A Parallel-System Design Toolset for Vision and Image Processing

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

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

Abstract. This paper analyses the requirements of a toolset intended for integrated parallel system design. Real-time, data-dominated systems are targeted, particularly those in vision and image-processing. A constrained abstract model is represented based around software pipelines each stage of which can contain a parallel component. The toolset is novel in seeking to enable generalized parallelism in which the needs of the system as a whole are given priority. The paper describes the tool infra-structure, and reports on current progress on the performance prediction toolset element, including simulation visualizer and analytic support.

1 Introduction

This paper describes a parallel-system design toolset which will guide the engineer 'in-the-field' towards the construction of Pipeline Processor Farm (PPF) applications. PPF is a stylized design and development methodology aimed at continuous-flow, embedded systems. More precisely, the target systems are expected to be driven by soft real-time constraints such as pipeline throughput, and traversal latency. Recently, a number of medium-scale, data-dominated, vision and image-processing applications [1, 2, 3, 4, 5, 6], have been parallelized along PPF lines. We tentatively define medium-sized systems to contain several algorithmic components, with approximately 3,000 to 50,000 lines of code. Examples of applicable irregular, continuous-flow systems can be found in vision, radar [7], speech processing [8], and data compression [9].

PPF promotes the notion of a software pipeline, which can subsequently be transferred onto available hardware according to client need, analogously to the way relational database systems are mapped to differing parallel hardware [10]. It has been observed [11] that many parallel algorithms merely form a sub-system of a vision-processing system. A way of combining algorithmic components in a coherent whole is required, though any subsequent changes should be self-contained. A pipeline may be the natural architecture for vision processing [12]; indeed, it is the architecture used by the human mind, the result of engineering through natural selection. In PPF, a pipeline is a convenient organising entity.

Each stage of the pipeline can independently cater for the differing requirements of a system's algorithmic components, either through centralized processing, data farming, or some other form of algorithmic parallelism.
By adjusting the system parameters differing performance goals can be achieved. For example, identification of handwritten postcodes falls into three stages: pre-processing of a digitised postcode image; classification of the characters within the postcode; and matching the postcode against a dictionary of postal addresses. By timing the algorithmic components of a system in a sequential setting, the ratio of processing power required at each stage of a pipeline can be initially assessed, though the extent of testing will determine the veracity of the results. For full generality (rather than heuristics), either second-order statistics or even identification of the statistical distribution formed by the processing times is necessary. The distribution can be estimated by fitting a distribution to the timings' histogram, for example by a chi-squared test. In the postcode example, the first stage might be achieved by farming complete postcode images to a set of worker processors, the second stage by the same or by decomposing processing to individual postcode characters, and the third stage by means of a trie search on each processor. A pipeline enables (say) easy transition to an alternative search engine, such as a syntactic neural net. The first two stages may be approximated by parameterized truncated Gaussian (normal) distributions while processing of the final stage is bi-modal, UK postcodes being either six or seven characters in length. It emerges that a reduction in latency is achievable if the second-stage processing is on a character basis and the postcode characters can be re-assembled into postcodes without significantly impeding the dictionary search [4].

As timings on sequential code give static requirements, the system designer, aiming for a reduction in hardware costs, will also want to know dynamic effects such as: the mean, variance and maximum work flow, i.e. the throughput and traversal latency metrics of a particular configuration of a parallel pipeline in the long run; the likely steady-state per-stage scheduling regime behaviour; what memory requirements will be needed if there are temporary holdups in the workflow requiring buffering; the effect of varying compute and communicate parameters; and will a particular processor employed at any one stage spend a significant time idling while a less-powerful processor would suit the granularity of the tasks more closely. The PPF performance tools progress the designer to a resolution of these issues.

