,108 buffered I/O pins on the largest member of the family. Additionally, on the Virtex-II Pro [7], with up to 10 million gates, between zero and twenty-four Rocket serial transceivers with individual data rates of 3.125 Gbps can be added as cores, as can up to four PowerPC cores. The Virtex-II Pro is, at the time of writing, at an advanced specification stage, and in the work reported in this paper, the Virtex-I was employed.

To overlap communication with computation a pipelined design is proposed, providing sufficient I/O if a continuous flow of image ensembles is available. This is exactly the form in which the KLT is likely to be applied in practice, for instance, if the algorithm is used to compress multi-resolution satellite imagery when there are usually large databanks of images. For example, the Landsat satellite was capable of producing hundreds of image ensembles per day, each ensemble consisting of ten different spectral channels at 512×512 resolution. Note that for such imagery standard codecs such as JPEG (Joint Photographic Experts Group) [8] are likely to lose important high-frequency features such as building edges [9].

One reason that take-up of the DAP floundered is that algorithms developed for this architecture were not transferable to other machines because each generation of bit-serial machine had different processing elements and interconnect topology [10]. This situation is not as critical for FPGAs because, at least in principle, from input of a netlist (list of a circuit's components and its interconnection), hardware synthesis can take place automatically. In the Xilinx case, automatic synthesis takes place through the agency of the Xilinx Foundation Series tools. At the programmatic level, there are hardware compilers, such as Handel-C from Celoxica Inc.[11], available that abstract from the hardware. The output of Handel-C is either a netlist, which can be in the industry standard EDIF (Electronic Data Interchange Format), or in RTL (Register Transfer Level) VHDL [12].

The critical part of the KLT algorithm is the middle stage, eigenvector extraction from the image ensemble covariance matrix, as it is unsuitable for parallelisation because of the low dimensionality of the matrix, and therefore forms a bottleneck in the pipeline. In the prototype analysed in this paper, the middle stage was performed on a Pentium host connected through a PCI (Peripheral Component Interconnection) bus to the RC1000-PP system development board from Celoxica Inc. [13]. However, the local PCI bus was clocked at 33 MHz, not 66 MHz, and was 32 bits not 64 bits wide. The RC1000-PP board, Figure 1, has four banks of 2 MB SRAM (Static RAM) memories, but these are accessible only four bytes individually (or one word) at a time. In [14], the bandwidth bottleneck on this and other boards is essentially recognized, as a task parallel processing model is implemented for the RC1000-PP. These comments do not in any way constitute a criticism of the RC1000-PP but merely reflect the constraints imposed in a realistic development environment for this board.

Whether task parallelism or pipelining is used, respectively an overlap or balance between tasks on the processor and reconfigurable device is required. We were interested in whether a balance might exist if

direct communication were possible to the Virtex-I, treating the RC1000-PP as a development board and not the target system. In fact, an on-chip PowerPC405 core (running at 350 MHz) would remove any I/O bottleneck, though the proposed version of the PowerPC core for the Virtex-II Pro supports fixed-point not floating point operations. In due course, co-simulation may be a way of estimating the transfer of algorithms such as the KLT to a Virtex-II, though there will remain a problem of providing an early development environment due to I/O limitations on PCs. A further possibility is to include a specialist eigenvector engine, and indeed systolic designs have long existed for symmetric matrices, for example [15].

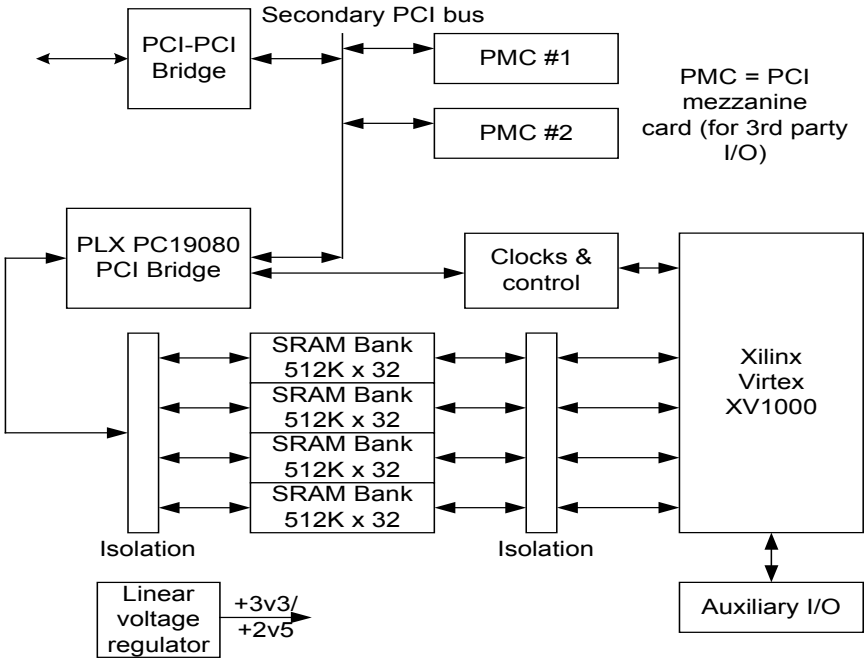

Figure 1 RC-1000PP board from Celoxica Inc. [12]

Section 2 of this paper describes the KLT algorithm and its computation. Section 3 introduces the development environment for the KLT prototype as this may be relevant for other such designs. Section 4 is a detailed analysis of the fine-grained architecture of the prototype, whereas Section 5 presents initial results and timings that establish the main result for the potential performance of a KLT engine. Section 5 also includes a design for an I/O subsystem for a Virtex-II Pro based design. Finally, Section 6 draws some conclusions.

2 The KLT

The nomenclature of the discrete-time KLT is confused [16]. 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 optimality features of the KLT principally arise when the data is zero-meaned. The term Hotelling transform is sometimes reserved [17] for the discrete version of the transform.

The KLT differs from other common orthogonal transform algorithms, such as the Fourier transform, in two respects: the transform kernel is data-dependent; and the transform is applied to an image ensemble, a collection of usually strongly correlated images. 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 be regarded as acting on a sample set of images, from a possibly infinite population of images.

The KLT is most widely used in applications such as multi-spectral analysis of satellite-gathered images [18] through the spectral signature of imaged regions or, as mentioned in Section 1, for compression purposes. Multi-resolution scanning electron-microscope images (micrographs) are also analysed by means of a KLT. For compression applications, images in the transform domain (channels) with the least variance are abandoned. While it is true that the channel with the most variance contains the gross features it is also true that other channels contain detailed features that, by their texture, identify regions within the image. Another application is the removal of a noise-bearing channel.

The KLT has also been applied to sets of face images [19]. 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. Notice that the dimensionality of the image set is potentially much larger in facial recognition applications.

The KLT has several features attractive to such applications. Amongst the features relevant to the computation of a KLT are:

- The KLT transform achieves optimal data compression in the mean-square error sense. That is $\varepsilon(k) = \mathrm{E}[(x-\hat{x})^T(x-\hat{x})]$ is a minimum, where $\hat{x}$ is the representation of vector x truncated to k terms. E is the mathematical expectation operator and superscript *T* represents the transpose. The minimal orthonormal basis set is found by the method of Lagrangian multipliers, using the orthonormality of the basis vectors as the constraint [20] or by consideration of bit allocation upon quantization [21]. The KLT projects the data onto a basis that results in complete decorrelation, though again 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. Informally, this property means that a transformed image set can be compressed by discarding all images with numbering greater than k (with ordering usually by eigenvalue), and subsequently reconstructing by using the same transform kernel but with the discarded images replaced by zero images. Though retained images are normally taken in descending order of eigenvalue, there is an

alternative and related transform, the lower-triangle transform that retains images in ascending order of eigenvalue.

- 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 or indeed a Discrete Cosine Transform) is available [22]. However, this is not the case for multi-spectral or multi-resolution images, or in any of the applications which attempt to decorrelate strongly correlated sequences of images.

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, though some iterative algorithms also exist [23], as well as a neural-net algorithm [24].

