Structured parallel design for embedded vision systems: a case study

A. Çuhadar*, A. C. Downton and M. Fleury
Department of Electronic Systems Engineering,
University of Essex, Wivenhoe Park,
Colchester CO4 3SQ, UK
tel: +44 - 1206 - 872910
fax: +44 - 1206 - 872900
e-mail acd@essex.ac.uk

Abstract

The authors describe the parallelisation of a complex multi-component embedded vision system using a generalised parallel design methodology based upon pipelines of processor farms (PPFs). The complete vision system is decomposed into two pipelines: one recognises postcodes and the other extracts features from the remainder of the handwritten or hand-printed addresses and verifies them against an expected address derived from the postcode. The implementations illustrate how data parallelism, algorithmic parallelism and temporal multiplexing can all be implemented using a single design model, and how an arbitrarily complex system comprising many independent algorithm components can be decomposed in a structured way to achieve scalable speedup, throughput and latency. The initial implementation on a transputer-based Meiko Computing Surface is unable to meet the specification for postcode throughput, so the design and code has also been ported to a technology which allows the time constraints of the specification to be comfortably met. Details of the implementation refinements necessary for this hybrid transputer/i860 machine are provided. The range of algorithms utilised in this case study, and the flexibility of the parallel solutions generated, lead us to believe that the design method is applicable to a wide range of embedded vision applications.

Keywords: parallelisation, system design, postcode recognition

Introduction

In a previous paper [1], a design methodology for top-down structured parallelisation of embedded image processing applications was presented and illustrated using simple applications from the fields of image coding and computer vision. In this paper, we present a case study which applies the design method to a complex multi-component embedded vision system for recognising handwritten postal addresses on envelopes. The case study illustrates several important features of the design methodology:

1. Application of the methodology produces an incrementally scalable family of parallelisations rather than a single fixed solution, and so can easily be adjusted to cope with

*Also affiliated to the Department of Electrical and Electronic Engineering, University of Gaziantep, Turkey

changes in the structure and execution times of algorithmic components of the overall system. This means that it is well-suited to application throughout the research and development cycle, and not only to completed vision algorithms.

2. The design method initially assumes a static (*i.e.* constant execution time) model for individual components. This is not realistic for typical computer vision algorithms, where execution times may be strongly data-dependent. In practice however, dynamic variation of component algorithm execution times is accommodated by the choice of a suitable processor farm component-level implementation model, combined with empirical adjustment to the number of workers in each farm.

3. Substantial speedups can be obtained even for complex algorithms with abstract data structures and many inherently sequential components, provided that the overall application exhibits continuous unidirectional data flow from input to output. The case study achieves speedup factors of over 20 in our initial implementation, limited only by the number of processors available (32 in our case), and in some cases by the communication bandwidth between processors. The use of more powerful processing modules together with refinements of the implementation arrangements allowed the specification to be easily met.

4. The design method allows the latency specification for an embedded application to be met by further data decomposition and/or algorithmic parallelism.

Although the case study is based upon a single application, the range of algorithms it utilises, and the flexibility of the parallel solutions generated, lead us to believe that the design method is applicable to a wide range of embedded vision applications.

Application overview

Handwritten postal address recognition is a typical multi-level knowledge-based vision application for which a real-time embedded solution is sought. Real-time OCR systems for recognising printed addresses are widely used by the Royal Mail and foreign postal services, but systems capable of recognising handwritten postal addresses have not yet achieved either the recognition performance or the throughput required for commercial application.

Figure 1 shows a top-level block diagram of our handwritten postal address recognition system, which can be broken down into two independent and inherently parallel pipelines of components. One of these attempts to recognise the characters within the postcode (from which a corresponding address can be determined using the UK postal address database) while the other simultaneously extracts features which are used to verify the address predicted from the postcode. If the verification features match the postcode address prediction sufficiently accurately, the address is accepted, otherwise the mail-piece is rejected for manual sorting. Full details of the algorithms used within the postcode and address verification pipelines are presented in [2] and [3] respectively.

In this particular application, the system design specification is that 10 envelopes/second must be processed by the machine. Furthermore, the size of the computational pipeline is limited by the associated mechanical pipeline which is specified as having a maximum capacity of 90 envelopes (*i.e.* 9 seconds latency). An ordering constraint also exists, in that results

from the OCR system must emerge in the same sequence that the mail-pieces were scanned, to ensure correct phosphorescent dot coding of each envelope with its corresponding postcode.

Since the postcode recognition and address verification pipelines shown in Figure 1 can operate concurrently and independently, each can be parallelised separately using the Pipeline Processor Farm model described below, and their processing speeds may be balanced to meet the overall throughput and latency specification. The following sections therefore present independent parallelisation case studies for the postcode recognition and address verification algorithms. The succeeding section steps down a level in detail in describing a port to recent hardware more capable of meeting the specification. The fact that this was readily and rapidly accomplished makes plain the soundness of the design method. The final section draws some conclusions.