2 Toolset Overview

Previous work on the PPF toolset has entailed identifying semi-manual methods of partitioning existing sequential code, i.e. identifying likely partition points, and transferring the partitioned code to pre-written high-level software templates which will enact individual processing stages. The ability to routinely parallelize algorithms in a generic manner enables algorithms to be prototyped on intermediate, general-purpose parallel hardware. Introducing an intermediate stage has several advantages: a central parallel version of the system is available for transfer to a variety of client machines according to individual processing needs; the correct working of the parallelisation can be checked; and if instrumented high-level, software templates have been used in the construction of the pipeline an iterative design cycle is set-up in which predicted performance is compared to actual through the medium of an event trace. The PPF design cycle is shown in Fig. 1. The performance predictor works by simulation and analytic means. Because one wishes to compare predicted with actual performance the simulation should set-up a visual impression of activity within a pipeline segment which the designer can compare with recorded activity on the prototyping machine, using a similar display format. The display format of the simulator is deceptively simple as: the PPF methodology restricts the degrees of freedom in the parallel system development path; and extraneous detail is avoided so that the mapping between prototype design and target system(s) is not prematurely fixed. For generality, a machine-neutral description [13] is required, which will reflect a designer's linguistically-based thought processes in developing a design. For example, it is important to present the concept of 'pipeline' and 'data-farm' which do not already exist in other common visualizers, such as ParaGraph. Analytic results are used for synchronous pipelines (Section 4) and to check the simulated results for some mathematically tractable distributions. Synchronous and asynchronous pipeline segment results are then combined. Single-stage scheduling regimes using either fixed or variable task sizes are also open to simulation and analytic prediction.

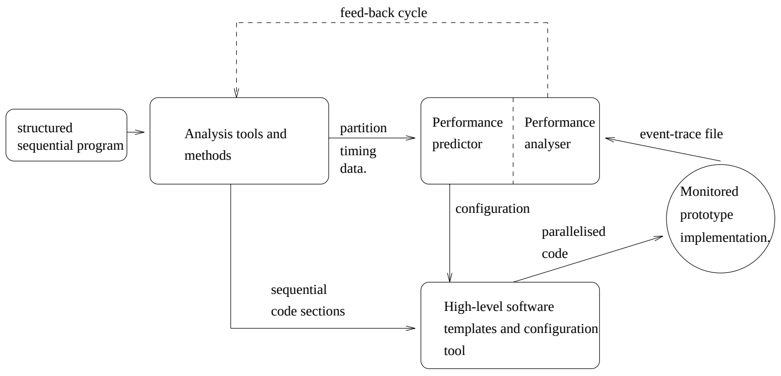


Fig. 1. The PPF Design Cycle

3 PPF Outlook

In the vision and image-processing fields, the choice of parallel hardware has narrowed, since SIMD-style machines are not generally available — not because of any particular difficulty with the design but because a critical mass in the market place has not been achieved.[1] Similarly, topology was a source of considerable academic debate when store-and-forward communication was predominant amongst first generation message-passing machines. With the advent of generalized communication methods such as the routing switch, the 'fat' tree, wormhole routing, and virtual channels the situation appears less critical.

Communication latency is also apparently becoming less of an issue. The Bulk Synchronous Parallelism (BSP) programming model, which has been widely ported to recent machines, adopts a linear model of piece-wise execution time, characterised by a single network permeability constant. Message-latency variance can be reduced by a variety of techniques, such as the message aggregation used on the IBM SP2 SMP. Routing congestion can be palliated either by a randomized routing step, or by a changing pattern of communication generated through a latin square. It appears that a single metric, such as the mean latency, may suitably characterize PPF communication performance. The advantages for the system designer were this to be the case is that the behaviour of the algorithm becomes central, with communication characteristics remaining stable and decoupled from the algorithm.

