ANALYSIS PREDICTION TEMPLATE TOOLKIT (APTT) FOR REAL-TIME IMAGE PROCESSING

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

University of Essex, UK

1 Introduction

Image-processing applications often must not only provide accurate results but also meet real-time exigencies. This suggests a sensible division of labour, since in practice algorithmic designers are firmly wedded to workstations or PCs. Real-time acceleration in machine vision [1] can be provided either by specialist hardware such as field-programmable gate arrays (FPGAs) or by parallel processing, neither of which are convenient for algorithmic development. Radar, vision, and some varieties of speech-processing all commonly have a strong pipelined structure. The Analysis Prediction Template Toolkit (APTT) provides a seamless way to model graphically an image-processing pipeline before purchase of the target hardware, and subsequently construct a parallel application from the developer's code, without significantly compromising algorithms. Testing on a handwritten post-code recognition application has confirmed agreement to within 10% between simulation and target system for pipeline traversal latency and throughput, at the same time allowing the designer to gain an intuitive feel for the behaviour of the application. Analytic results are available [2] to refine the prediction, though the simulation is already suitable for cross-architectural comparisons of asynchronous pipelines.

2 Design cycle

APTT supports a design cycle, Fig. 1, aimed at producing pipelines with a single primary data flow constructed from a set of parallel data farms. Sequential bottlenecks are masked by single processor stages, possibly accelerated. Profiling the development code enables a tentative allocation of functions to different stages of the pipeline based on mean timing ratios. Normally, the potential for loop parallelism will determine the per-stage weighting of processors. A set of inner-loop timings enable distribution fitting by statistical or other means. Identifying a job as one iteration of the inner loop, jobs can then be grouped into tasks. Note that the grouping results in a change in distribution as the addition of distributions is a convolution. Two classes of problem are catered for: vision applications where the semantic content pre-determines the extent to which an optimal grouping can be made; and low-level image processing where there is generally more latitude in the way image segments can be grouped.

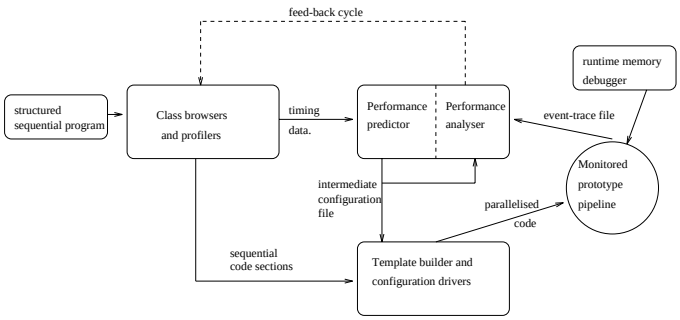

Figure 1: Design cycle

To arrive at an optimal arrangement also requires consideration of dynamic effects such as communication bandwidth both across the pipeline backplane and in subsidiary farms. Buffering can be added, at a cost, to enhance the bandwidth and smooth work flow. Though synchronous pipelines are best dealt with analytically, asynchronous pipelines are difficult in the general case, requiring the statistics of large deviations, which makes a discrete event simulation an attractive and malleable tool.

Integrated with APTT is a parallel-application generator, taking as input high-level descriptions of shared data structures. The other input is inner-loop sequential code sections after which boiler-plated parallel code can be output. Built-in trace instrumentation and communication calls are intended for two generic targets: a network or cluster of workstations, or a modestly parallel machine. The former is suitable for verification of correct working of the application and the latter is suitable for performance testing. By means of the trace, the simulation prediction can be compared to actuality. As with the configurer tool, an intermediate form is employed for the application code so that communication primitives can be generated for a variety of machines.

Both simulation (prediction) and trace (analysis) are presented in a similar graphical format, though

the former is intended as an abstract model while the latter is a physical model. A key difference is that feedback is explicitly represented in the analysis tool while the predictor reduces feedback to a flat representation.

3 Worked example

To check our work, simulated results were correlated with an application [3], previously implemented on an eight module message-passing parallel machine, the Transtech Paramid. Recognition of handwritten postcodes with appropriate weightings can be split into three stages: identification of features within each character of a postcode, classification of those features to form a ranked list of candidate characters, and a search to match candidate postcodes against a dictionary of available postcodes. This moderately-sized system, 4.5k lines of code, employs multiple algorithms with data-dependency introduced in the final stage (UK postcodes tested could have 6 or 7 characters in the ratio 145:155), achieving 80% recognition accuracy. If the throughput constraint of 10 postcode/s or the maximum latency constraint of 8s were to be exceeded then the computer processing would not keep up with the mechanical conveyor belt which transports the mail items.

The static timings are set out in Table 1. Timing a set of 300 (1945) postcodes (characters) and then applying separately Kolmogorov-Smirnov and chi-squared tests, established that the distributions of processing times were approximately deterministic (and not Gaussian as had been supposed before tests), while the final stage, assuming random ordering in the input file set-up to test recognition accuracy, was matched by a Bernoulli distribution. In the original implementation, interstage buffering had been set by trial-and-error at 20 slots, while the local input buffer sizes were 10 slots. The aim had been to find the best throughput if jobs were instantaneously available in the worst case scenario.