Generic design method

Pipeline Processor Farm (PPF) design model

The PPF design model is part of a parallel design methodology for embedded systems with continuous data flow which can be used to decompose existing sequential applications onto any type of parallel processor network with any communication model [1]. It maps the sequential algorithm structure to a generalised parallel architecture based upon a pipeline of stages with well-defined data communication patterns between them. Each stage of the pipeline exploits parallelism in the most appropriate way, for example data parallelism applied at various levels, algorithmic parallelism, or temporal multiplexing of complete input data sets. Processor farming is used to implement all these forms of parallelism because it allows indefinite incremental scaling of any stage and results in a single tractable analytical model.

The application is mapped onto pipeline stages using top-down profiling data derived from sequential application execution. The mapping process identifies the individual stages and their relative computational requirements and allows the required number of processors in each stage to be directly calculated. In its simplest form the model ignores communication overheads between processors, and assumes static task execution times. This leads to an upper-bound performance scaling estimate which, although not practically achievable, is useful in determining performance scaling trends. More accurate performance modelling can be achieved by analysing dynamic profiling data and inter-function communications, but this is not strictly necessary, as efficiency can be optimised empirically by varying the number of worker processors in each processor farm, as is shown in the case study examples below.

Implementation issues

The address recognition system was originally implemented as a sequential simulation algorithm on a Sun Sparcstation [4]. It was then ported to a single T800 transputer running on a Meiko Computing Surface, which was used to obtain the execution profiling data presented below. All initial parallelisations were implemented using Meiko's CSTools parallel programming environment. This provides a library of C functions for implementing virtual channel communication between arbitrary pairs of processors, whether or not they are physically directly connected. A further library of functions (CSBuild) which run on the host processor (a Sun workstation), allows a custom loader program to be written which configures and then loads and runs a network of transputers.

In the case study examples presented below the CSBuild loader program reads parameters from the command line which define the overall pipeline configuration and specify the number of worker processors to be configured within each processor farm. This allows practical results to be rapidly obtained for a large number of different PPF configurations, as shown in the results graphs below. An important advantage of using processor farms to implement all forms of parallelism is that the same communication functions are used for communication between a farm master and its worker processors, regardless of the number of workers in the farm. As a result it is straightforward to program reconfigurable and incrementally scalable PPFs.

Parallelisation of the postcode recogniser

The OCR algorithm for handwritten postcodes utilises character features proposed in the characteristic loci algorithm [5] (preprocessing stage) combined with a quadratic classifier (classification stage). Ranked lists of characters at each postcode character position are then pruned by applying postcode syntax rules. The n ($n \leq 5$) highest ranked characters remaining in each postcode character position after applying the syntax rules are then permuted to form n^6 (6 character postcodes) or n^7 (7 character postcodes) possible postcodes which are presented to the dictionary. Postcodes which are matched in the dictionary are sorted according to an overall match function derived by multiplying the matches for individual characters, and the addresses corresponding to one or more best matched postcodes are retained for verification against features extracted from the handwritten address.

It is necessary to introduce some limited data parallelism (splitting the processing up by a data division, though subsequent load balancing can be dynamic or static) in order to meet the application latency specification of 9 seconds, since although speedup can be achieved using temporal multiplexing alone (*i.e.* retaining the original processing granularity), temporal multiplexing has no effect on latency, and the latency of the original sequential algorithm is 10.5 seconds. Conversely however, temporal multiplexing requires less parallel design effort than either data or algorithmic parallelism, hence the designer's objective is to introduce only as much data or algorithmic parallelism as is necessary to satisfy the specification. Since some parts of the application are inherently sequential, the overall application must be partitioned so as to separate these parts (which can only be speeded up using temporal multiplexing) from other parts to which data and/or algorithmic parallelism can be applied. In the following section, possible partitioning points for the postcode recognition algorithm are identified by considering what forms of parallelism can be applied to each algorithm component.

Partitioning the postcode recogniser

Table 1 summarises the average processing times for the major functions within the postcode recognition algorithm, and is derived from sequential execution time statistics obtained while running the algorithm on 100 address images of data on a single T800 processor.

As can be seen from the table, the preprocessing and classification tasks exhibit almost the same computational complexity and are three times slower than the dictionary search algorithm (measured for $n = 5$).

The decomposition of the system into a pipeline of processor farm stages can be carried out as follows:

- **Preprocessing.** This is composed of 5 basic tasks as shown in Table 1. Due to the sequential nature of the tasks, algorithmic parallelism is not feasible, but data parallelism can be implemented at the level of postcode characters or character features. Since the features for each of the characters need to be combined before the classification stage takes place, it is necessary to implement classification on a different processor farm from preprocessing if different levels of parallelism are exploited in the two stages. In the implementation described below, both feature extraction and classification exploit character-level parallelism, however each stage was implemented as a separate processor farm to allow a future upgrade to the use of feature-level parallelism within the preprocessing stage if necessary (this would potentially reduce latency further).

- **Classification.** This comprises three algorithms: transformation of the features into quadratic space, location of the region where the classes lie in quadratic space and production of a ranked list of postcode characters. Again algorithmic decomposition is not applicable, and in this case data parallelism can be implemented only at the character level. Since the ranked lists of characters need to be combined before presenting them to the dictionary stage, it is necessary to implement the classification and dictionary stages on different processor farms.