4 PPF Taxonomy

The postcode example given in Section 1 may give the impression that PPF pipelines are invariably asynchronous, though the need to collect a complete postcode before commencing the dictionary search is a form of synchronous constraint. Throughput and traversal latency depend on whether there is a synchronous constraint on a pipeline stage, which, in PPF, can be: an algorithmic constraint, i.e. completion of a given set of jobs; one or more feedback loops; a folded-back pipeline. An ordering constraint is a feedback loop that can be confined to one stage, but generally feedback constraints extend over at least one stage. A folded-back pipeline is a way of avoiding synchronisation overhead by using the same pipeline stage for more than one step in the processing, but ineluctabley this strategy will only partially succeed because partitioning is imperfect and because even a perfect partition may have variance due to data-dependency. PPF specifies the elementary types of software pipeline (Fig. 2) as: a linear pipeline; a dual pipeline; a pipeline with feedback; a folded pipeline; and combinations of any of the four previous pipeline types. Input to the complete pipeline is separately specified as either being instantaneously available or modelled by an input stream distribution, typically exponential.

A pipeline is split into asynchronous and synchronous segments. Examples of splittings are given in Fig. 3. A form of pipeline process algebra is possible.

[1] MasPar Inc. and their word-oriented machines and Cambridge Memory Systems, who support the DAP bit-serial machine, remain in active trading.

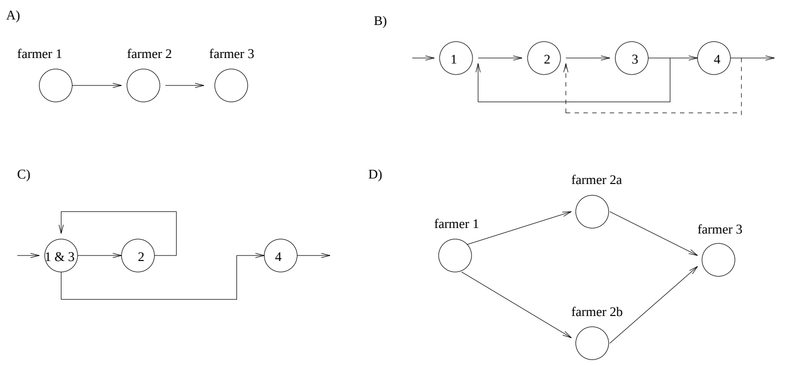


Fig. 2. Elementary Pipeline Types: A) Linear B) Simple feedback C) Folded D) Dual

A segment is a set of adjacent stages. A segment need not be a proper subset of the whole pipeline. A synchronous segment may contain any finite number of complete asynchronous segments and optionally one additional part of an asynchronous segment. An asynchronous segment is the largest set of adjacent stages, other than the trivial case, that can be formed at any one site in which no synchronous constraints apply. An asynchronous segment is capable of being simulated and any single stage may also be simulated. Synchronous segments must contain all synchronous constraints to ensure that the segment is self-contained but cannot include any constraints which are not needed to make the segment self-contained. Notice that a synchronous segment may contain another synchronous segment as a subset.

For the purposes of the predictor tool, a feedback-loop can be replaced by a nominal pipeline stage which models the delay distribution experienced by the succeeding stage (Fig. 4). A nominal stage, which can be synchronous or asynchronous, will not correspond to a stage in the implemented software pipeline. A folded-back pipeline can also be unwrapped again to form a linear pipeline by means of first replacing the folding and then providing a nominal stage. By alternating simulation and analytic predictions the performance of a complete pipeline can be built up in a piecewise fashion. In the pipeline of Fig. 3(C), after first replacing the feedback loop by a nominal stage, the performance metrics for the first segment are found. The throughput characteristics act as inputs to the first asynchronous segment proceeding in the direction of data flow, which is simulated. The output metrics of the first asynchronous segment are available for subsequent segments. The maximum latency is additive between pipeline segment. Notice that the ubiquitous central-limit theory in various forms suggests that the result of concatenating random variables, the service times, across a number of stages is a Gaussian distribution, which is determined solely by mean and variance.