In Figure 2, the image intensities for the equivalent pixel in each image serve to form a multi-spectral data vector. The sample covariance matrix is formed out of the zero-meaned data vectors by correlating each pixel in an image with the equivalent pixel in every image. It turns out that the eigenvectors of the covariance matrix are orthogonal and thus, when projected (through zero-meaning) onto the original basis, have the effect of an axis rotation. Moreover, the image data tends to be aligned within the reduced dimensional space formed by a few axes in the new basis. Thus, whereas in Figure 2, just two images have been sampled, if a number of images were sampled then the number of eigenvector axes (marked `_klt` in the figure) would be the same as the number of images, and the clustering would be in the sub-space formed by a few but not all of the eigenvectors. It is also apparent from Figure 2 that, if additive noise is present, the Signal-to-Noise Ratio (SNR) will be improved after transform (*i.e.* projection onto each in turn of the new axes) in the higher-component (indexed by the eigenvalue magnitude) images, as no realignment takes place. (Noise-dominated image sets may be analyzed through the low-component images.)

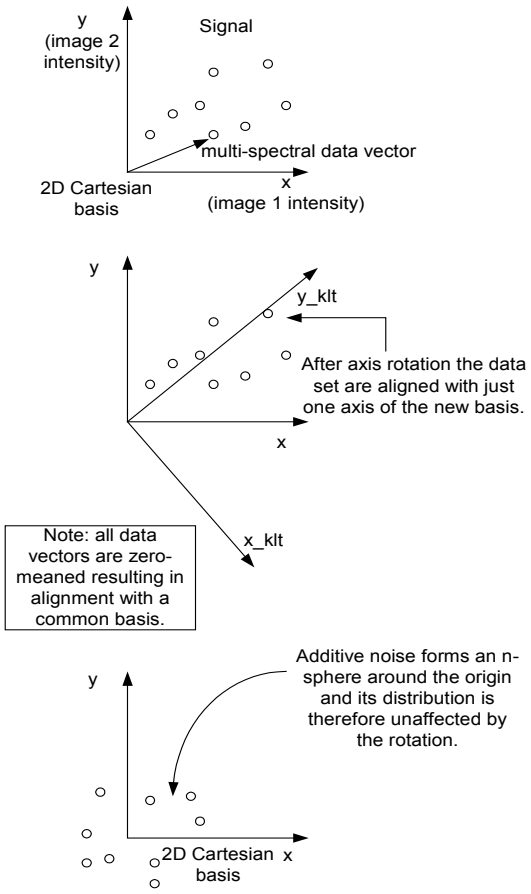

Figure 2 Effect of the KLT change of basis on signal dimensionality and noise

2.1 KLT algorithm

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 $N \times M$ then form the column vectors $x_k = (x_{ij}^0, x_{ij}^1, \ldots, x_{ij}^{D-1})^T$ for $k = 0,1,\ldots,MN-1, i = 0,1,\ldots,M-1$ and $j = 1,2,\ldots,N-1$. Calculate the sample mean vector:

$$m_x = \frac{1}{MN}\sum_{k=0}^{MN-1} x_k$$

Use a computational formula to create the sample covariance matrix:

$$C_x = \frac{1}{MN}\sum_{k=0}^{MN-1} x_k x_k^T - m_x m_x^T,$$

which is appropriate if the image ensemble is formed by a stochastic process that is wide-sense stationary in time. It is important to note that C_x has rank *D*, which being the number of images in an ensemble is, in

practice, usually below ten.

Form the eigenvector set:

$$C_x u_k = \lambda_k u_k, \quad k = 0,1,\ldots,D-1,$$

where u_k are the eigenvectors with associated eigenvalue set λ_k. C_x is a symmetric and non-negative definite matrix, which implies that the 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. This property arises because $\varepsilon(k) = \sum_{j=k+1}^{\infty} \lambda_j$.

Finally, the KLT kernel is a unitary matrix, *V,* whose columns, vectors u_k (arranged in descending order of eigenvalue amplitude), are used to transform each zero-meaned vector:

$$y_k = V^T (x_k - m_x)$$

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. Calculate the set of outer products and sum, $\sum_{k=0}^{MN-1} x_k x_k^T$, in $O(MND^2)$.

- Form $m_x m_x^T$; subtract matrices to find C_x; and find the eigenvectors of C_x. The eigenvector calculation is $O(D^3)$. Convert the x_k to zero-mean form in $O(MND)$.

- Form the y_k by $O(MND^2)$ operations.

Figure 3 shows these equations represented as a pipeline. The KLT comprises three distinct processing stages: 1) covariance formation, 2) eigenvector calculation, and 3) eigenspace transform. Since the covariance matrix is generally too small to justify parallel decomposition in general or hardware implementation on an FPGA in particular, the total implementation complexity is $O(MND + MND^2)$.

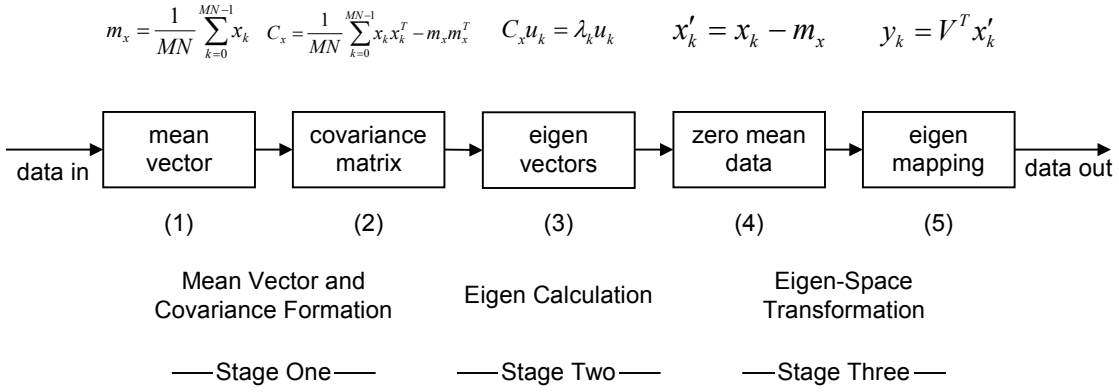

Figure 3 Mapping the KLT to a pipeline

2.2 Computational considerations

Notice that in Section 2.1 it has been tacitly assumed that the variance is calculated by a computational formula, as in Stage One part (2) of Figure 3, rather than by the direct calculation

$$\frac{1}{MN}\sum_{k=0}^{MN-1}(x_k - m_x)^2$$

However, this last equation requires prior calculation of m_x the mean vector, which requires normalization, *i.e.* an additional division by *MN*, which, of course, leads to a real-valued number, not an integer. The normalization can be postponed if the computational formula is employed, as the final subtraction of the constant $m_x m_x^T$ can be postponed in the expression

$$\frac{1}{MN}\sum_{k=0}^{MN-1} x_k x_k^T - m_x m_x^T$$

Equally, the division of $\sum_{k=0}^{MN-1} x_k x_k^T$ by *MN* in the expression can also be postponed. Finally, also to avoid real-valued numbers, an unnormalized version of the mean can be initially calculated , *i.e*

$$m_x = \sum_{k=0}^{MN-1} x_k \,.$$

All postponed calculations take place on a host microprocessor, thus allowing floating point calculations to be conveniently employed.

On the other hand, normalization for arbitrary image dimensions requires fixed-point operation for real-valued numbers, and hence all operations would take place in fixed-point form on the FPGA. A commitment to fixed-point operation on the FPGA both complicates the implementation and results in

increased word length so as to minimize arithmetic errors. Increased word length impacts on the sizing of multiplier hardware and leads to a rapid increase in gate count. For example, a word length of 5 bits results in 680 gates usage for a multiplication on the Virtex-I, whereas a word length of 25 bits results in a gate usage of 16,120. These points are returned to in Section 5.2.

Calculation by the direct formula for the variance also requires m_x to be found before forming $(x_k - m_x)$, i.e. imposing a sequential order on the calculations. In the computational formula, $\sum_{k=0}^{MN-1} x_k x_k^T$ can be calculated in parallel with unnormalized m_x, resulting in a saving of the time needed to calculate unnormalized m_x.

In Figure 4, the pipeline has been decomposed so that Stage One parts (1) and (2) of Figure 3 are in parallel. Additionally, all normalization takes place within Stage Three of Figure 4.