- **Dictionary.** This involves application of the syntax rules (a table look-up operation), generation of all possible postcodes from the remaining ranked lists of characters, and a trie search [6] in the postcode/address dictionary. A form of algorithmic parallelism can be applied here by dividing the full postcode dictionary into 6 sub-dictionaries corresponding to the 6 possible UK postcode formats, but in the implementation reported below temporal multiplexing alone was applied to this stage.

By introducing character-level data parallelism to the preprocessing and classification stages, the latency of these stages should be reduced by a factor of between 6 and 7, leading to an overall mean latency of less than 3 seconds (ignoring communication overheads). The implementation below therefore describes a design which utilises data parallelism in the first two pipeline stages and temporal multiplexing alone in the final dictionary stage. The performance for this implementation is then compared with that of an implementation which utilises temporal multiplexing in all three stages.

Scaling the postcode recogniser

Parallel implementation of the system was performed in two steps. Firstly, parallel versions of each stage were implemented independently, to enable the dynamic scaling performance of each stage to be measured. Then the overall system was formed by configuring the stages into a pipeline. Figure 2 shows a plot of throughput (postcodes per second) performance measured against number of processors for each independent pipeline stage for the case where character-level data parallelism has been utilised in the preprocessing and classification stages, and temporal multiplexing alone in the dictionary stage.

The plots reveal a number of important points:

- The throughput achieved by each stage scales incrementally and fairly linearly up to the maximum number of transputers available in the Meiko system (32). At this limit, the dictionary stage is at the required throughput level, and the other two stages are operating at just over half the specified throughput. Further increases in throughput

could be achieved at each stage if more transputers were available, as no stage is close to saturation of its communication links.

- The throughput of the dictionary stage scales less rapidly than might be expected from the static profiling statistics of Table 1. This is because the execution time for this stage is an average of a strongly bimodal distribution (see Figure 3), as 7 character postcodes take about five times as long as 6 character postcodes to process in the dictionary.

  The large variation in execution times leads to queueing delays at the output of the dictionary, since postcode ordering within this stage must be preserved. This degradation in the performance due to the wide distribution of processing times in the dictionary stage can also be predicted theoretically [7].

- For a given throughput, Figure 2 allows the required number of processors in each stage to be estimated so as to achieve a balanced and efficient PPF implementation.

The full postcode recogniser PPF consists of a pipeline of three ternary tree processor farms. Each processor farm comprises a master processor which receives data from the previous stage (from the input device in the case of preprocessing) and distributes it over the worker processors allocated to that stage. As soon as a worker finishes processing, it returns its results to the master, which forwards them to the next stage and sends new data to the worker. In the first two stages, where data parallelism is exploited, each work packet comprises the data required to process a single postcode character; in the dictionary stage each work packet consists of the ranked list of character matches for a complete postcode.

Performance achieved

Speedup, throughput and latency of the overall PPF were measured by first choosing fixed and equal number of processors at the preprocessing and classification stages (as required from Figure 2). The number of processors in the dictionary stage was then chosen as the independent variable because of the large divergence between static and dynamic scaling predictions for this stage.

Example speedup graphs are shown in Figure 4 for the cases of 3, 5, 7, 9 and 11 worker processors in each of the first two pipeline stages. Each graph shows how the speedup varies with the number of workers in the dictionary stage, for the specified number of workers in the other two stages. The ideal graph is a line of gradient 1 which represents a parallel implementation in which all processors operate at 100% efficiency. In reality, efficiency will be less than 100% due to processor overheads such as the master processor's housekeeping operations for each stage, variability of execution time for different work packets, and communication overheads.

The general form of each speedup graph is similar: as the number of processors in the dictionary stage is increased, the achieved speedup also increases until saturation occurs when the dictionary stage no longer limits the pipeline throughput. Optimum efficiency occurs at the point where each graph most closely approaches the ideal graph; at this point, the pipeline is balanced. For the graphs shown, balanced pipelines are achieved with 9 (3+3+3), 15 (5+5+5), 19 (7+7+5), 25 (9+9+7) and 29 (11+11+7) workers, and correspond to efficiencies of 53.3%, 58.7%, 60%, 64.8% and 66.9%. The gradual increase in efficiency is a result of the master processor overhead decreasing as the size of the processor farm increases, and would eventually decline again as communication saturation is approached.

Figure 5 compares the throughput of the complete 3 stage PPF postcode recogniser described above with a similar alternative implementation which utilises only temporal multiplexing in each of the three stages. Both implementations achieve linear throughput scaling as long as the numbers of workers in each of the three processor farm stages are maintained in balance, but the PPF implemented solely using temporal multiplexing achieves slightly greater throughput for two reasons:

1. The fixed overhead of header information in each communication packet means that more data must be communicated in total when the data stream is divided into character-based packets than when it is divided into postcode-based packets.

2. Splitting the postcodes up into individual characters and processing them independently increases the ordering constraints, as the ranked list of characters at each postcode character position must be recombined into postcodes at the dictionary searching stage before any processing starts in this stage.