5 Implementation of the Simulator

Fig. 5 shows a sample screen from the simulator, which is implemented for portability through the Java 1.1 AWT. The number of pipeline stages and jobs to be completed as well as the job input type and arrival rate are entered initially. Within any one stage, jobs can be grouped into tasks. The user enters: the per-stage statistical distributions and parameters as estimated from test runs; the number of processors; and their performance characteristics. The interconnect parameters are also adjustable. If algorithmic parallelism is used then each worker may output work according to a different distribution. The scrollable pipeline display animates a discrete-event simulation running as a background thread. The display includes running indication of minimum, maximum and mean pipeline traversal latency. Throughput details, including a measure of distribution spread, are also included. Simulation time, which is adjustable, is shown in counter or clock format. The display can be zoomed into to show activity on individual pipeline stages. The state of the pipeline can be displayed, e.g. the total idle time to date of any worker process is to be available by clicking on that worker.

The simulation moves forward in time according to the global minimum remaining service time for any task. A preliminary, two-stage, version of the simulator used local minima on a stage-by-stage basis, which correctly calculated the output statistics but did not preserve correct event ordering on the screen. To keep a record of job latency a job is tagged with its time to date in the pipeline along with any remaining time at an individual stage. The simulation program also accounts for differing numbers of jobs per task at each stage of the pipeline. Interstage buffer queues are represented internally by calls to the vector class library, which enables dynamic data structures. At each time update point, the latency times of all jobs in a buffer are incremented. To give an impression of parallelism on a sequential machine using a slower semi-compiled language, communication is animated by changes in the colour and size of the links resulting in a persisting display.

6 Analytic Prediction

PPF is intended to develop systems that can guarantee or adapt to specifications. A principle problem is to determine maximum latency given data-dependent behaviour. The maximum latency distribution is the output distribution for algorithmically synchronous pipeline stages, i.e. latency and throughput are bound together. Distribution spread occurs even on regular problems within synchronous pipeline segments as there will still be system noise [14]. The obvious analytic technique, queueing theory, is generally confined to exponential arrival distributions. A further serious detraction is that the distributions of waiting times are not easily found, though we have made some progress with delay-cycle analysis which will estimate variances as well as means. The waiting time distribution is needed to find maximum latency due to buffering in

