Pipelined Parallelisation of Automatic Face Inspection

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

Dept. of Electronic Systems Engineering, University of Essex,

Wivenhoe Park, Colchester, CO4 3SQ, U.K

tel: +44 - 1206 - 872817

fax: +44 - 1206 - 872900

e-mail fleum@essex.ac.uk

Abstract

A parallel pipeline is shown to be a natural method of speeding up a typical computer vision application, face inspection using 'Eigenfaces'. Faces within a stream of video images are continuously surveyed in a manner akin to a 'conveyor belt' inspection process. The parallelisation is a new exemplar of a scheme for the rapid prototyping of large-scale, multi-algorithm applications suitable for transfer to a message-passing multicomputer. A general solution, pipelined processor farms (PPF), is preferred to a customised solution. This paper gives details of the software tools and software-engineering methods employed to tackle this class of problem.

Keywords: parallel pipeline, data farm, face identification, performance analysis

1 Introduction

Many inspection applications have a generic pipelined structure. A parallel pipeline has one or more of its stages parallelised in addition to the inherent pseudo-parallelism arising from processing different data at the same time across the stages of the pipeline. The present paper describes a parallel pipeline solution for a well-known method of face inspection, 'Eigenfaces' [15,

36], capable of separating a face from a video scene and matching that face against a set of candidate faces, taken from a pre-compiled database of faces. The variant of Eigenfaces described in [26, 22, 23] was the subject of the parallelisation. The aim of the parallelisation was to speed-up an existing sequential processing pipeline originating from MIT Media laboratory in order to reduce the time taken to identify a face in a given database. It is assumed that there is a continuous sequence of video images to be matched to the database as may typically occur in visual surveillance applications.

We have previously employed parallel pipelines to speed up a number of applications including a handwritten postcode inspection application [2] by the *same* development method described in the present paper. In handwritten postcode inspection, a conveyor belt moves envelopes marked with address and postcode alongside the computer system. Once the digitised envelope image is within the computer system, the address and postcode are extracted and a number of candidate solutions for the postcode are found. The objectives of the parallelisation were: to reach a throughput of 10 matched postcodes/s; and to ensure that no one identification should have a latency longer than 8s, in order to keep up with the conveyor belt. There was also an output ordering constraint. The envelopes are stamped with phosphorescent dots, indicating the postcode, after which which they move on to the next stage of processing. There were three stages within the pipeline: pre-processing of the digital images; classification of the postcode characters; and matching against a database of addresses. The first two stages were parallelised by means of data farming on a set of processors. A data farm [37] is a parallel programming paradigm consisting of a data farmer process which manages the distribution of data to be processed and a set of worker processes. Within a parallel pipeline, the data farmer may also carry out any centralised processing before passing on the data to the next stage. In the handwritten postcode application, the data were the extracted postcode characters.

Our target parallel processor for Eigenfaces was a medium-grained message-passing machine, the Transtech Paramid [35], equipped with eight i860 superscalar microprocessors [1]. Transputer communication coprocessors (valency four) link the i860s in an electronically re-

configurable network. The Paramid comes into the class of 'lowly' parallel computers [27], being a response to the exigencies of the market-place. Other 'lowly' parallel computers include networks or clusters of workstations with the same distributed-memory architecture. A software template based upon a real-time programming model [8] was employed and adapted for the Paramid [12]. The structure enabled the chore of 'plumbing', routine connection of message-passing channels [38], to be by-passed as sequential sections of code could be placed within an existing template skeleton. For example, over 40 different types of messages were passed between the processes making up the eigenfaces application.

The objective was to reduce the sequential eigenfaces pipeline throughput of about 2 mins/image on a SPARCstation 2 to less than 10s/image on the Paramid. In the first instance, this would enable evaluation and testing of the prototype software in a realistic setting. However, the processing structure is intended to be incrementally scalable so that with appropriate hardware a video-rate solution could be achieved.

As the eigenfaces application consists of *circa* 8700 lines of code, including options but not library code, considerable logistic organisation is required (for a single programmer) to effect the parallelisation. The inspection application involves multiple algorithms, with 183 functions called in a typical run. It has been remarked [5] that the parallel application literature abounds with bespoke solutions to single algorithm parallelisations of engineering problems but, in contrast, typical parallel engineering applications comprise many component algorithms. For example, in this case study, the processing time spent on locating a face within an image and normalising it dominates the time spent on face identification (Section 2.4) even though it is the latter algorithm that has attracted attention in the literature. To regularly provide solutions to complete systems like the eigenfaces application requires the systematic use of software tools and a generic design strategy.

The approach taken in this paper is to use a generalised method of system design, pipelined processor farms (PPF) [5]. The aim of this methodology is to introduce flexibility and reduce software development costs by imposing a generic solution [6, 3]. PPF is suited to the parallelisation of legacy code, or code written on workstations supporting a sequential pro-

gramming model. The algorithmic developers need not be the same persons responsible for parallelisation. In other words, the code may be unfamiliar requiring an initial exploratory stage. PPF is built around a design pattern consisting of a pipeline of data farms, Figure 1. A pipeline with a single backplane [20] has the advantage that it can map onto a variety of architectures. In some cases, the data farmers are connected by a high bandwidth bus. Sequential bottlenecks are masked by single processor stages. The deployment of worker processes is determined by the relative per-stage workload, the desired global throughput, and the desired pipeline traversal latency. Flexibility is required so that late algorithmic changes can be introduced. By partitioning the application into independent pipeline stages it is possible to introduce such changes without algorithmic side effects. By choosing a data farm solution, if increased processing is needed as a result of the changes, the number of worker processes can potentially be increased. In fact, we were aware of a possible algorithmic change to the eigenfaces application, whereby not one but several candidate heads within the video image [36] could be passed to the second stage of our pipeline which would result in a four-fold increase in processing time on a workstation.