The main advantage of introducing data parallelism into the implementation is that it decreases the latency of the pipeline, as is shown in Figure 6 which indicates the latency measured (in seconds) for the two different implementations. The latency of the postcode-parallel implementation remains constant regardless of the number of processors since there is no sub-division of either the data or the algorithm in this implementation. In contrast, the latency of the character-parallel implementation decreases as more processors are added until sufficient processors are available to fully exploit the data parallelism in the design (7 workers in each of the first two pipeline stages). For PPFs with sufficient processors, the latency of the character-parallel implementation is 3.2 seconds whereas it is 10.8 seconds (*i.e.* the same as for the original sequential application, but with additional communication overheads) for the pure temporal multiplexing approach. Hence the introduction of some limited data parallelism into the implementation makes it possible to satisfy the latency specification for the system, at the expense of a slight decrease in throughput and efficiency.

Parallelisation of the address verifier

The redundancy between the postcode and the remainder of the address is exploited by extracting features of the address which are then matched against one or more candidate addresses corresponding to the postulated postcode(s) and derived from the UK postcode/address database [2]. The verification process consists of two major stages:

- **Preprocessing.** Address images are processed to extract first the address lines, then the address words, and finally, a slant correction process is performed on each address word to minimise variations in the features relative to the slant of the handwriting.

- **Feature extraction.** A number of algorithms are applied to the address word images to extract predefined global and local features, which include initial and final characters of words, loops, and ascender/descender sequences.

The final stage of the complete address recognition system involves matching features extracted from the address image against corresponding features derived from the reference addresses found by the dictionary search. If a sufficiently high match is found, the postcode is accepted otherwise it is rejected. This matching process is around 10^3 times faster than

the other algorithms described above, therefore parallelisation effort was concentrated on the preprocessing and feature extraction stages of verification.

Preprocessing and extraction of verification features is roughly 10 times more computationally expensive than postcode recognition, and the volumes of data communicated between stages are also correspondingly larger, because verification utilises the full address image, whereas postcode recognition only operates on the constrained postcode image field.

Partitioning the address verifier

Figure 7 shows that the preprocessing and feature extraction stages of the address verification algorithm can be broken down into a pipeline of 4 independent stages, and gives the average sequential execution time on a single T800 transputer for each stage, and also the amount of data communicated between the stages.

- **Line extraction.** At the line extraction stage, image processing operations are applied to the complete address image: parallelisation can therefore best be achieved by temporal multiplexing. The input data to the task is the complete address binary image, and the output data is a set of up to 5 binary address line images.

- **Word extraction.** Word extraction operations are applied to each line of words in the address separately, hence line-level data parallelism can be readily exploited in this stage as well as temporal multiplexing. The output data from this stage is a set of up to 5 binary word images per line.

- **Slant correction.** Slant correction is applied separately to each word image extracted by the previous two stages, hence word-level or line-level data parallelism is easily exploitable here in addition to temporal multiplexing. Both input and output data consist of a set of independent images of each word in the address.

- **Feature extraction.** Feature extraction consists of four independent algorithms (character segmentation, word case classification and ascender/descender detection, character recognition and loop detection) which are configured as shown in Figure 8.

  Word-level data parallelism is again available at this stage, but in addition algorithmic parallelism can also be exploited, since up to three tasks can be performed concurrently (corresponding to the three independent branches).

Scaling the address verifier

The partitioning above initially suggests that a pipeline of four independent processor farms with worker processors provided for each stage in the rough ratio 3:1:2:5 will achieve approximate throughput balance within the pipeline, and provide maximum opportunity to exploit data parallelism, and thus minimise pipeline latency. In practice however, it was noted that by combining the word extraction and slant correction stages, a simpler three-stage pipeline could be achieved in which static balance occurs for worker processors in the ratio 3:3:5. The results reported below are for this configuration, in which temporal multiplexing alone is applied at the first stage, line-level data parallelism and temporal multiplexing at the second stage, and algorithmic parallelism, word-level data parallelism and temporal multiplexing at the final stage. It should be apparent that several other processor configurations are also possible.

Address verification farms

The address line and address word segmentation farms exploit temporal multiplexing and data parallelism and are conceptually similar to the postcode recognition farms described earlier. However, the address feature extraction stage combines data parallelism (the complete address is divided into individual word images) with algorithmic parallelism (workers execute one of three different algorithms, as shown in Figure 8), and therefore operates in a somewhat more complex way than any of the processor farms previously described. For simplicity, the algorithmic parallelism is distributed statically, by allocating one third of the total processors in the farm to each of the three required algorithms (each process has roughly similar static computational complexity). The master processor therefore has to buffer the input word image data, since the same data is sent to two of the three algorithmic processes.

The first worker process receives word images from the master and performs loop detection. The number of extracted loops is returned to the master. The second worker process receives the same word images from the master and performs the first character segmentation, word case classification and ascender/descender detection, last character segmentation and recognition tasks. It returns the ascender/descender sequence or the recognised last character in ranked order, depending on whether an upper case or mixed case word was detected. This worker process is also responsible for transmitting the first character of the word image after the segmentation process to the third worker process. Finally the third worker process receives the first character of the word image from the second worker and performs character recognition on it. Recognised characters in ranked order are returned to the master.