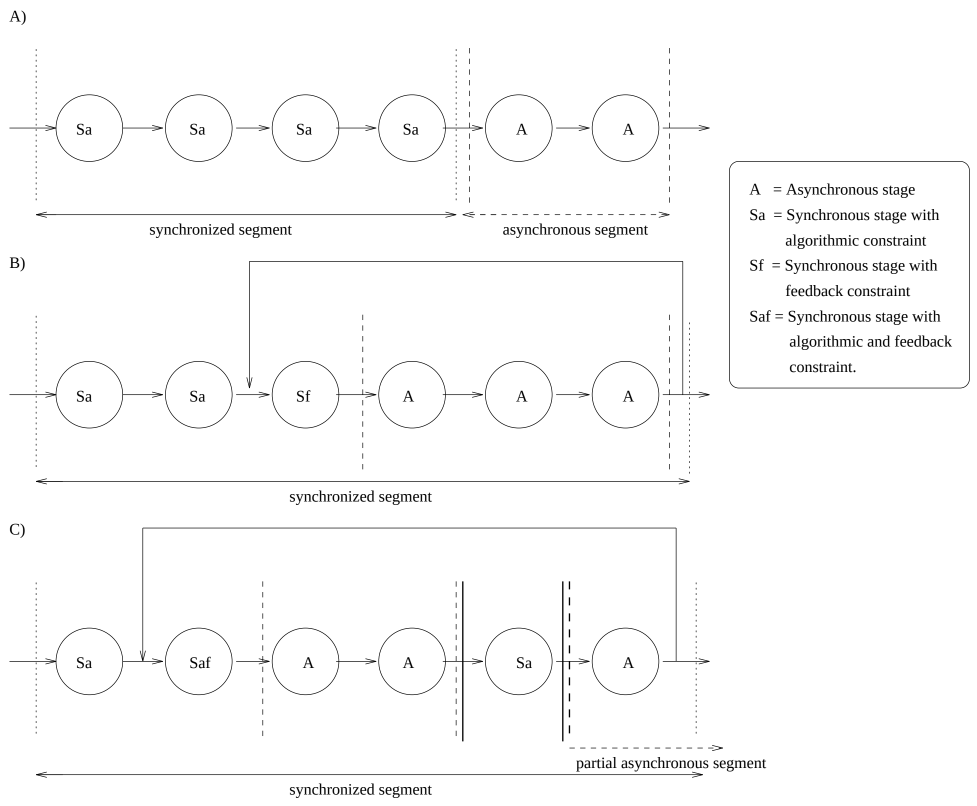

A)
Sa
Sa
Sa
Sa
A
A
synchronized segment
asynchronous segment
A = Asynchronous stage
Sa = Synchronous stage with algorithmic constraint
Sf = Synchronous stage with feedback constraint
Saf = Synchronous stage with algorithmic and feedback constraint.
B)
Sa
Sa
Sf
A
A
A
synchronized segment
C)
Sa
Saf
A
A
Sa
A
partial asynchronous segment
synchronized segment

Fig. 3. Pipeline splittings: A) Disjoint segments B) Singly Nested segments C) Multiply Nested Segments

asynchronous pipeline segments. However, order statistics have been shown in simulation and in timing experiments [15] to give suitable statistics for maximal behaviour.

Because of the changed nature of parallel hardware, as envisaged in Section 3, it becomes possible to use order statistics whereas previously linear programming or queueing theory or both appeared necessary [16]. This paper considers just one way of using order statistics. Taylor expansions of all order statistics [17] are available from $X_{i:p} = F^{-1}(U_{i:p}) = G(U_{i:p})$, when the $E[U_{i:p}]$ are $i/(p+1) = p_i$, resulting in:

$$X_{i:p} = G(p_i) + G'(p_i)(U_{i:p} - p_i) + \frac{1}{2}G''(U_{i:p} - p_i)^2 \cdots, \qquad (1)$$

with $G' = d(G(u))/du|_{u=p_i}$, $u = F(x)$. For example, with F a logistic distribution, $p = 20$, after taking the expectation of (1), the results are plotted in Fig. 6

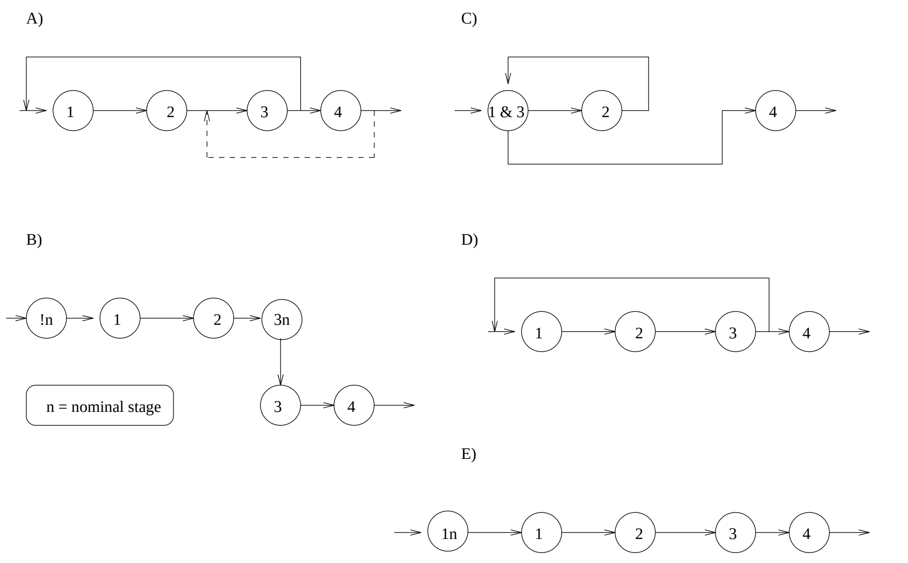
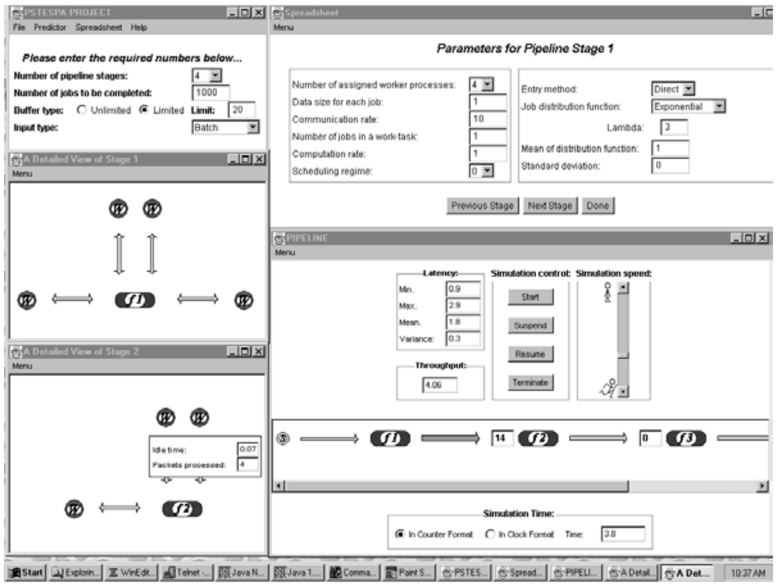