The remainder of the paper is organised thus. In Section 2, the initial design stage, which involves exploration of the algorithm, is described. Since the parallelisation took place in incremental stages, Section 3 describes the first pipeline stage and Section 4 discusses subsequent stages. Section 4 also discusses results of the paralllelisation. Finally, Section 5 summarises and draws some conclusions.

2 Analysis of the Algorithm

Though parallelisation of face recognition has taken place before, a pipeline structure has not been employed. This may have been because the motive for previous parallelisations was to speed up algorithmic development and not eventually to provide a real-time processing pipeline. The sequential version of Eigenfaces was constructed as a pipeline which immediately makes it a candidate for real-time implementation. The regular structure present in the

head-search and feature-matching processing is suitable for data farming. The head-search processing is particularly computationally demanding, which implies a favourable compute-communicate granularity. There are no feedback loops which would otherwise impose a synchronous constraint on any pipeline.

2.1 Parallel Implementation of Face Inspection Algorithms

Parallelisation of face inspection software has previously been described in [18, 17], which is an example of a graph-based approach on hardware with a similar architecture to ours. Three independent processing tasks were identified with differing compute/communicate ratios, namely fast Fourier transform of image data; computation of Gabor filters in the frequency domain to form 'objects'; and comparison with stored 'objects' held as sparse graphs. A processing farm was used but the vital step of forming a pipeline was apparently not made. The implementation of the graph-based method is reported [17] to take 25s with optimised code to make a match between face images of size 128×128 pixels (captured in controlled circumstances) and one of 87 stored 'objects'. 21 active 20 MHz transputers were arranged in a tree topology with two other transputers providing system support. A more recent and enhanced version of elastic graph matching is described in [39], though without implementation details.

2.2 The Eigenfaces Algorithm

In the eigenfaces method, faces are projected into eigen-space by a Karhunen-Loève transform [24, 7], avoiding reliance on semantic knowledge.[1] Since identification appears to depend on good intensity correlation between images, substantial pre-processing is needed [19]. Changes in scale are a principal cause of error for this method if no correction were to be made [36]. Allowance for lighting and contrast variation [22] and affine transformation of the head position are also made. Changes in facial expression and time variation in facial appearance

[1] Notice that the KLT approach has been shown to have some of the aspects of human vision such as ability to distinguish between individuals and genders, and form affinities [25].

are not addressed in the variant of Eigenfaces parallelized herein.

2.3 The Eigenfaces Pipeline

Though a sequential software pipeline already existed it is not guaranteed that the pipeline when parallelized would be balanced, since this depends on the computational load within each pipeline stage. Partitioning points within the application are identified by a static analysis (omitting communication and synchronisation costs) guided by the designer's understanding of the sequential application software architecture [4].

In the static analysis, a profiler is employed to find the percentage times taken to execute the functions within the application. (Eigenfaces is written in the 'C' programming language.) A function call graph is analysed to find branches which take up a high proportion of the total processing time in a top-down manner. It will not be worthwhile spending development time on parallelizing short-lived portions of the code. (Code only executed at initialisation is also passed over in pipeline systems.) The functions are then inspected to find likely sources of parallelism. Loop (data) parallelism is common but algorithmic parallelism can also sometimes be exploited. Synchronisation constraints should also be identified. For example, the eigenfaces application has no feedback paths between images (as may be the case with, for example, image-coding algorithms [3]) which would otherwise impose restrictions on the achievable speed-up.

The bulk of the processing (the first stage) is taken up with finding a head within an image scene at different positions. (The candidate head after normalisation by the 'average' face from the existing database is measured against its projection onto the database eigenspace [36] by a maximum likelihood (ML) estimator. This procedure is designed to screen out objects that are close to an individual eigenface but not close to the class of heads. Only ten eigenvectors were used at this stage. A computational formula reducing the disparity error calculation to a set of correlations is employed. The possibility that a person is present in an image scene can be ascertained by prior motion detection using spatiotemporal filtering.) In tests, the grey-scale scene was sized 512×512 pixels. Candidate heads, once located, are

normalised to size 128×128 pixels. Since the size of the head is not known, an error map is compiled for candidate heads at 21 different possible scales.[2]

A second stage is feature detection. Once the best position and scale for a head is found, four facial features are sought within appropriate parts of the face. The features should conform to the known facial geometry (e.g. nose below eyes and mouth below nose). The feature detection stage also has some scope for parallelisation as each feature is sought independently from the others. The feature detection employs five eigenvectors in a similar procedure to the head-search. The eventual pipeline did not utilise this possibility in the interests of balancing the load throughout the pipeline. The location of the facial features serves to parameterise an affine warp which brings the head into standard alignment with the faces in the database. The head is also cropped to produce a face.

The third and final stage, also had the potential to be parallelized: the normalised face is projected into eigen-space with 100 coefficients, whence the nearest three matching images are output.

Because of the dominance of the scale search stage of processing it may be appropriate to consider an alternative scale (and rotation) invariant transform of image and head, such as the Mellin transform, before performing the correlation step. A parallelisation of the log-polar transform, an essential step in the Mellin transform, is available in [11].

2.4 Profiling the Eigenfaces Sequential Pipeline

When approaching unfamiliar software with a view to parallelisation, a profiler such as `gprof` is [13] a convenient tool. This will remain the case while the structured programming approach is common. Provided the time spent within a function is not overly affected by the sampling rate of the profiler [29], any profiler is adequate. Extensive recursion, which is also a problem

[2]Because of the dominance of the scale search stage of processing a suggestion is that it may be appropriate to consider a scale (and rotation) invariant transform of image and head, such as the Mellin transform, before performing the correlation step. A parallelisation of the log-polar transform, an essential step in the Mellin transform, is available in [11].

for profilers that sample the program counter, is not expected within our class of problem. However, because a visual display is provided and because object-code instrumentation is used (with a view to greater accuracy by a machine cycle-counting method of profiling), we have preferred a recent type of profiler, `Quantify` [32]. `Quantify` timings are independent of the system load at run time though data-dependent. The ratio (not absolute value) of timings between functions is needed to effect a partition.