Overall performance achieved

The full address verification PPF was built by connecting the three stages described above in a pipeline configuration. The performance of the verification pipeline was measured by fixing the number of processors in the first two stages and varying the number of processors at the feature extraction stage, as shown in Figure 9. In this Figure, the 'ideal' graph represents the speedup if perfect buffering between stages were obtainable and communication overheads could be ignored, but the residual sequential overhead of each pipeline stage's master processor farming out data to its workers is taken into account by applying Amdahl's law [8]. Three sample practical speedup graphs are also shown; these represent parallel configurations in which 3, 5 and 7 worker processors are utilised in each of the first two pipeline stages, and the number of workers in the final stage is then varied. As can be seen, for each configuration, the performance increases fairly linearly as additional processors are added to the final stage, until that stage no longer limits throughput, at which point adding further processors has no effect. Thus the speedup scales incrementally with the number of processors used, achieving a maximum of about 15 with 26 workers (29 processors total), an efficiency of 52%. Figure 10 shows that the throughput achieved with optimal PPF configurations also scales fairly linearly up to 29 processors, but at this level it is still about 80 times slower than the required real-time performance specification.

Meeting the Specification

The maximum speedup achieved for the verification pipeline was 15 with 29 transputers in total, and in this case the throughput (0.12 address/second) and latency (58.7 seconds)

specifications are still far from being met. More significantly, only a limited increase in speedup could be achieved by adding more processors, because the communication channels saturate with 15 - 25 workers in each processor farm. In reality, it is not surprising that the specification has no chance of being met by the verification PPF, as this would require a speedup of around 10^3, which is well beyond the practical scaling range feasible using this technique. Future attempts must await the arrival of processors with higher communicate to compute ratios.

For the postcode recognition pipeline, an overall maximum throughput of just over 2 postcodes/second and mean latency of 3.2 seconds were achieved with a PPF of 32 transputers (29 workers and 3 masters). These results correspond to a speedup of 21.4 for the application as a whole, and an overall processor efficiency of 66.9%. As no stage is close to communication saturation, it was concluded that simply by introducing more powerful computational engines to augment or replace the transputer the specification could be fully met. The remainder of this section describes a port to an eight module Paramid machine [9], where each computational node consists of one Inmos T805 transputer and one Intel i860-XP micro-processor. The i860 acts as a high-performance computational engine allowing the transputer to become a dedicated communications co-processor. In our implementations, transputers also took the rôle of master processors, distributing work packets to workers, so that it was possible to configure a maximum of 3 pipeline stages with up to 8 workers in total.

Unlike the prototype implementation on the Meiko CS, buffering was used both locally at the interface between each i860 and transputer and globally between each stage of the pipeline. The former buffers are needed to ensure that the i860 does not wait for the transputer and the latter are used to ease flows along the PPF. In particular this reduces the bottleneck at the dictionary stage where there is a bi-modal computation time distribution (Figure 3). The best number of slots in all local buffers was found to be at least ten.

As in the earlier transputer implementation, the Paramid implementation utilised data parallelism in the preprocessing and classification stages and temporal multiplexing in the dictionary classification stage. Table 2 shows the throughput and latency achieved with an optimal 4:3:1 configuration of 8 processors overall within the pipeline, and confirms that with this configuration both the throughput and latency specifications for postcode processing are met. The demand-based farming method timings now show an increase in throughput, because of the added flexibility.

Conclusion

This paper has presented a case study showing how a top-down structured generic parallel design method can be successfully applied to a typical multi-level computer vision algorithm. Incrementally controllable speedups of up to 21.4 for the postcode recogniser and 15 for the address verifier were obtained using up to 32 processors. When the postcode recognition pipeline was transferred to the more powerful i860-based Paramid machine the specification for postcode specification was met and surpassed. The range of algorithms utilised in the case study, and the flexibility of the parallel solutions generated, lead us to believe that the design method is applicable to a wide range of embedded vision applications.

Acknowledgement

This work was carried out as part of project GR/K40277 ("Parallel Software Tools for Embedded Signal Processing Applications") and as part of project IED3/1/2171 ("Parallel Reconfigurable Image Processing Systems").

References

[1] 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, December 1994.

[2] A C Downton, R W S Tregidgo, and E Kabir. Recognition and verification of handwritten and hand-printed british postal addresses. *International Journal of Pattern Recognition and Artificial Intelligence*, 5(1):265 – 291, 1991.

[3] Hendrawan, A C Downton, and C G Leedham. A fuzzy approach to handwritten address verification. In *Proceedings of the Third International Workshop on Frontiers in Handwriting Recognition*, pages 409 – 416, Buffalo, New York, USA, May 1993.

[4] Hendrawan. *Recognition and Verification of Handwritten Postal Addresses.* PhD thesis, University of Essex, December 1994.

[5] A L Knoll. Experiments with characteristic loci for recognition of handprinted addresses. In *IEEE Trans. Comp.*, April 1969.

[6] D. E. Knuth. *The Art of Computer Programming*, volume 3 Sorting and Searching. Addison-Wesley, 1973.

[7] R W S Tregidgo. *Parallel Processing and Automatic Postal Address Recognition.* PhD thesis, University of Essex, 1992.