Fig. 4. A) Two simple feedback loops B) Replacement by nominal stages C) Folded-back pipeline D) After unwrapping E) After substituting a nominal stage.



Please enter the required numbers below...

Number of pipeline stages:	4
Number of jobs to be completed:	1000
Buffer type:	Unlimited [✓ Limited] Limit: 20
Input type:	Batch

Idle time:	0.07
Packets processed:	4

Parameters for Pipeline Stage 1

Number of assigned worker processes:	4
Data size for each job:	1
Communication rate:	10
Number of jobs in a work-task:	1
Computation rate:	1
Scheduling regime:	0

Entry method:	Direct
Job distribution function:	Exponential
Lambda:	3
Mean of distribution function:	1
Standard deviation:	0

Latency:

Min.	0.9
Max.	2.9
Mean.	1.8
Variance:	0.3

Throughput: 4.06

Simulation control: Simulation speed:

14 0

Simulation Time: [✓ In Counter Format] In Clock Format Time: 3.8

Fig. 5. Sample Screen Display for the PPF Simulator

for odd values of p. The advantage of this procedure is twofold: the amount of work that needs to be reserved at each scheduling round in order to minimize the final idling time, assuming a monotonically-reducing task-size duration scheduling regime, can be found by observing the shape of the cdf of $\{E[X_{i:p}]\}$; and likewise the performance degradation from any ordering constraint on task output is estimated by the expectation of the order statistics expectation cdf.