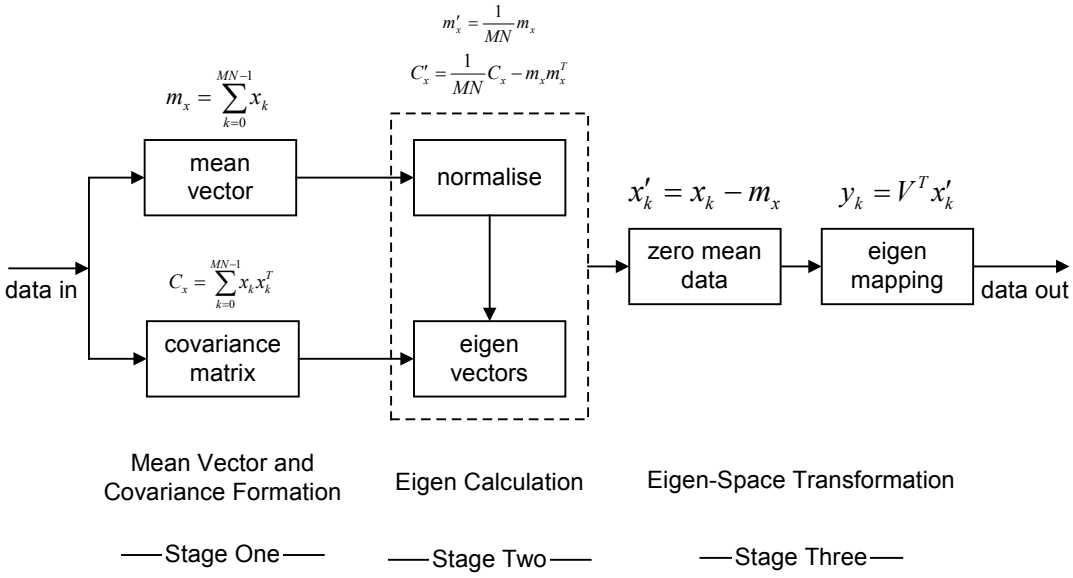

Figure 4 Parallel pipeline algorithmic decomposition

3 System development

3.1 Handel-C, CSP, and SystemC

Handel-C hardware compilation takes a program written in Handel-C and outputs a netlist representing pre-defined combinational circuit elements onto which elements of the language are mapped [25]. While Handel-C has the `look-and-feel' of C, Handel [26] owes its origin to the parallel processing language occam [27]. Put another way, Handel-C is a strongly typed variant of C with additional parallel constructs.

Occam itself is an implementation of the specification language CSP (Communicating Sequential Processes) [28], which, however, supports an asynchronous time model, not a synchronous clocked model. We have experimented with a higher-level design model that sits above the Handel-C programmatic interface and allows a first-level design that is free from clocking constraints. In this design model, KLT algorithmic components are modelled as CSP tasks. In the CSP model, channels [29] serve both as a mechanism for synchronisation of asynchronous processes, and also for data transport. In Handel-C, channel communication is synchronous with clock edges but the moment of communication is dependent on the completion of handshaking. An alternative method of design is based on registers placed between each module of the application. In the wider sense, this is akin to shared-memory parallelism. This more low-level form of design does not in our experience (Section 4.2) bring benefits to justify the extra time taken in design, and almost certainly will not scale. Further detailed comparison with occam, CSP, and conventional Hardware Description Languages (HDL's) can be found in [30]. In [30], it is argued that though a variety of tools exist to accomplish (say) the retiming of digital circuitry, these are mainly aimed at those with a hardware background, whereas, typically, those involved in parallel computing are more likely to favor a software approach to retiming hardware. Handel-C also offers an integrated approach, without the need for auxiliary software tools.

SystemC [31] is an industry *de facto* standard, which addresses the need for multi-vendor tool interoperability. The founding members of the consortium have each contributed expertise in the form of class libraries for an implementation of SystemC, and, in particular, in this design use has been made of Frontier Design's contribution, which is a class library for fixed-point data-type mapping. The public domain version of SystemC does not support hardware synthesis but it does include a simulation kernel that can be linked in to allow the SystemC model to act as an executable specification. This facility has been availed of in the KLT development; refer to Section 4.2.

3.2 Run-time executive

The RC 1000-PP board upon which the Virtex XVC1000 FPGA is mounted, has two 8-bit ports allowing direct communication between the FPGA and PCI bus. These ports are accessible by an API (Application Programming Interface) called from a host stub program. There are matching calls within Handel-C. As previously mentioned in Section 1, four external memory banks of 2 Mbytes are available, accessible by FPGA and PCI bus alike (using an interface construct in Handel-C to associate a RAM bank data type with an external device). DMA (Direct Memory Access) transfer to these banks is via API calls on the host. Access conflict resolution is through hardware semaphores [13]. SRAM banks on the Virtex, comprising 64 bits taken from LUTs (Look-Up-Tables) per each of 64×96 CLBs (Configurable Logic Blocks) on the XCV1000, are accessible by default, whereas a further 32×4096 bits of dual-ported SRAM (block selectRAM), forming blocks on the perimeter of the CLB array, can be used to communicate between independent code blocks. The LUT-based SRAM is distributed and accessible in parallel, whereas access to

block SRAM is serialized, and therefore reduces performance (Section 5.1). However, using LUT-based RAM impacts on CLB usage, restricting circuit size.

We facilitated development of the KLT by first constructing a task harness that avoided the need to redesign access to memory banks, synchronization with the host, and communication between tasks. Figure 5 shows the run-time executive (RTE) design arranged as a set of self-contained components. The number of tasks shown is arbitrary. All components run in parallel. Communication to the host is through a simple signaling protocol. The modular arrangement makes possible (say) the removal of the debug component or the substitution of one memory manager for another. Given our RTE, developers have the capability to utilize components of the RTE during application development and omit them for efficiency once a design is complete, while we can easily reconfigure the RTE components to match an alternative FPGA architecture or a new device generation. As Section 5.3 indicates, inclusion of the RTE had minimal impact on the performance of the KLT.

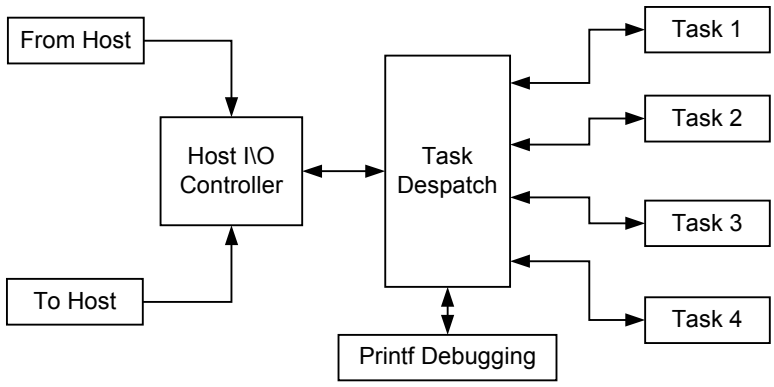

Figure 5 Block diagram of the RTE

4 KLT pipeline

The KLT pipeline comprised three distinct processing stages, which were mapped to CSP and in turn to RTE tasks: covariance matrix formation, eigenvector calculation, and eigenspace mapping. As mentioned in Section 1, tasks one and three were implemented on the Virtex-I XCV1000 FPGA to exploit data parallelism, while task two is implemented in software to take advantage of the floating-point efficiency available from a general-purpose processor, which, initially for development purposes, was a Pentium II running at 220 MHz. Notice, however, that both the Pentium and the Virtex-I are high-power devices, with the FPGA consuming at least 10 W.

A small pilot implementation of the KLT was completed first using a 20×6 matrix with data and results from [32], corresponding to six images with just 20 data items in each. This enabled the results of the implementation to be verified to ensure correct working, after which gate usage for a much larger image

size could be estimated simply by replicating the basic KLT building block. One of the reasons for the small size of the pilot implementation was the I/O bottleneck on the RC1000-PP board already referred to in Section 1. Another potent reason is that compiling under Handel-C and performing place-and-route on a full-scale design, even with a high-performance host, took about one hour. Repeatedly changing a design and testing it can be a time-consuming activity. Results from the scaling exercise are reported in Section 5.2.