[8] G M Amdahl. Limits of expectation. *Int. Journal Supercomputing Applications*, 2:88–94, 1988.

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

List of Figures

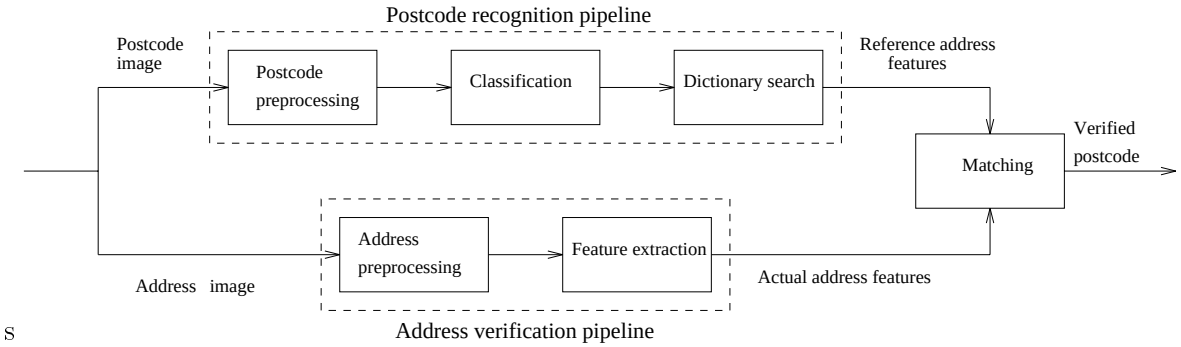
1 Postal OCR system 16
2 Throughput graphs achieved by scaling individual stages of the postcode recogniser 17
3 Distribution of execution times for the dictionary search stage of the postcode recogniser 18
4 Speedup graphs for the postcode recognition pipeline 19
5 Throughput graphs for character-parallel and postcode-parallel postcode recognisers 20
6 Latency graphs for character-parallel and postcode-parallel postcode recognisers 21
7 Functional block diagram of the address verification system 22
8 Functional block diagram of address feature extraction 23
9 Speedup graphs for the complete address verification pipeline 24
10 Throughput scaling graph for the complete address verification pipeline . . . 25

Process	*Function*	*Average time (second/image)*	*Data packet size (bytes)*
Preprocessing	Filtering	0.650	2112
	Feature extraction	1.170	
	Feature concentration	0.220	
	Feature unification	1.640	
	Feature counting	0.790	
	Total	4.470	
Classification	Quadratic space transformation	0.007	33
	Character classification	4.400	
	Ranking calculation	0.040	
	Total	4.447	
Dictionary	Search	1.500	82
	Sort	0.004	
	Total	1.504	

Table 1: Average execution times for the functions in the postcode recognition process and data packet sizes communicated between the stages

Number of Postcodes	Time (s)	Throughput (postcode/s)	Mean Latency	S.D.
100	9.23	10.83	1.81	0.29
200	17.93	11.16	2.11	0.37
300	25.89	11.58	2.14	0.32
400	35.11	11.39	2.23	0.33

Table 2: Throughput and Latency of The Postcode Recognition Pipeline



S

Figure 1: Postal OCR system

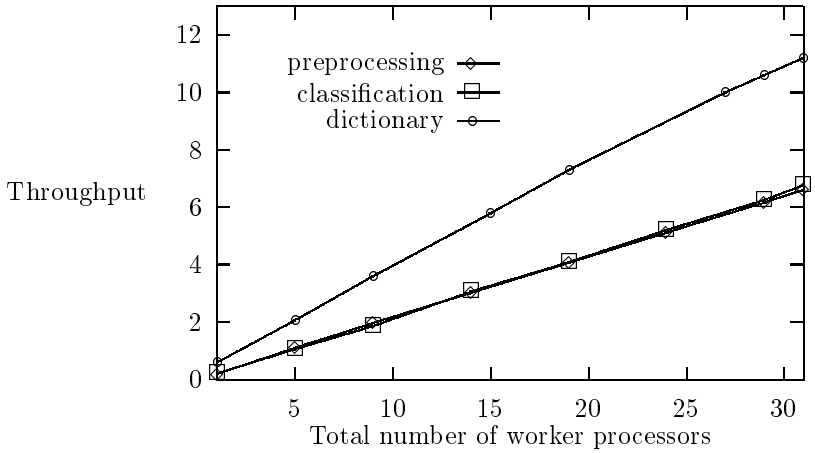


Figure 2: Throughput graphs achieved by scaling individual stages of the postcode recogniser