A snap-shot of the function call-graph of the application generated by `Quantify` is shown in Figure 2. The top twenty function display is shown for clarity in Figure 2, though `Quantify` can display the top hundred descendants of any function or call structure to arbitrary detail. The thickness of the lines between function names indicates the number of calls to a function from all other functions. The graph can be annotated with the number of calls, though the time spent in each function is shown in Figure 2. The initial head-search function dominates (`ms-search`) the sequential pipeline stemming from the function `pipe`. This function and its descendants were to make up the first stage of the parallel pipeline. The `eig-search` function was to form the basis of the second stage in the parallel pipeline. The call graph also allows analysis of the interconnection structure prior to decomposition. For example, it can be seen that two functions, `mahalanobis` and `affine-warp`, are shared between the two computational branches `ms-search` and `eig-search` requiring separate copies of these functions within each of the first two stages. The functions `load-eigen` and `scanf` are called at initiation time and therefore can be discounted from the steady-state pipeline behaviour.

Table 1 records the sequential execution times on a SPARCstation 2, after loading data files (15s). The ratio of these times for balancing a pipeline is more important than the absolute values, since the ratios are more stable. Improvements in time will come from appropriately fast processors, as we have already found when porting the handwritten postcode recognition application from a transputer-based machine (the Meiko computer surface) to the Paramid [2]. First-order projections of performance on alternative processors (DEC Alphas) have also been made by extrapolating from a set of benchmarks using a two-parameter model of performance [34].

Provided the same software structure is employed little change is needed to the code. This naturally requires a structure which will port between different machines. In [9], the data-farm template is transferred from the Paramid, to a network of workstations running the socket API with SunOs 4.1 Lightweight Processes for multithreading, and to the VxWorks real-time executive. A Java implementation using RMI of the data farm template occurs in [34]. An example of the template in use to construct parallel pipelines on the VxWorks system, with another version on the Paramid for the same application (KLT transform) is described in [7]. Multithreading is required within the template to allow a rapid response to communication events. An access control mechanism (usually counting semaphores) is needed to control buffering, which is placed at the input and output of worker processes and between stages of the pipeline. An efficient and fair demultiplexing structure (for example adaptation of the socket API `select`) is needed by the farmer to accept returning work. A means of broadcasting from the farmer to its workers to supply common data is the final underlying facility needed to construct the template, other than the usual message-passing arrangements.

3 Preliminary Parallelisation

3.1 Porting the Application to the Target Processor Architecture

The first step in the logistic exercise involved in porting the code was to check the correct behaviour of the application on Sun workstations. The libraries were then reconstructed for use on a single i860 processor (since the Paramid can also act as a throughput machine).

It was found that the application did not work on the i860 without a small modification. This appeared to be due to array-bound corruption, which sometimes emerges when transferring from a big- to a little-endian machine. Passing the `Purify` memory debugger [32] over the original code indicated the problem area. Some tidying-up of code at this stage might also be deemed necessary to achieve clarity. For instance, data pointers may be passed to a function which never uses the data, though at some stage in the earlier software development process the data were used.

Five images (from within the database) formed the test set. They were correctly matched to the same images held in database form. The behaviour on the i860 was identical to the Suns except that, for image five, the second and third candidate match images differed. The head positions and scales in the five test cases were the same as for the Sun version. Checking the behaviour of the i860 version was not a principal aim of the parallelisation, except to ensure no major aberrations occurred in the porting process. In fact, fidelity to the original code is preserved as closely as possible in order to facilitate algorithmic changes based on the sequential version.

3.2 Parallelizing the First Pipeline Stage

The second step was to form one farm, Figure 3a, in which the first, computationally dominant, head-matching stage was parallelized, the remaining sequential code being performed centrally while the farm is stalled. The i860, being run in single-threaded mode, must change from data farmer to sequential processor, shown by the dashed circle in Figure 3a. Table 2 gives the recorded execution times, excluding the initial file loading time. "Farm 6" in Table 2 refers to six worker processors and one data-farmer. The worker processors were arranged as a binary tree. The nominal clock speed of the i860-XP is 50 MHz (and the transputer runs at 30 MHz). Using the features of the i860 Portland C optimising compiler [30] considerably improved the performance. Level four optimisation principally vectorises the code.[3]

However, [31] reports order of magnitude degradation of performance between hand-coded routines and code produced by two i860 Fortran optimising compilers, either due to memory access patterns or due to failure to pipeline floating-point instructions. True performance can be 5-8% of the manufacturer's peak figures.[4] This suggests a future route to further performance improvements: hand-coding of inner loops, e.g. [16], which was however avoided

[3]Other options subsumed in level 4 introduce pipelining, loop unrolling and function inlining *inter alia*. Debugging code was turned off.

[4]These results are replicated on a range of recent RISC machines once the stages of the caching system are exhausted [33].

so as to retain compatibility with the original sequential code.

Figure 4 shows the proportion of time spent in the eigenfaces parallel section compared to the overall time. The proportion of time in the head-inspection stage is larger than the original static times suggest. It is apparent that beyond seven processor modules the application speed-up is starting to saturate. In order to preserve fidelity to the original tests 21 scales were farmed out. Purely from the point of view of sharing out work, one would want a larger number of scales, a number for which the number of worker processes was an exact divisor. The latter requirement reduces the proportion of time spent idling while waiting for the processing to finish. In fact, depending on the application, the range of scales will vary; for example: when heads are captured as 'mugshots'; when images are taken by a door-entry system; and when external surveillance cameras are employed. Again, this emphasises the need for flexibility in the parallel implementation.

Most of the data I/O, which is considerable as it involves loading of face and feature databases, takes place at the start-up phase of the pipeline. For computational tests a reduced database of 60 known faces was searched[5] though the eigenfaces basis set had dimension 100. For long runs such overhead would be amortised over the lifetime of the pipeline. The original and scaled images must be broadcast during each processing round but otherwise communication is nugatory.