Parallelism arises in various ways in the design. First and obviously the stages of the KLT pipeline can be partially overlapped in time. In fact, the second stage on the targeted Virtex-II Pro could be elided with the third and final stage as it consumes little time (Section 5.3 & 5.4), making for a reasonably balanced pipeline. Secondly there is internal parallelism in the first stage between the covariance and mean-vector sub-stages. Thirdly and most importantly, once the pilot design is replicated (Section 5.2) then there is data parallelism across the images' width. One of the features distinguishing Handel-C from some HDLs is that nesting of parallelism is not constrained to two levels making it relatively easy to scale a design.

The design of the first stage is reported in Section 4.1, whereas details of the development process are reserved for the third and final stage of the pipeline in Section 4.2

4.1 Covariance matrix formation and eigenvector calculation

The main issue in designing the first stage of the pipeline, Figure 6, was data management. Figure 6 includes the per-row cost of operations. In Figure 6, the 48 cycles for transferring results to the host is amortized over all twenty data rows input and, hence, with scaling its significance will reduce. Notice that the 1.5 cycles refers to a per-byte average for the whole row, as the number of bytes in the image ensemble direction, six, is not a multiple of the word size (refer to Section 4.3 to see how the data I/O was handled). The two calculations of the mean vector and covariance vector can be performed in parallel, following the algorithmic rearrangement described in Section 2.2, which is a saving of 6 cycles. It is possible to improve the clock cycle count for the covariance calculation by the addition of extra Multiply-Accumulate (MAC) units. However, it turns out that it is only worthwhile to add one MAC, as after that the bottleneck in the overall pipeline shifts to the third stage. Calculations showed that adding one MAC would exceed the capacity of the device available to us (refer forward to Table 6). These points are returned to in Section 4.4.

In general, input is available from the memory banks in the form of 4×8-bit data samples, representing one-byte samples from each of the four memory banks. This is a significant constraint on I/O on the RC1000-PP board. As mentioned in Section 2.3, the order of operations, from the two alternative ways of calculating the variance, is selected to avoid fixed-point calculations. In the KLT pilot, the Mean Vector and Covariance summation sub-tasks processed data in rows of 6-byte blocks. This required data buffering and data alignment between memory I/O and processing. Once the test data had been copied from the Pentium II host to the SRAM bank connected in turn to the FPGA then each 6-byte block took 1.5 clock

cycles to pass to the parallel sub-tasks. Synchronisation between the parallel covariance and mean-vector sub-stages was by a 48-bit wide channel, from which data input was copied into two buffers for each of the sub-stages. Covariance calculations include a multiplication whereas calculation of the mean vector involves addition. As the Virtex family of FPGAs include fast add carry-chains in the vertical direction, reducing the clock width of addition and subtraction operations, though, unless hardware multipliers are present, this is a theoretical advantage (which point is returned to in Section 5.4).

Upon completion of processing of all rows, the first stage terminates causing a notification to be relayed to the host processor. The host processor then forms the covariance matrix and produces the eigenvectors, which are subsequently copied to an SRAM bank in order that the final hardware processing stage can be completed.

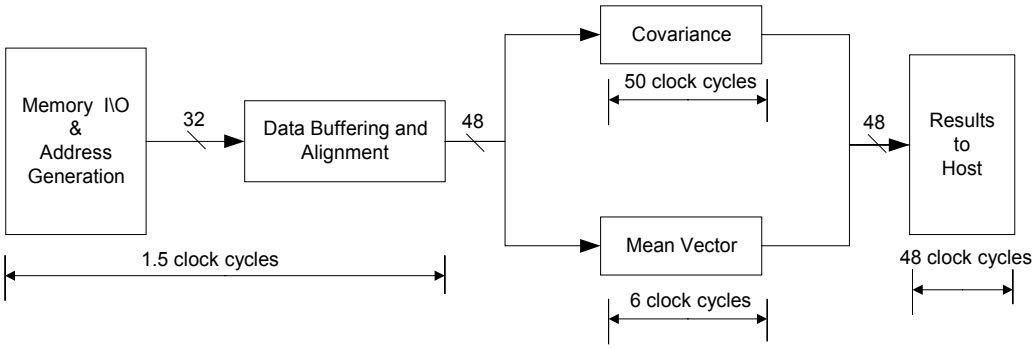

Figure 6 KLT stage 1 data flow

4.2 Eigenspace mapping development process

The layout of the eigenspace mapping stage is shown in Figure 7. The computation performed by the pipeline consists of mean-normalisation of vector elements, transformation of elements into eigenspace, and summation of elements in an accumulator. Source input and result output data pipeline stages manage the flow of data between external SRAM banks and the KLT pipeline processor. Normalization- and eigenvector data are stored in distributed RAM elements present on the FPGA selectRAM.

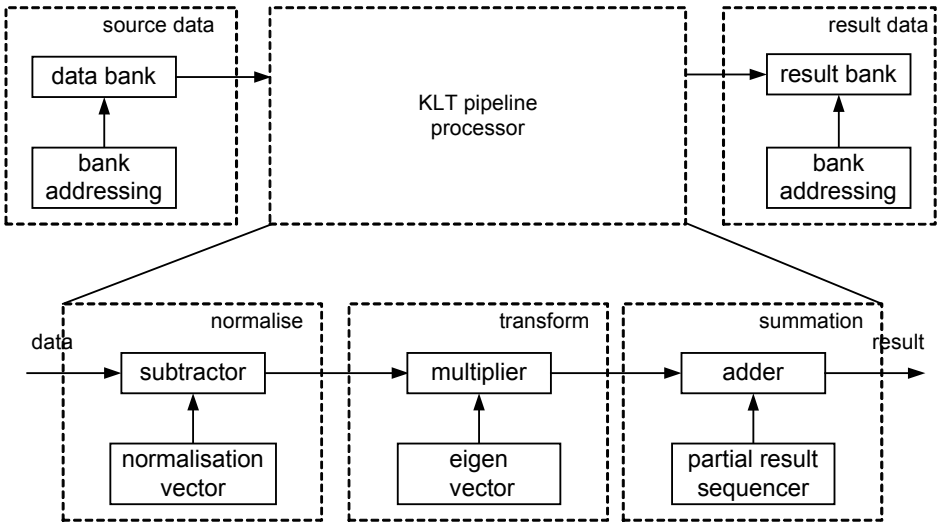

Figure 7 Normalisation and Eigenspace transformation stage

At the normalisation sub-stage, incoming data is transformed by subtraction from a normalization vector stored in distributed RAM. The normalised data is then transformed into eigenspace by multiplication with eigenvector data, also stored in distributed RAM memory. Finally, the summation sub-stage accumulates the partial results generated by the multiplier into a result data element for storage in the result SRAM data bank. Counters, not shown in Figure 7, are used to generate the distributed RAM lookup table addresses and sequencer control logic.

The eigenmapping task was first implemented using ANSI-C running sequentially on a Pentium so that the algorithm could be evaluated and baseline source and result datasets created. This model was modified to include SystemC fixed-point modelling to establish candidate pipeline partitioning points and optimal register widths from the input data. From these results, it was concluded that the pipeline should be constructed from Normalization and Eigenspace transformation stages, with separate sub-stages for input and output, corresponding to the top-level model shown in Figure 7.

The next step was to verify pipeline operation and generate FPGA-based module timing results to identify which pipeline stage should be speeded up in the next iterative step. Creating a first-cut implementation of the eigenmapper involves defining the channel-based modules from the pipeline partitioning exercise and performing a cut-and-paste of code from the corresponding parts of the C-software design. In practice, this Handel-C porting exercise is essentially transparent as the Handel-C language supports most ANSI-C statements, although some adjustment is needed to include variable width specifications - in this case, already determined by the fixed-point modelling.

This initial design was implemented as four sub-stages, each performing their specific operations sequentially. The sub-stages concerned and sub-stage latency figures relating to the processing time for a single data element appear in Table 1. From these figures, it can be seen that the Eigenspace

transformation sub-stage has the largest latency, and therefore defines the overall pipeline throughput (the reciprocal of the worst-case stage latency). Since it dominates pipeline performance, this stage was the subject for optimisation in the next design step.