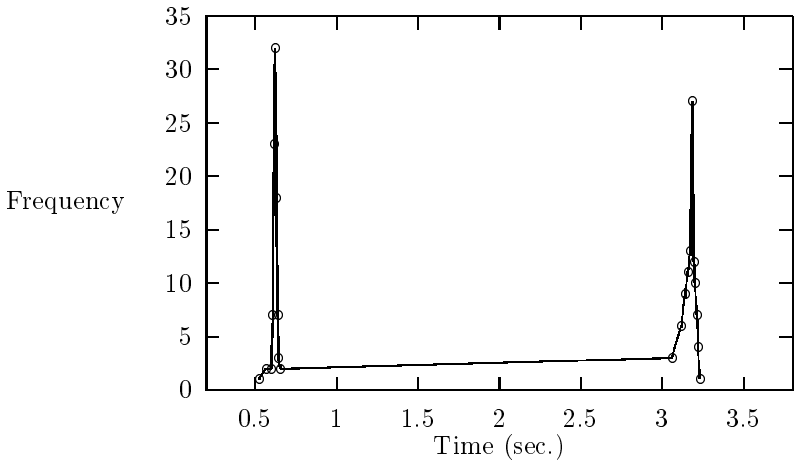


Figure 3: Distribution of execution times for the dictionary search stage of the postcode recogniser

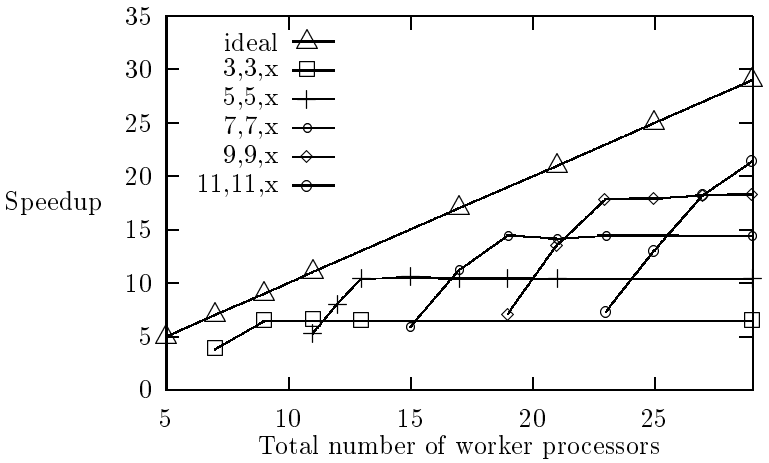


Figure 4: Speedup graphs for the postcode recognition pipeline

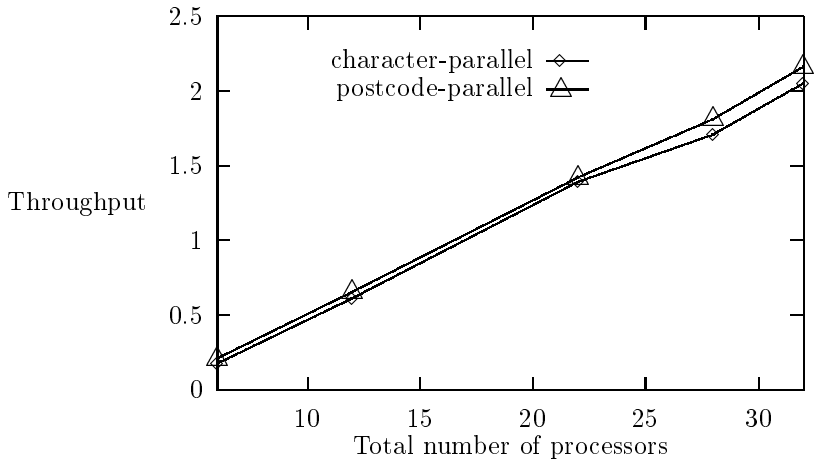


Figure 5: Throughput graphs for character-parallel and postcode-parallel postcode recognisers

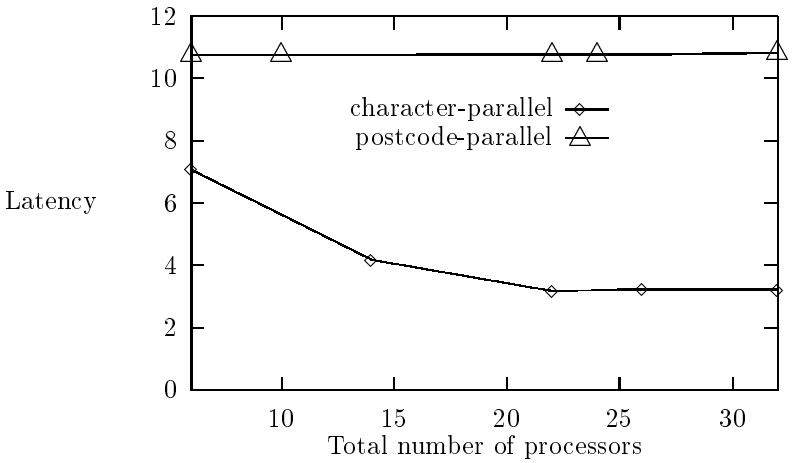


Figure 6: Latency graphs for character-parallel and postcode-parallel postcode recognisers