3.3 Discussion

When a medium- and large-scale (over 10,000 lines of code) sequential applications are developed the application is frequently split into modules to reduce compilation time. Notice that when approaching unfamiliar code, to see the relationship between functions in terms of parameter-passing it is more convenient to merge all relevant function modules within one file. In general, the principal difficulty in parallelizing the code was identification of the data that had to be communicated. The problem was exacerbated by the quantity of data and by

[5] A 99% rate over 155 individuals on the FERET database for the variant of the system used by us is reported in [21]. Encouraging results for the 1996 MIT system are also given in [28].

the use of Fortran-style indexing in some places and 'C'-style pointers in others.

4 Extension to Three Farms

4.1 Introduction of the Second and Third Farms

A second farm for feature matching was constructed with the farmer process on a transputer and the workers on i860s. The basis of the parallelisation is the matching of features within the normalised face image that is passed on from stage one. The face image therefore has to be broadcast to the worker processes at the onset of processing. Feature matching against an average feature taken from the database again takes place in eigenspace. Due to the limited number of facial features, the potential for data farming is restricted. However, the parallelisation went ahead in part because, if multiple candidate heads were later used to achieve greater identification accuracy, the parallel structure would already exist. Were this extended form of processing to be introduced performance scaling of the second stage could take place.

In Figure 3 b), the second farmer process, hosted on an i860, is shown sharing a processor module with a worker process, hosted by a transputer. The two processes communicate by overlapped memory and not by a transputer link. The first farm solely used i860s since the farmer originally had not only to pass on data but also to perform processing. Provided there is enough memory (4M bytes on the transputer as opposed to 16 Mbytes on the i860), then it may be possible to free another processor on the Paramid by also moving the farmer on farm one onto a transputer, which otherwise has no hardware support for memory management. To verify the behaviour of the second farm, the output was fed back to the first farm where the remaining processing was completed. When the second farm was seen to work the remaining processing was transferred to a third farm, which actually has only one 'worker' on an i860. In other words, the granularity of the pipeline as a whole did not justify partitioning between farmer and workers in the third farm.

4.2 Balancing the Pipeline of Processor Farms

A three-farm pipeline was tested in the processor ratio 5:2:1 and then 6:1:1. The topology for the second pipeline arrangement is shown in Figure 3 c). A test was made processing 25 images (the five test images repeated), recording pipeline traversal latency and throughput, Figure 5. The latency of an image passing through the 5:2:1 pipeline was found to have increased from 11s for the single farm to 11.96s (mean of 25 images). However, the throughput increased from an image every 11s to an image every 9.24s. The 6:1:1 pipeline resulted in a further increase in latency (12.80s/image) but a similar increase in throughput: an image every 7.48s. Since the first farm is not in saturation with 5 processors (and 4 workers) increasing to 6 processors improves the speed that images emerge from the first farm, freeing the farmer to load another image. The second farm will be depleted by one worker which consequently increases the latency. Clearly on a larger machine one would increase the number of processors in farm two as well as farm one, when the latency would be reduced along with the throughput. The performance figures depend on the speed of I/O when loading from disk across a SCSI bus and on the system load at the time of the tests. I/O time for the original image, which is variable, is included to give realistic timing figures. If more images were tested the start-up time of the pipeline would be shared over a longer time-span, again improving the results.

4.3 Further improvement from re-ordering of processing

The processing scales, which vary from 0.152107 (scale 1) to 0.4000 (scale 21) in the supplied test file, differ on an ascending scale in the time taken to process them. `gprof` records 1s for scale 2 and 10s for scale 21. If the scales are processed from 21 downwards (i.e. in descending size) the result is an improvement in latency and throughput to 12.2s/image and an image every 6.92s respectively. The reason for this improvement over Figure 5 is that any worker left idling for work at the end of processing an image will spend less time waiting if the final scales take less time to process.

4.4 Visualisation of System Performance

The behaviour of the parallel algorithm has been captured by our event trace tool, which produces a trace-file in standardised format [40]. Since the Paramid does not have a global clock it was necessary to design a clock synchronisation sub-system [10] which imposed no more than a 5% overhead from synchronisation signals and was accurate to at least within 0.5ms. Two versions of the template exist, uninstrumented and instrumented, though provided a client's specification (in terms of throughput and latency) is met there is no obvious need to remove instrumentation.

Figure 6 is a space-time diagram from the post-mortem visualiser `ParaGraph` [14] generated from the output event trace-file. The initialisation by a hefty set of messages to farm three (processor 7) has been suppressed from the display, but the distribution of data to processors 1 to 6 is shown at the onset of the trace. Thereafter, periodic distributions of the scaled image to processor 6 and the original image to processors 1 to 5 take place (shown twice in the figure). Immediately after the original image is sent, the scales are distributed. Processor 5 from the first farm is under-utilised. Processor 6 is idle for the relatively short period between passing on its results to processor 7 and receiving the new scaled image. By adding an artificial message from processor 7 to processor 0 at the end of processor 7's period of work, it is apparent that processor 7 is idle for some time. Therefore, further improvement in performance (and cost saving) will arise by merging processor 7 with the second farm.

5 Conclusion

This paper, by way of a case study, has demonstrated a generalised method of developing software suitable for automatic visual inspection. The particular case study, Eigenfaces for face inspection, demonstrates a potentially wider application scope compared to a traditional view of machine vision. A relatively low-cost multicomputer is employed to accelerate existing sequential software code. In a production environment, the parallel machine would be hosted by a PC or workstation with attached scene-acquisition hardware.

The paper has set out the steps in incrementally forming the pipeline. Partitioning and task decomposition is assisted by high-specification profiling and memory debugging tools. The features of an optimising compiler are exploited to improve the performance of individual tasks and at some future date customised assembler routines would be incorporated. A reusable data-farm template aids rapid prototyping of the processing pipeline. The template is instrumented with event-tracing in order to automatically pin-point how the performance could be further improved.