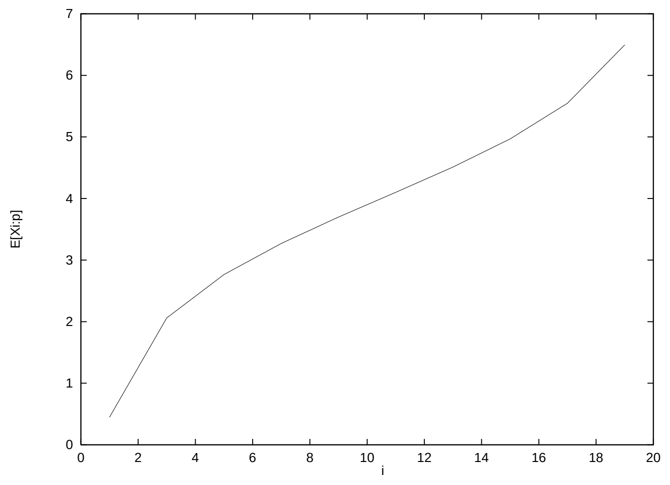


Fig. 6. Approximate Mean Order Statistics for the Logistic Distribution

7 Conclusion

A coherent scheme, PPF, for developing parallel systems has been introduced. The system design element is supported by a performance prediction tool which gives an abstract graphical representation of pipelines formed from independent parallel components. The complete-pipeline performance is built-up by combining simulated and analytic predictions for respectively asynchronous and synchronous pipeline segments. Input to the predictor is found by test runs of software developed in a sequential environment, which implies that the first stages of development can occur without purchase of parallel hardware. In fact, the path to full implementation is further eased because an intermediate step is introduced whereby the system is prototyped on general-purpose parallel kit, the results being fed-back for comparison with predictions. Order statistics are used to form estimates of maximal pipeline segment output characteristics. A discrete-event simulator, written in Java, has been animated to visualize pipeline activity. The event traces from typical runs can be animated for graphical comparison.

Acknowledgement

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

References

1. A. C. Downton. Speed-up trend analysis for H.261 and model-based image coding algorithms using a parallel-pipeline model. *Signal Processing: Image Communications*, 7:489–502, 1995.
2. H. P. Sava, M. Fleury, A. C. Downton, and A. F. Clark. A case study in pipeline processor farming: Parallelising the H.263 encoder. In *UK Parallel '96*, pages 196–205. Springer, London, 1996.
3. A. Çuhadar, D. Sampson, and A. Downton. A scalable parallel approach to vector quantization. *Real-Time Imaging*, 2:241–247, 1996.
4. A. Çuhadar, A. C. Downton, and M. Fleury. A structured parallel design for embedded vision systems: A case study. *Microproc. and Microsys.*, 21:131–141, 1997.
5. M. Fleury, A. C. Downton, and A. F. Clark. Pipelined parallelization of face recognition. *Machine Vision and Applications*, 1998. submitted for publication.
6. M. Fleury, A. C. Downton, and A. F. Clark. Co-design by parallel prototyping: Optical-flow detection case study. In *High Performance Architectures for Real-Time Image Processing*, pages 8/1–8/13, 1998. IEE Colloquium Ref. No. 1998/197.
7. M. N. Edward. Radar signal processing on a fault tolerant transputer array. In T.S Durrani, W.A. Sandham, J.J. Soraghan, and S.M. Forbes, editors, *Applications of Transputers 3*. IOS, Amsterdam, 1991.
8. S. Glinski and D. Roe. Spoken language recognition on a DSP array processor. *IEEE Transactions on Parallel and Distributed Systems*, 5(7):697–703, 1994.
9. A-M Cheng. High speed video compression testbed. *IEEE Transactions on Consumer Electronics*, 40(3):538–548, 1994.
10. J. Spiers. Database management systems for parallel computers. Technical report, Oracle Corporation, 1990.
11. S-Y. Lee and J. K. Aggarwal. A system design/scheduling strategy for parallel image processing. *IEEE Trans. on PAMI*, 12(2):194–204, February 1990.
12. D. P. Agrawal and R. Jain. A pipeline pseudoparallel system architecture for real-time dynamic scene analysis. *IEEE Trans. on Comp.*, 31(10):952–962, 1982.
13. K. M. Nichols and J. T. Edmark. Modeling multicomputer systems with PARET. *IEEE Computer*, pages 39–47, May 1988.
14. M. Dubois and F. A. Briggs. Performance of synchronized iterative processes in multiprocessor systems. *IEEE Trans. on SW. Eng.*, 8(4):419–431, July 1988.
15. M. Fleury, A. C. Downton, and A. F. Clark. Modelling pipelines for embedded parallel processor system design. *Electronic Letters*, 33(22):1852–1853, 1997.
16. M. Fleury and A. F. Clark. Performance prediction for parallel reconfigurable low-level image processing. In *International Conference on Pattern Recognition*, volume 3, pages 349–351. IEEE, 1994.
17. F. N. David and N. L. Johnson. Statistical treatment of censored data part I fundamental formulæ. *Biometrika*, 41:228–240, 1954.

This article was processed using the LaTeX macro package with LLNCS style