Pipeline sub-stage	Stage Latency (clock cycles)
Source data	2
Data normalisation	3
Eigenspace transformation	8
Result storage	2

Table 1 Pipeline latency

Eigenspace transformation consists of two sub-operations, vector transformation and partial result accumulation. In the revised design, an Eigenmapper task was created that performs both of these operations. Performance was improved both by converting sequential code to parallel operation, and by implementing each sub-operation as a separate pipeline task. The Eigenmapper task can be implemented using either register- or channel-based assignment to interconnect pipeline sub-stages. In order to provide a comparison of the merits of each approach, both alternatives were implemented.

In the channel-based design, the two-section micropipeline was replaced with three parallel executing 'micro-tasks' connected via channels. The revised design is shown in Figure 8, with eigen-data running in parallel with the transform stage. However, the main change over Figure 7 is not the extra micro-task, but the inclusion of channels to decouple each micro task. While decoupling could also be achieved by the inclusion of registers, once the new micro-task is introduced extra registering is required to equalise the timing. This could be achieved by manual intervention or, depending on availability, by suitable software tools, but Handel-C channels provide automatic retiming without going outside the hardware compiler.

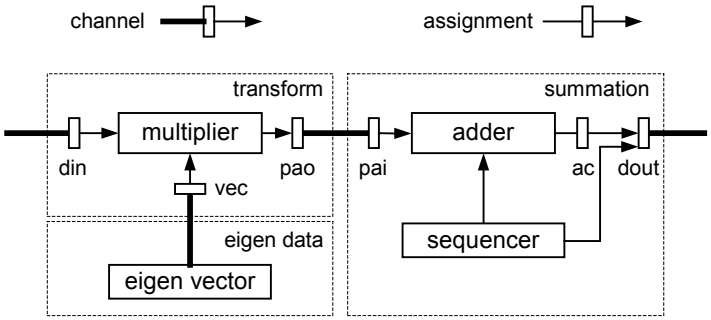

Figure 8 Channel-based Eigenmapper task

The transform and summation tasks are each partitioned into single cycle (channel-based) sequential input and output stages, and a single cycle processing stage, and therefore take three clock cycles to process

a data element. The eigen data task is constructed in similar fashion, but only needs two clock cycles to complete, as there is no data input stage. The unequal execution times of the transform (three cycles) and eigen-data (two cycles) task operations would, in the case of a register-based design, necessitate the insertion of additional registers to synchronise input from registers `din` and `vec`. Channels, however, handle this requirement automatically, as is explained by the Handel-C source code in Figure 9. In Handel-C, channel input and output are identified respectively by the ? and ! operators. The `par` keyword indicates process parallelism. Each parallel process must complete before a `par` block is left.

```
par{
// eigen data task
while(TRUE){
    vchan ! eigenram[addr];
    addr = addr == 35 ? 0 : addr + 1;
}

// transform task
while(TRUE){

    // read data into multiplier input registers
    par{
        vchan ? vec;
        dchan ? din;
    }

    // eigen-space transformation
    pao = din * vec;

    // eigen-space partial result output to summation stage
    ochan ! pao;
}

// summation task code
...
}
```

Figure 9 Eigenmapper code

The code shown in Figure 9 employs `while` loops to implement two continuously running tasks, executing in parallel. The `par` block in the transform task synchronises the flow of input data and eigenvector elements into the multiplier. Because the thread of execution only passes out of a `par` block once all enclosed statements have completed, data input to the multiplier is correctly synchronised irrespective of the relative execution time of the respective tasks.

This design improved over the register-based design both in term of performance and design resilience. With regard to performance the clock-speed is increased from 30.58 MHz to 31.44 MHz and pipeline start-up latency was reduced by one clock cycle as channel handshake auto-scheduling avoids the need for additional (redundant) datapath retiming registers. Gate count was increased from 14,921 to 15,099 gates, but in practice this difference is insignificant given that the Virtex family of devices has capacities of 1-10 million gates.. Therefore, the main applicable gain from using Handel-C is the automatic retiming afforded by channels.

4.3 Data I/O

The application test data was arranged as a twenty-row by six-column matrix (Section 4), with each data element being 8-bits wide. These data were mapped into the FPGA data space as thirty rows by 32-bits. The data were organised in this fashion in order to maximise data transfer between the FPGA and external SRAM memory banks. The memory banks can be configured for 8- or 32-bit transfers - clearly, 32-bit data accesses offered a bandwidth advantage over the 8-bit alternative.

The memory management task was to extract the required data element from the appropriate memory bank 32-bit word. Data bytes were accessed by reading SRAM bank data words into a register and writing data placed at the first-byte position into the pipeline data stream (see Figure 10). Performing an 8-bit right shift for three subsequent clock cycles realigns higher-order bytes into the lower-byte position. An SRAM read operation was then generated in order to access the next data word.

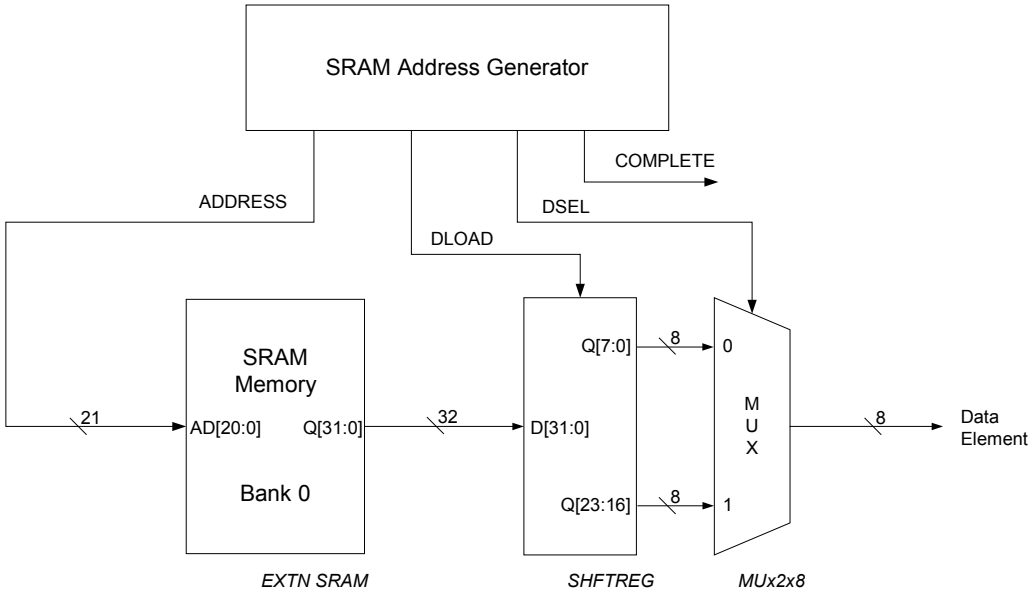

Figure 10 Data pump design

The heart of the Data Pump was the Address Generator, Figure 11. The objective was to create a design, which was simple though sufficiently general to be easily changed as it evolved. The approach taken was to store the SRAM address offsets and various control signals as a lookup table in ROM. A counter provides for basic scheduling, while a 'row0\row1' register provides the ROM address offset that selects between the scheduling requirements of even (ROM addresses 0-5) and odd rows (ROM addresses 8-13).