The eigenfaces method of face inspection is a good candidate for parallelisation by a pipelined method as each stage of processing is independent. There is substantial latent parallelism within 'Eigenfaces' which we exploited by two data farms at stages one and two of our pipeline. Event tracing showed that the the final pipeline stage could be merged with the second farm so as to improve performance. The pipeline traversal latency in the implementation is still high but this will only be reduced by using more processors than were available on our research machine.

There are ways to improve the performance further on the Paramid, were this to be the final target machine. For example, an i860 could be freed in the first farm if a transputer were to host the data farmer. The absence of feedback constraints and the dominance of the head search stage means that the residual serial sections of the code can be masked within the pipeline, except that is for time taken to load images into the pipeline. (Specialist hardware would be needed to speed up the image transfer, which is available on some versions of the Paramid.)

In general, the parallelisation of the eigenfaces application has demonstrated the advantages of the PPF methodology, since the development took less than one man-month, including time spent on familiarisation with the code prior to parallelisation. Given the generalised structure of our parallel solution it is anticipated that porting to a different, faster, parallel environment would *not* pose significant problems.

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. The eigenfaces sequential software described in this paper was originally developed at the MIT Media Laboratory, Cambridge, MA, and was made available to this project through a collaborative research activity with BT laboratories, Martlesham, UK. The authors are indebted to BT Visual Applications Division for permission to publish the the experimental work on parallelizing the eigenfaces application. Opinions expressed in this paper are the authors' personal views, and should not be assumed to represent the views of BT.

References

[1] M. Atkins. Performance and the i860 microprocessor. *IEEE Micro*, pages 24–27,72–78, October 1991.

[2] A. Çuhadar, A. Downton, and M. Fleury. Structured parallel design for embedded vision systems: A case study. *Microprocessors and Microsystems*, 21:131–141, 1997.

[3] A. C. Downton. Generalised approach to parallelising image sequence coding algorithms. *IEE Proceedings Part I (Vision, Image and Signal Processing)*, 141(6):438–445, 1994.

[4] A. C. Downton. Parallelisation of embedded image, speech and signal processing applications. Technical report, BT Laboratories, Martlesham, Suffolk, UK, 1995. Report on Short-Term Fellowship.

[5] A. C. Downton, R. W. S. Tregidgo, and A. Çuhadar. Generalized parallelism for embedded vision applications. In A. Y. Zomaya, editor, *Parallel Computing: Paradigms and Applications*, pages 553–577. Thomson, London, 1996.

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

[7] M. Fleury, A. C. Downton, and A. F. Clark. Karhunen-Loève transform: An exercise in simple image-processing parallel pipelines. In *Euro-Par'97*, pages 815–819. Springer, Berlin, 1997. Lecture Notes in Computer Studies Vol. 1300.

[8] M. Fleury, 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.

[9] M. Fleury, A. C. Downton, and A. F. Clark. Constructing generic data-farm templates. *Concurrency: Practice and Experience*, 20(9):1–20, 1999.

[10] 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 Part I (Computer and Digital Techniques)*, 144(2):65–73, March 1997.

[11] M. Fleury, L. Hayat, and A. F. Clark. Parallelizing grey-scale coordinate transforms. *IEE Part I (Vision, Image, and Signal Processing)*, 142(4):207–212, August 1995.

[12] 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.

[13] S. L. Graham, P. B. Kessler, and M. K. McKusick. `gprof`: a call graph execution profiler. *ACM SIGPLAN notices*, 17(6):120–126, June 1982.

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

[15] M. Kirby and L. Sirovich. Application of the Karhunen-Loève transform procedure for the characterisation of human faces. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12(1):103–108, 1990.

[16] Kuck & Associates, Inc., 1906 Fox Drive, Champaign, IL 61820. *CLASSPACK Basic Math Library/C User's Guide 1.3*, 1993.

[17] M. Lades, J. C. Vorbrüggen, J. Buhmann, J. Lange, C. von der Marsburg, R. P. Würtz, and W. Kohen. Distortion invariant object recognition in the dynamic link architecture. *IEEE Transactions on Computers*, 42:300–311, 1993.

[18] M. Lades, J. C. Vorbrüggen, and R. P. Würtz. Recognizing faces with a transputer farm. In T. S. Durrani, W. A. Sandham, and J. J. Soraghan, editors, *Applications of Transputers 3*, volume 1, pages 148–153. IOS, Amsterdam, Holland, 1991.

[19] S. Lawrence, C. L. Giles, and A. D. Tsoi, A. C.and Back. Face recognition: A convolutional neural-network approach. *IEEE Transactions on Neural Networks*, 8(1):98–114, January 1997.

[20] S-Y Lee and J. K. Aggarwal. A system design scheduling strategy for parallel image processing. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12(2):194–204, 1990.

[21] B. Moghaddam and A. Pentland. Face recognition using view-based and modular eigenfaces. *SPIE*, 2277:12–21, 1994.

[22] B. Moghaddam and A. Pentland. Maximum likelihood detection of faces and hands. In *International Workshop on Automatic Face- and Gesture-Recognition*, pages 122–128, 1995.

[23] B. Moghaddam and A. Pentland. Probabilistic visual learning for face detection. In *International Conference on Computer Vision and Pattern Recognition*, pages 786–793, 1995.

[24] 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.

[25] A. O'Toole, H. Abdi, K. A. Deffenbacher, and D. Valentin. A perceptual learning theory of the information in faces. In T. Valentine, editor, *Cognitive and Computational Aspects of Face Recognition*, pages 159–182. Routledge, London, UK, 1995.

[26] B. Pentland, A. Moghaddam and T. Starner. View-based and modular eigenspaces for face recognition. In *IEEE Conference on Computer Vision and Pattern Recognition*, 1994. Also MIT Technical Report #TR-245.

[27] G. F. Pfister. *In Search of Clusters: the Coming Battle of Lowly Parallel Computing Systems.* Prentice-Hall, Upper Saddle River, NJ, 1995.