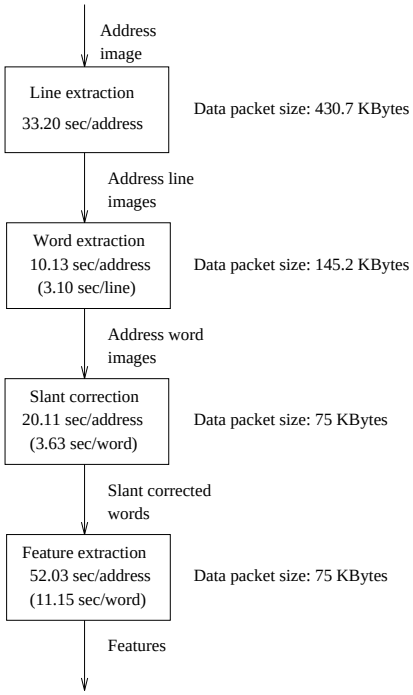


Figure 7: Functional block diagram of the address verification system

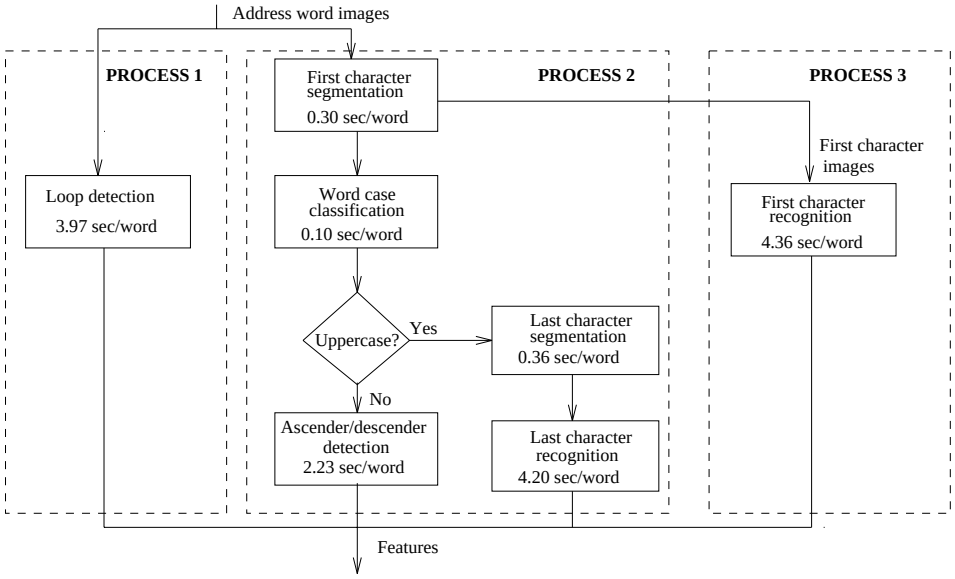


Figure 8: Functional block diagram of address feature extraction

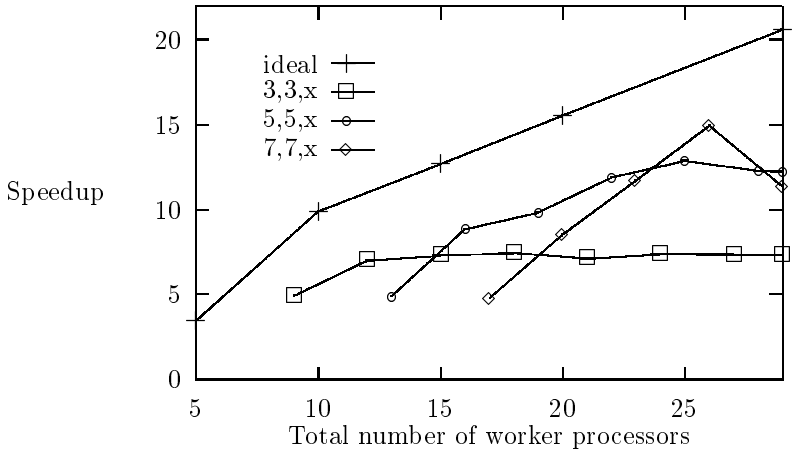


Figure 9: Speedup graphs for the complete address verification pipeline

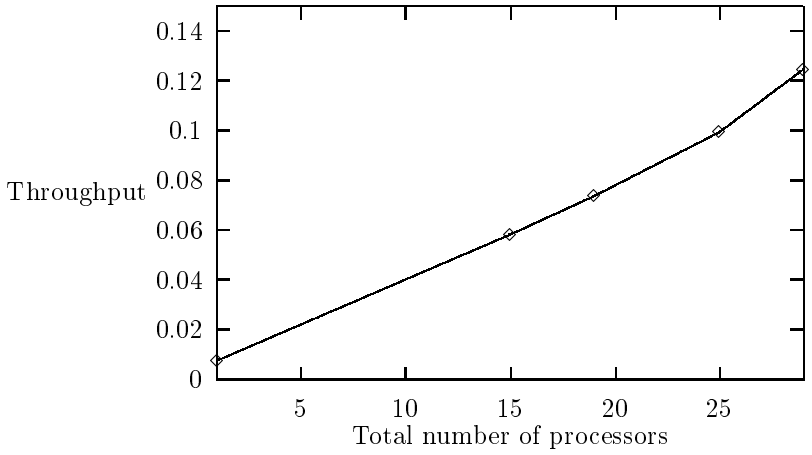


Figure 10: Throughput scaling graph for the complete address verification pipeline