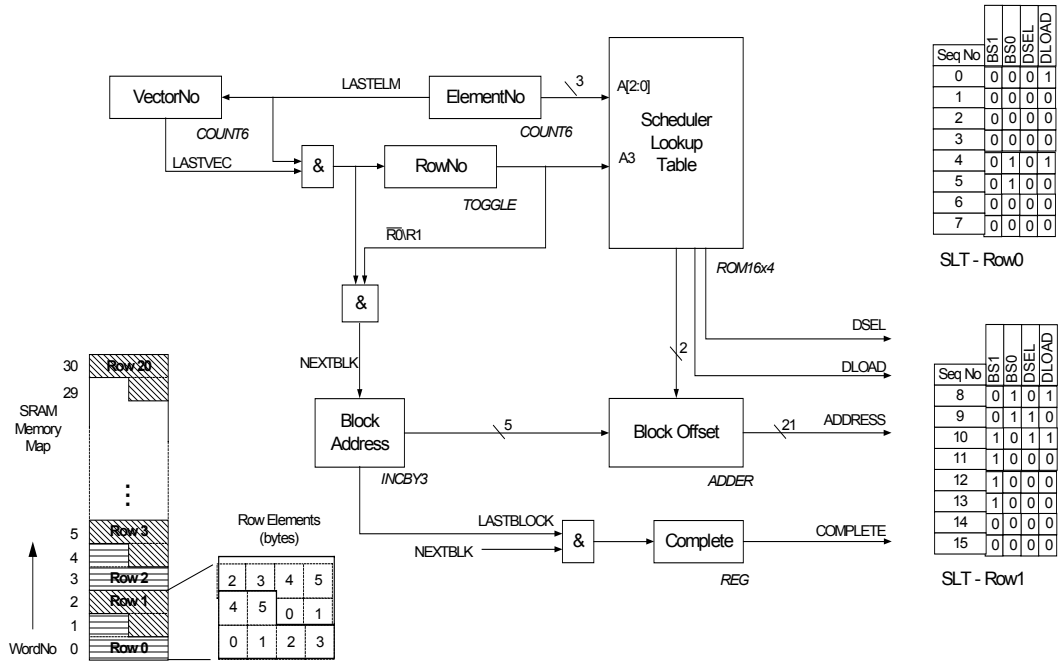

Figure 11 SRAM address generator

4.3 Summary

Figure 12, shows the final pipeline as implemented, with data flows. Tasks one and two of the RTE (Section 3.2) synchronize stages one and three of the KLT pipeline, and run on the Virtex-1 on the RC-1000PP board. Stage two runs on the host PC. In stage one, data in (di) are used to form a mean vector (mv) and covariance vector (cv) before data are output (do). Similarly the data are zero-meaned (zm) and mapped to eigen space (em-mac) in stage three. RC-1000PP control lines (crtl) provide synchronization between host and FPGA, and data are passed over the PCI bus to the memory banks (PC), actually on the RC-1000PP board.

Stages one and three were coded using the Handel-C hardware compiler. Construction of the two stages was accelerated by means of the RTE, also coded in Handel-C. Additionally, the data I/O of Section 4.2 was coded in Handel-C. The Handel-C simulator was used to iteratively refine the design. Output from Handel-C is a netlist. The netlist acts as input to Xilinx Foundation Series place-and-route tools, which automatically produce the reconfiguration bitfile. The bitfile is downloaded to the FPGA at boot-time. The host program is a normal `C' program with library calls to perform I/O to the FPGA board. Dimensioning of the fixed-point calculations in stage three was performed by means of a SystemC library kindly provided by Frontier Design.

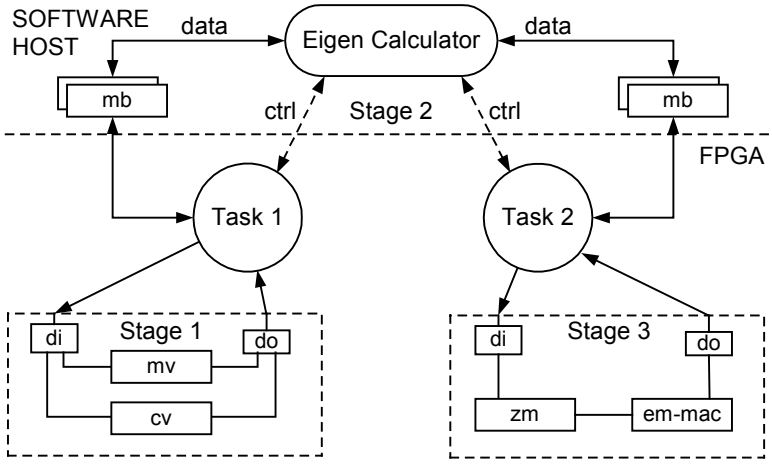

Figure 12 KLT pipeline implementation

5 Results

5.1 Optimisation of the pilot design

Table 2 shows results from the small pilot study. The system gate count is a Xilinx metric based on the notional cost of implementation on an ASIC. Details of the accounting procedure for this metric are given in [33], and it should be noted that designs that include on-chip (FPGA) memory attract relatively large gate counts. There are two identical slices per CLB on a Virtex FPGA, and in general `slices' are a more reliable metric. Together both stages occupy approximately 7% of the Virtex-I capacity. The clock times given, though taken from the output of the place-and-route tool (to position and connect the netlist components on the FPGA) rather than the implementation, were found to be very reliable. Section 5.3 gives absolute timings from implementation on the device.

The clock speed of the stage three design was lower than expected. From earlier timings made on arithmetic operators a clock rate in the range 20-26 MHz was thought to be a realistic target. It should be noted that others report similar clock speeds in this range for applications on the RC1000-PP board, for example in [14]. One problem was thought likely to be caused by the fact that the path through stage three is composed solely of combinational logic. Additional pipeline registers to limit propagation delay were then included in the design. Registers can also remove positional dependencies between components.

Obviously from Table 2, stage three will dominate the performance and therefore should be optimised. In Table 4, the successive optimisation steps are detailed. In this table, the statistics refer to both stages one and three. Step 4 was an experiment rather than an optimisation, as the confusedly named blockSelect RAM is RAM at the boundaries of the device, whereas selectRAM is RAM formed from CLB LUTs. Because memory usage is increased, the system gate count increases, though not the number of slices, whereas the clock speed is affected by the increased propagation delay to the boundary of the 'chip. Step 5 decouples selectRAM from multiplier and causes the main step up in performance. Replacing the Handel-C

netlist multiplier with a CoreGen Multiplier, which is soft IP (Intellectual Property) in EDIF, has the effect of removing any unnecessary spatial separation between the multipliers' components. Finally, as the CoreGen Multiplier is already buffered, the now redundant registers were removed.

The registers were then replaced by channel communication as detailed in Section 4.2, to give a further improvement in performance.

	Gate Count	No of Slices	Clock Speed (MHz)
Stage 1: Mean Vector and Covariance Matrix	18,288	561	33.37
Stage 3: Data Normalisation and Eigen Mapping	13,402	397	13.66
Stage 1 & Stage 3	30,567	909	13.28

Table 2 Initial place-and-route statistics

Step	Description	Gate Count	No of Slices	Clock Speed (MHz)
1	Pipeline registers at boundary: DataManager-Normaliser	30,681	951	16.248
2	Pipeline registers at boundary: Normaliser-EigenMapper	30,823	953	16.925
3	Pipeline registers at boundary: Eigenmapper-ResultStorage	31,087	986	17.511
4	Replace distributed RAM with BlockSelect RAM	44,321	967	10.581
5	Pipeline registers added to EigenMapper multiplier	31,287	1,001	26.773
6	Replace Handel-C multiplier with Xilinx CoreGen multiplier	31,632	972	28.373
7	Remove pipeline registers, use Xilinx CoreGen multiplier	31,432	953	30.418

Table 4 Optimisation steps with place-and-route statistics

5.2 Scaling to full size

Table 5 records the results when the pilot KLT pipeline consisting of first and third stages was replicated *n* times. In these measurements, no account is taken of coordination of data I/O, as only the first pipeline in

every replication set is properly connected to memory banks. For the same reason, the clock frequency should only be taken to indicate an approximate speed of 20 MHz. The 16-replication build failed during place-and-route; the tool was unable to find sufficient paths to route all of the signals.

What the results do show is that the number of replications is limited to twelve. This represents an image of just 240 bytes, whereas a representative image is either 128×128 or 512×512 pixels. To accommodate images of these dimensions, many times what can be accommodated in one pass on the Virtex, it is necessary to pass repeated streams of data many times through the KLT engine, accumulating results either on the host processor or on the FPGA, before the eigenvector calculation is performed. This issue is returned to in Section 5.3.

When the mean vector and covariance totals are accumulated on the FPGA, then the data width will be larger, respectively increasing from 11 to 26 bits and 17 to 34 bits for a $6 \times 512 \times 512$ dataset. Table 6 records the results of accounting for the increase in dynamic range, establishing that the number of replications that can be supported, in fact, remains the same.