[28] P. J. Phillips, H. Moon, P. Rauss, and S. A. Rizvi. The FERET September 1996 database and evaluation procedure. In *First International Conference on Audio and Video-based Biometric Person Authentication*, pages 395–402. Springer, Berlin, March 1997. Lecture Notes in Computer Science 1206.

[29] C. Ponder and R. Fateman. Inaccuracies in program profilers. *Software–Practice and Experience*, 18(5):459–467, May 1988.

[30] Portland Group, 9150 SW Pioneer Court, Suite H, Wilsonville, Oregon 97070. *PGCC User's Guide*, 1992. Release 2.3.

[31] R. Pozo. *Performance Models of Parallel Architectures for Scientific Computing.* PhD thesis, University of Colorado, 1992.

[32] Pure Software Inc., 1309 South Mary Ave., Sunnyval, CA. *Quantify and Purify User's Guides*, 1992.

[33] U. Rude. Iterative algorithms in high performance architectures. In *Euro-Par'97*, pages 57–71. Springer, Berlin, 1997. Lecture Notes in Computer Science Vol. 1300.

[34] N. Sarvan, R. Durrant, M. Fleury, A. C. Downton, and A. F. Clark. Analysis prediction toolkit (APTT) for real-time image processing. In *IEE Image Processing and its Applications, IPA99*, volume 1, pages 116–121, 1999.

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

[36] M. Turk and A. Pentland. Eigenfaces for recognition. *Journal for Cognitive Neuroscience*, 3:71–86, 1991.

[37] A. S. Wagner, H. V. Skreekantaswamy, and S. T. Chanson. Performance models for the processor farm paradigm. *IEEE Transactions on Parallel and Distributed Systems*, 8(5):475–489, May 1997.

[38] G. V. Wilson. *Practical Parallel Processing.* MIT, Cambridge, MA, 1995.

[39] L. Wiskott, J. -L. Fellous, N. Krüger, and Ch. von der Malsburg. Face recognition and gender determination. In *International Workshop on Automatic Face- and Gesture-Recognition*, pages 92–97, 1995.

[40] P. H. Worley. A new PICL trace file format. Technical report, Oak Ridge National Laboratory, Oak Ridge, TN, USA, September 1992. Report ORNL/TM-12125.

Pipeline stage	Time	Ratio
head location	116s	55%
feature detection	90s	43%
face inspection	4s	2%

Table 1: Sequential Pipeline Timings for the Eigenfaces Application

Processor	Time/image (s)
Sun4 (Optimised)	84
SPARCstation 5 (Opt.)	37
i860 (Level 2 Opt.)	33
i860 (Level 4 Opt.)	30
Farm 6 (Level 0 Opt.)	19
Farm 6 (Level 2 Opt.)	13
Farm 6 (Level 4 Opt.)	11

Table 2: Initial Timings for the Eigenfaces Application

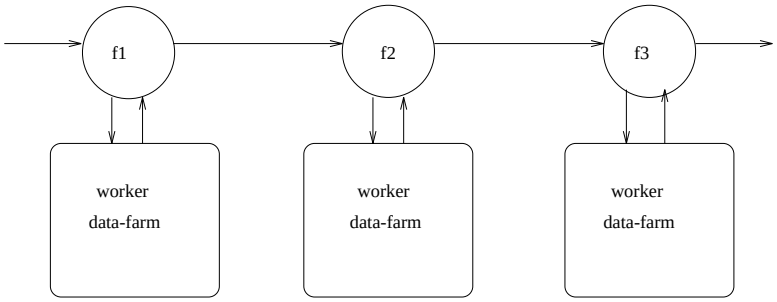


Figure 1: The PPF Design Pattern

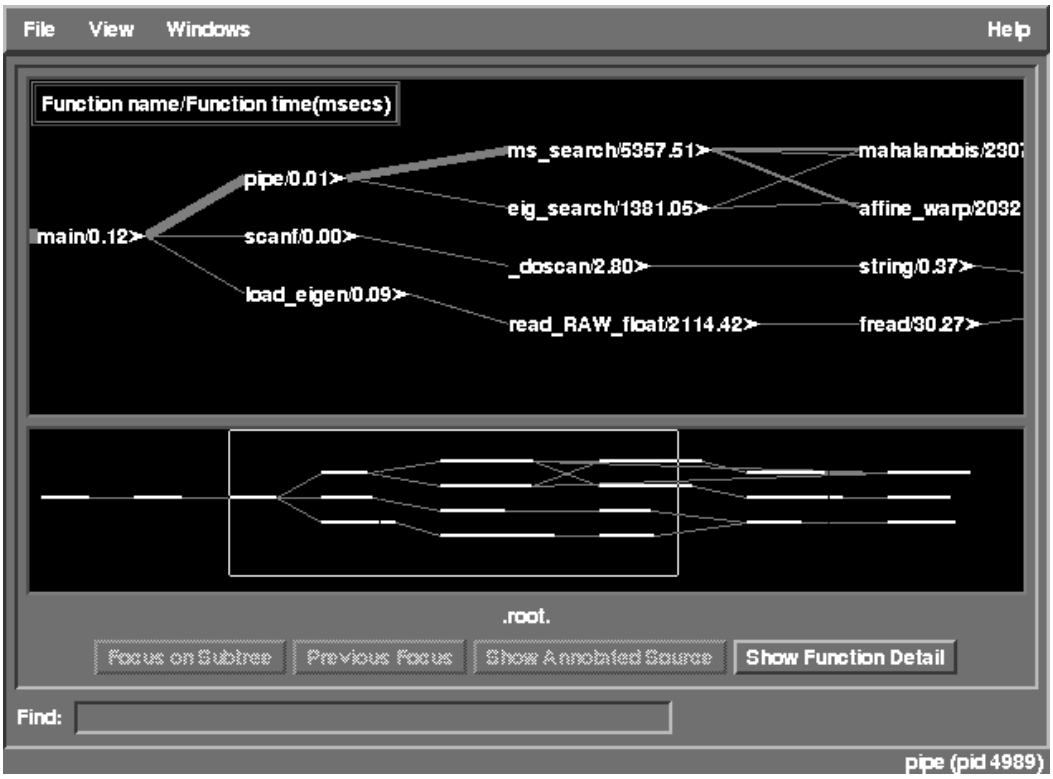


Figure 2: Call Graph for Part of the Eigenfaces Application

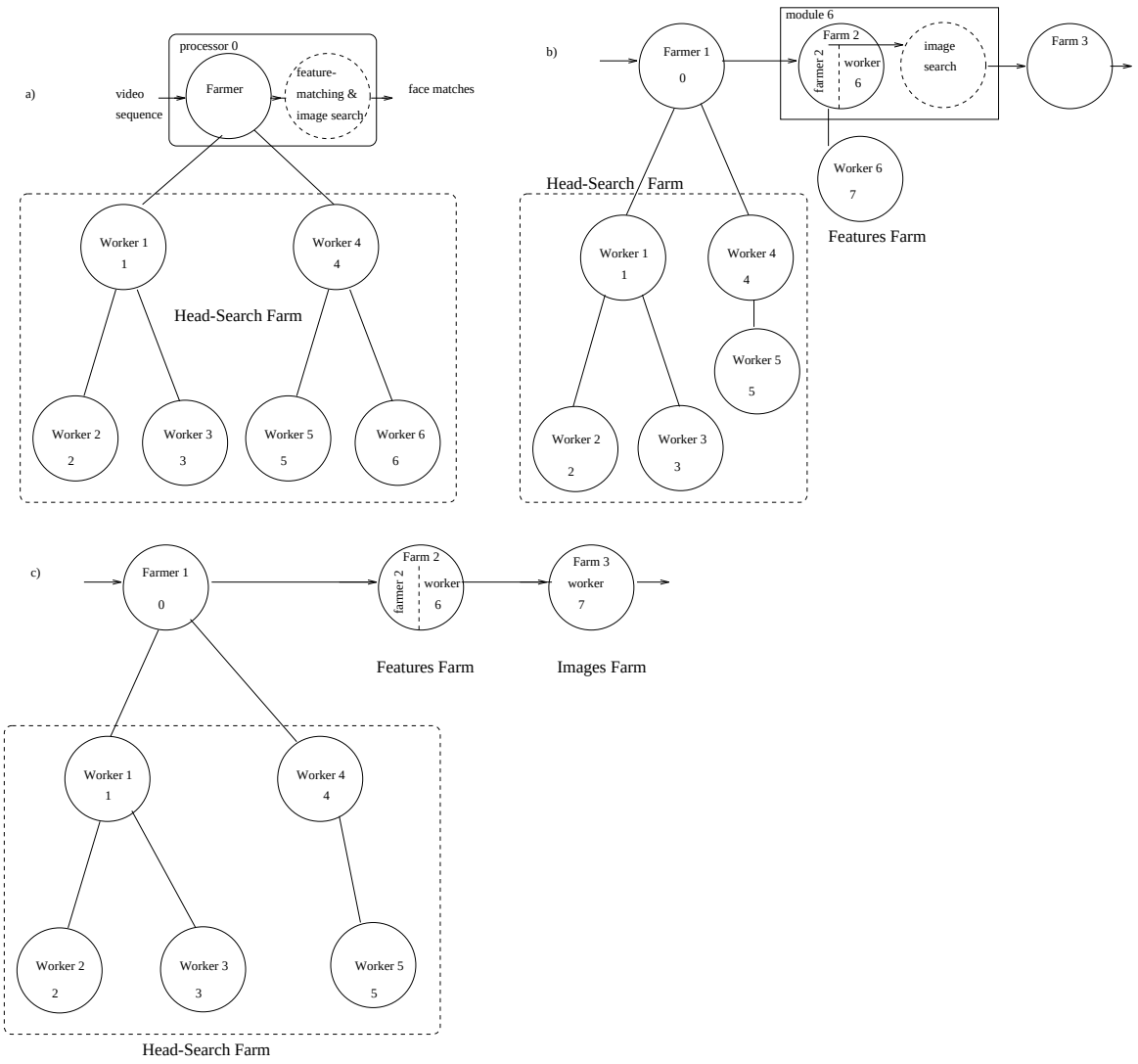


Figure 3: Implementation Topologies for the Eigenfaces Application

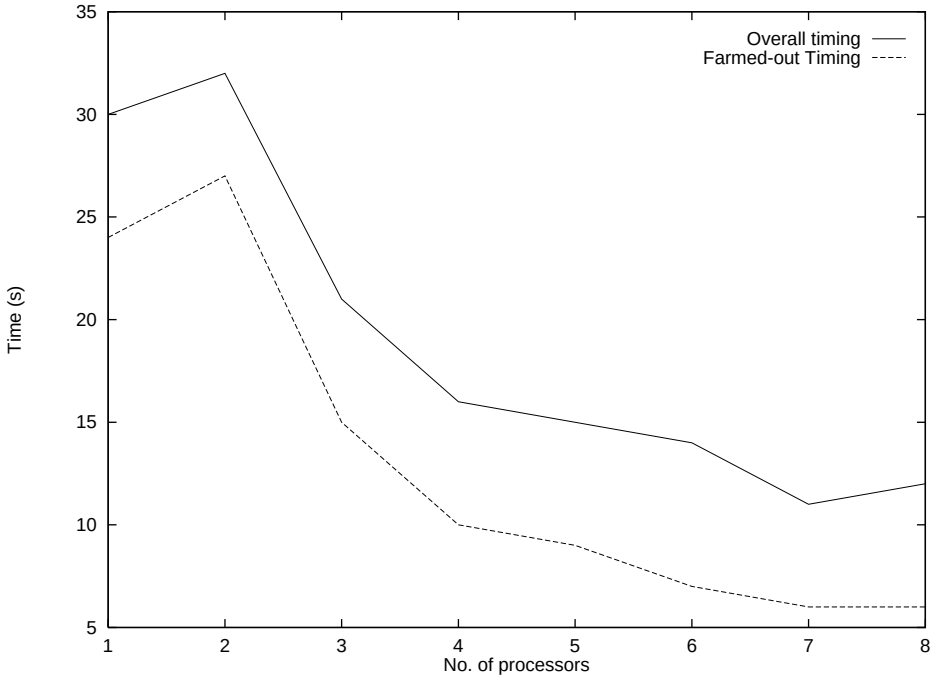
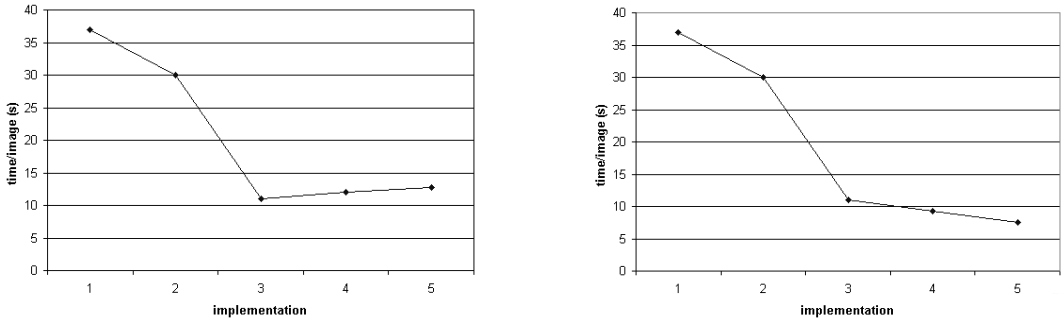


Figure 4: Eigenfaces Timings for One Farm



a) Latency b) Throughput

Figure 5: 1) SPARCstation 5, 2) i860, 3) Single farm, 4) 5:2:1 pipeline, 5) 6:1:1 pipeline Eigenfaces Timings

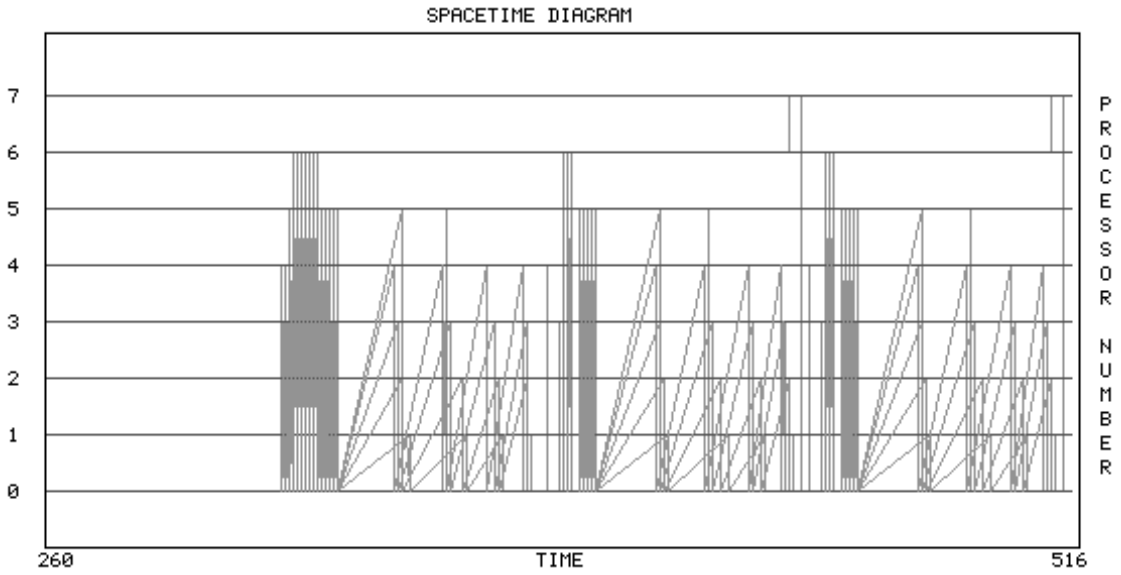


Figure 6: Event Trace of Eigenfaces Parallel Pipeline

List of Tables

List of Figures

6 Authors' brief biographies

Martin Fleury obtained a BA (Hons) in Maths./Physics in 1988. He received an MSc in Astrophysics in 1990 from the University of London and an MSc in Parallel Computing Systems in 1991. Between 1992 and 1998 Martin has worked at the University of Essex, on image-processing for reconfigurable parallel computers and from 1995 on the PSTESPA project concerned with embedded signal-processing applications. In 1997, Martin was awarded a PhD for work on Parallel Image-Processing Software. He is now employed as a lecturer at Essex. His current research is on the software engineering of distributed and mobile real-time multimedia systems.

Andrew Downton was educated at Southampton University, where he obtained a first class honours degree in Electronic Engineering in 1974, and a PhD in 1982. From 1974-1976 he worked in industry at the Lucas Group Research Centre. He returned to Southampton in 1976 as a research fellow, and was appointed a lecturer in 1978. In 1986 he moved to the University of Essex where he is now a Professor and Head of Department. His current research interests are in Document Analysis and Handwriting Recognition, Parallel Processing and Real-Time Systems, Virtual Reality and Human-Computer Systems.

Adrian F. Clark obtained a BSc in Physics at the University of Newcastle-upon-Tyne in 1979 and a PhD in Image Processing from University of London in 1983. After postdoctorate research into parallel image processing at Kings College, he worked for British Aerospace during 1985-88, researching in the general area of object recognition. He joined the University of Essex in 1988, where he is now a Reader. His research interests are principally in the algorithms that underlie the processing of image data and in the software techniques required to instantiate them on serial and parallel computers.