Replications (*n*)	Gate count	Slices	Capacity Used	Speed
1	31,608	1,016	8%	27.243
8	204,942	7,582	62%	23.974
10	253,096	9,313	76%	21.560
12	302,738	11,211	91%	19.529
16 (failed)	400,804	12,286	100%	14.658

Table 5 FPGA usage from place-and-route statistics

Replications (*n*)	Xilinx Gates	Slices	Capacity Used	Speed (MHz)
1	39,705	1,077	9%	25.295
8	270,126	8,065	66%	22.444
10	334,306	9,920	80%	22.077
12	400,334	11,921	97%	20.115

Table 6 FPGA usage with increased data widths

5.3 Calculations on the host

The complete transform was modelled in C++ to make use of the SystemC simulation kernel, giving the opportunity to make comparative timings in software and hardware. The calculation of eigenvectors on the

host used the standard Jacobi iterative method of calculation [34]. The RTE (Section 2.5) with two tasks instantiated occupied 1,438 gates or 91 slices, being 1% of the Virtex-I capacity, and was clocked in isolation at 67.8 MHz. Therefore, the RTE had an insignificant impact on the results of this Section and Section 5.4.

The timing for a complete KLT on a Pentium II, running at 220 MHz, and a Pentium IV, running at 1.7 GHz, was respectively 220 μs and 45 μs. However, it should be emphasized that these timings were for the very small matrix of the pilot study and not a realistically sized image ensemble. Given the strictures on the limited I/O capability of the RC-1000 board (Section 1) it should not come as a surprise that partitioning the software between host and board resulted in a timing of 3,770 μs, when the host was the Pentium II. This is a considerable slow-down, but is simply because the overheads in loading the FPGA with data exceed the computation cost when only limited data parallelism is exploitable. Section 5.4 shows that when a realistically sized image ensemble is employed, the situation is effectively reversed.

The accuracy of the timings in this Section and Section 5.4 was established by averaging over many thousands of tests. The MS Windows High-Resolution Timer API, which in turn examines a hardware register on the Pentium, was called to make the timings. Notice also that there is an extra one-off cost of 117 ms, due to loading the complete FPGA configuration bitfile at about 30 MHz (whether all LUTs are configured or not).

5.4 Overall timings

Timings for a realistically sized image ensemble are shown in Table 7. Taking absolute timings for the first stage for 12 iterations of the pilot design (Section 5.2) resulted in a time of 54 μs and for the third stage 38 μs. The first stage is dominant, and scaling this result to the full image ensemble, requiring 1,093 passes of data gives the result in Table 7. Not included in Table 7 is the comparatively small time taken to find the transform kernel on a microprocessor, which can be surmised from the results in Section 5.3.

Description	Timing (ms)
Software on Pentium IV (1.7 GHz)	301.0
Scaling for Virtex-1 FPGA (20 MHz)	59.2

Table 7 KLT execution times for a $6 \times 512 \times 512$ pixel image ensemble

As mentioned in Section 4.1, by adding an additional MAC unit the timing of the first stage could be reduced so that it approximately balanced the shorter transform stage time. This action was not taken, as

Table 6 reveals that 97% of the Virtex-1 capacity is already in use without the per pipeline additional MAC unit. However, notional calculations, made from clock-level simulations, show that the latency across stage one of the pipeline at 20 MHz clock rate would be reduced to 36.3 ms, representing a stage one per-row latency of 664 cycles or 33.2 μs. As the equivalent timing on the third stage is 39.96 ms, there is no point in reducing the first stage timing further, as the transform stage is now the limiting factor. Therefore, if the Virtex-1 had sufficient room then the scaling for the Virtex-1 would be 40.0 ms in Table 7.

The results in Table 7 represent the times for what is effectively a first-generation Virtex device. The Virtex-II family has a maximum clock speed of 400 MHz, compared with 100 MHz for the Virtex-1 used in our experiments. Hence the execution times for the Virtex-1 reported here represent a lower bound on the performance of these devices, which can be expected to scale with each new generation device in a similar way to PC performance.

The most significant additional architectural feature of the Virtex-II family, from the point-of-view of a KLT type algorithm, is likely to be the presence of between 40 and 192 hardware multipliers respectively for the 1 M and 10 M variety of Virtex-II. In contrast, the Xilinx CoreGen multipliers (refer back to Table 4), are CLB-based and, hence, restricted to the logic available on a CLB and not custom multipliers.

5.5 Design of a streaming I/O system

The scaled timing for the FPGA is an order of magnitude faster than the Pentium IV, but this pre-supposes that there is a streaming I/O sub-system to sustain the Virtex data requirements. In fact, a streaming sub-system has *already* been constructed for a different purpose [35] with four 8 Mb $\times 16$ bits access width for Virtex 1 modules. This section presents a design for the Virtex-II Pro, allowing an on-chip host processor to be placed within the pipeline. In the RC1000-PP board, access to memory is limited by the clock speed of the FPGA, whereas a stream sub-system can independently control memory access. In general, dual-ported selectRAM blocks act as a buffer, so that data can be accumulated before being read-out for consumption by the FPGA engine. This requires two clock domains. In fact, up to 12 parameterizable digital clock managers with maximum frequency of 400 MHz are available on the Virtex-II Pro for this purpose (though these are not directly programmable through Handel-C).

Multiple clock domains are necessary, as otherwise there is insufficient time for a streaming I/O system to update all KLT pipeline stage data buffers before the next row has to be processed, some 50 clock cycles later. 50 cycles per frame approximately equates to a four cycle update window per KLT pipeline. Increasing the DRAM controller clock speed to (say) 120 MHz extends the update window to 24 clock

cycles, which is sufficient to allow one write to DRAM memory from an external source inputting the next image set, and one read to each pipeline stage. This update procedure is repeated for each of the six row data elements giving a total update time of 18 cycles, well within the 24 clock cycles budget allowed.

Figure 13 shows our system design, including an on-chip PowerPC core (microprocessor) in stage two, which makes fixed-point calculations. The system consists of the streaming I/O and the KLT pipeline sub-system, respectively running at 120 MHz and 20 MHz. The I/O controller coordinates DRAM read/write activity and row data transfers from DRAM to KLT pipeline buffers. The summation unit accumulates the sample mean (sm) and the covariance (cv) partial results, generated by each of the twelve KLT pipeline units. The summation unit then interrupts the microprocessor to start the eigen calculator. Additional system controller logic and further DRAM memory is needed to store the transformed data output.

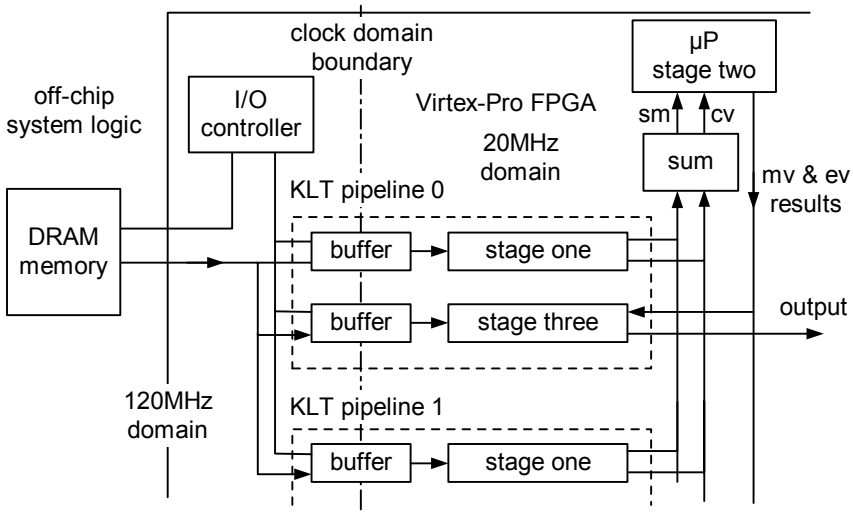

Figure 13 Virtex-II Pro I/O subsystem showing KLT pipeline

The design operated at 20 MHz allows six updates for reasonable DRAM access speeds of 120 MHz[1]. In fact, there is a greater latency because the KLT pipeline takes about fifty 20 MHz clock cycles per stage, allowing about 300 DRAM accesses before a new set of data is required. The data width for one stage of the KLT is 6×12 bytes. Each of the twelve data streams requires a 48-bit width, typically formed from three 16-bit wide DRAM memory modules. A write of the original images from an external source and two reads for each of the KLT stages is required for each of the twelve data streams, making 36 accesses in all, certainly allowing a further set of writes if the processed image ensemble were to be written to sufficiently large memory modules.

In fact, on the Virtex-II, a limitation is likely to be the availability of selectRAM blocks as the maximum

[1] Micron Inc.'s smallest Synchronous DRAM IM16A1 with minimum size 16 Mb $\times 16$ bits has clock speeds 125, 143, 166 MHz.

port width per block is 32 bits, requiring two blocks per stream, and hence 24 blocks per stage, 48 in all. The XC2V1500 with 1.5 M system gates, has exactly 48 selectRAM blocks, with a maximum of 192 blocks on the largest device in the family. Though it is possible that additionally or instead distributed RAM could be used in combination with external buffering, this would present a typical developer's trade-off. If Rocket I/O transceivers were to be used, then the transceiver includes deserializer and decoder, while buffering would take place in a similar way to the use of selectRAM. An FPGA can deliver four eight bit or encoded 10 bits for transmission by a transceiver.

6 Conclusions

For multi-spectral and multi-resolution imagery, the Karhunen-Loève transform is an indispensable tool for which no `fast' algorithm exists. Until the advent of platform FPGAs, and since the demise of SIMD architectures such as the ICL DAP, no cost-effective architecture has existed to perform parallel processing with the KLT. Indeed, in common with other algorithms more suitable for fine-grained processing, real-time processing in the recent past took place on medium- and coarse-grained parallel architectures such as those based on the Inmos transputer. Unlike some of these algorithms, the KLT is still difficult to parallelize because of discontinuities in the scale and type of processing across the three stages that the algorithm can be divided into.

This paper describes the development of a prototype KLT engine targeted at such devices but presently implemented on a PC-based development board. The findings of this paper in relation to developing parallel applications on platform FPGAs are:

- The development board can be used to effectively estimate the performance of an application on a target system if a pilot design is first completed and then scaled up.
- Such development boards, such as the one used in this study, may not be suitable for full-scale software acceleration because of I/O restrictions.
- A hardware compiler can reduce the time to produce a pilot design, if used in conjunction with a high-level modelling tool, such as the SystemC set of libraries.
- Though a hardware compiler allows a software-only approach to hardware design, it can also act as a form of hardware shorthand for the experienced hardware designer.

In respect to developing parallel signal processing algorithms, the paper reports that

- The Virtex family of platform FPGAs, running at low clock speeds, compete favourably with high-end general-purpose microprocessors, if sufficient data parallelism can be exploited.
- The addition of an embedded RISC core would significantly enhance the range of such algorithms that are realistic on a platform FPGA. The same may apply to other hybrid architectures that include a reconfigurable element.

- Even with a large FPGA, there may be insufficient gates to allow one pass image processing. Therefore, it may be necessary to multiply pass image segments through such a device.
- Multiple clock domains on the later Virtex-II device, along with on-chip processor cores allow an I/O sub-system to be designed.

Future work is to include our single algorithm parallelisation within a multi-algorithm application such as multi-spectral codecs that include a KLT as a component part. We should also consider whether our development methodology could be exploited to parallelise a range of signal-processing algorithms.

Acknowledgements

This work was carried out with assistance from an EPSRC/DERA CASE studentship 9930329X.

References

[1] Karhunen, K.: Ueber lineare Methoden in der Wahrscheinlichtskeitsrechnung. Annals Academy Science Fennicae Series A. I. Vol. 37, 1947.

[2] Loève, M. M.: Fonctions Aleatories de Seconde Ordre, In Process Stochastiques et Movement Brownien (P. Levt, ed.), Hermann, Paris, 1948.

[3] Savoji, M. H., and Wilkinson, G. G.: Karhunen-Loeve on an Array Processor - Applications to Multi-spectral and Single Band Images, IEE Conference Publications 214, 225-229, 1982.

[4] Hockey, R. W. and Jesshope, C. R.: *Parallel Computers 2*, Adam Hilger, Bristol, 1988.

[5] Wakerly, J. F.: *Digital Design: Principles and Practice*, 3rd edition, Prentice Hall, 2000.

[6] Virtex™ 2.5 V Programmable Gate Arrays Datasheet, Xilinx Inc., San Jose, CA., 2001.

[7] 'Virtex-II Pro™ Platform FPGA Handbook', Xilinx Inc., San Jose, CA., 2002

[8] Ghanbari, M.: *Standard Codecs: Image Compression to Advanced Video Coding*, IEE, London, 2003.

[9] Vaisey, J., Barlaud, M., and Antonini, M.: Multispectral Image Coding using Lattice VQ and the Wavelet Transform, Paper 712, IEEE International Conference on Image Processing, 1998.

[10] Kittler, J. and Duff, M. J. B. (editors), *Image Processing System Architectures*, Wiley, 1985.

[11] Bowen, M.: Handel-C Language Reference Manual, Celoxica Inc., Didcot, UK, 2001

[12] Chang, K. C.: *Digital Systems Design with VHDL and Synthesis: An Integrated Approach*, IEEE Computer Society publications, 1999.

[13] Sweeney, C. and Blyth, B.: RC1000-PP Hardware Reference Manual, Celoxica Inc., Didcot, UK, 2001.

[14] Weinhardt, M. and Luk, W.: Task-parallel Programming of Reconfigurable Systems, In Field-Programmable Logic and Applications, FPL2001, 172-181, LNCS 2147, Springer, Berlin, 2001.

[15] Schreiber, R.: Solving Eigenvalue and Singular Value Problems on an Undersized Systolic Array, SIAM Journal of Scientific and Statistical Computing, 7, 441-451, 1986.

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

[17] Hotelling, H.: Analysis of A Complex of Statistical Variables into Principal Components, Journal of Educational Psychology, 24: 417-441 and 498-520, 1933.

[18] Ready, P. J. and Wintz, P. A.: Information Extraction, SNR Improvement and Data Compression in Multispectral Images, IEEE Transaction on Communications, 31(10), 1123-1130, 1973.

[19] Kirby, M. and Sirovich, L.: Application of the Karhunen- Loève Procedure for the Characterization of Human Faces, IEEE Transactions on Pattern Analysis and Machine Intelligence, 12(1), 103-105, 1990.

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

[21] Gersho, A., and Gray, R. M.: *Vector Quantization and Signal Compression*, Academic Press, 1992.

[22] Jain, A. K.: A Fast Karhunen-Loeve Transform for a Class of Random Processes, IEEE Transactions on Communications, 24, 1023-1029, 1976.

[23] Percival, D. J.: Compressed Representation of a Backscatter Ionogram Database using Karhunen- Loève Techniques, Image Processing and its Applications, 574-578, IEE Conference Publications 410, 1995.

[24] Oja, E.: Principal components, Minor Components, and Linear Neural Networks, Neural Networks, 5, 922-935, 1992.

[25] Page, I.: 'Constructing Hardware-Software from a Single Description', Journal of VLSI Signal Processing, 12**,** pp. 87-107, 1996.

[26] Spivey, M., Page, I., and Luk, W.: How to Program in Handel, Oxford University Hardware Compilation Unit, 1995.

[27] *Occam 2 Reference Manual*, Inmos Ltd., Prentice Hall, New York, 1988.

[28] Hoare, C. A. R.: 'Communicating Sequential Processes', Communications of the ACM, 21(8), 666-677, 1978.

[29] Hilderink, G., Broenink, J., Vervoort, W., and Bakkers, A.,: Communicating Java Threads, 20th WoTUG conference, 1997, pp. 48-76

[30] Fleury, M., Self, R. P. , and Downton, A. C.: Hardware Compilation for Software Engineers: An ATM Example, IEE Proceedings Software, 148(1), 31-42, 2001.

[31] Gerlach J. and Rosenstiel, W.: System Level Design using the SystemC Modeling Platform, SDL'2000.

[32] Jennings, A. and McKeown, J. J.: *Matrix Computation*, 2nd edition, Wiley, Chichester, UK, 1992.

[33] Gate Count Capacity Metrics for FPGAs, Application note, Xilinx Inc., 1997.

[34] Press, W. H., Flannery, B. P., Teukolsky, S. A., and Vetttering, W. T.: *Numerical Recipes in C*, CUP, Cambridge, UK, 1988.

[35] Fisher, C., Rennie, K., Xing, G., Berg, G. B., Bolding, K., Naegle, J., Parshall, D., Protnov, D., Sulejmanpasic, A. and Ebeling, C., An Emulator for Exploring RaPiD Configurable Computing Architectures, In Field-Programmable Logic and Applications, FPL2001, 17-26, LNCS 2147, Springer, Berlin, 2001.