Table 1: -Input parameters timed on target processor, an i860, (a) 6- (b) 7-character postcode

stage	job time(s)	distribution	max. data transfer (bytes)
1	0.028	constant	2119
2	0.036	constant	40
3(a)	0.0045	bi-modal	112
3(b)	0.019	bi-modal	112

Each application has some special features. In the postcode application, differing postcode (task) sizes in the final stage occur which we bracketed by worst (all size seven) and best (all size six) cases. In Table 2, the worst-case estimates are compared with the implemented result with favourable accuracy. The 3:3:1 pipeline simulation is optimal, as had been suggested by preliminary static analysis. Note that one of the eight processor modules is reserved for feeding the test file to remove I/O dependency. Throughput is critical, while latency is well below the 8s requirement on the Paramid, though on earlier transputer-based machines latency was an issue. Though one might seek to apply a simulation capturing more of the computer system detail, experience has shown [4] that no greater accuracy necessarily results.

Table 2: - Comparison of simulated with timed results

worker ratio	run-time (s)	throughput postcodes/s	mean (s) latency	max. (s) latency
simulated				
3:3:1	26.20	11.43	1.02	1.27
2:3:2	28.65	10.47	0.57	0.86
1:4:2	56.90	5.33	0.78	0.79
2:2:3	35.90	8.36	1.13	1.28
3:2:2	35.90	8.35	1.24	1.60
implemented				
3:3:1	24.2	12.4	1.2	1.6
2:3:2	27.7	10.8	0.6	0.7
1:4:2	54.6	5.5	0.3	0.4
2:2:3	35.3	8.5	1.0	1.0
3:2:2	35.1	8.6	1.0	1.0

In the APTT simulation, varying the number of processors in the pipeline above and below the number in the Paramid, Table 3, established possible cost/performance tradeoffs and highlights the advantages of the simulation tool in allowing rapid and complete exploration of the design space of possible parallel solutions. The original buffer slot sizes were probably set too high as internal buffering was not found to be critical while interstage buffering could be reduced throughout to the postcode character size (seven slots).

Table 3: - Simulated results for a variety of pipelines

	Paramid		Dec Alpha (21064)	
pipeline worker ratio	run-time (s)	thruput pcodes/s	run-time (s)	thruput pcode/s
3:2:1	35.1	8.6	11.9	23.3
2:3:1	27.6	11.0	9.4	29.8
4:3:1	24.3	12.4	8.1	34.5
3:4:1	23.9	12.6	8.3	33.3
3:3:2	23.5	12.8	8.0	34.7
3:4:2	18.6	16.1	6.3	43.9
4:4:2	17.7	17.0	6.0	46.3
2:5:1	27.3	11.0	9.1	30.0

When testing a real-time application it may be difficult to remove the effect of system-dependent I/O except by pre-loading a file. However, latency also arises if jobs are blocked on requesting entry to the

pipeline. In order to judge the value of the simulation in that respect, after loading the file, assuming Poisson statistics for job arrival rates, a delay was artificially generated in the implemented version. In some operating conditions, latencies longer than 8s may arise due to the additional possibility of multiple arrivals while the pipeline is blocked.

4 Cross-architectural comparison

To allow the performance within APTT on one machine to be extrapolated to another we sought a simple but widely-recognised characterisation. A two parameter model of performance has now been applied to a variety of parallel architectures [5], though not apparently previously in a predictor tool. For example, in Fig. 2, which is a log-log plot, the Paramid reaches half its maximum bandwidth with messages of about 60 bytes (first parameter, established by linear regression) before reaching steady state (second parameter). In this case, the user need only know the message length and the target processor to project results.

Measurements on an individual Paramid processor, an i860, showed that a two parameter characterisation might be insufficient for computation as there was dependency on the computation kernel being performed, with additional cache effects evident. Fig. 3, showing results for four out of seventeen test kernels at full compiler optimisation, indicates two linear phases for some kernels where the vector length being computed stays within and steps outside the cache. However, it is not a difficult matter to store in a look-up-table the results for each machine and for each kernel. The user then selects a kernel, vector size, and processor to enable the performance tool to give a first-order approximation by means of scaling the computation times. This is likely to be more helpful for regular computations such as orthogonal transforms. An alternative characterisation is to use the computational intensity of the code, f, in units of flops/memory reference. Table 4 records steady-state performance (which is well below theoretically optimal performance), r_α^{ic} and r_α^{oc} being respectively in-cache and out-of-cache performance in units of Mflop/s.[1]

Applying the out-of-cache computational intensity test, a Dec Alpha (21064 at 175 MHz) server was found to scale over the i860 by a factor of 3.0 for $f = 5$ with a `-fast` compiler setting. As this is a load-dependent measurement, the arithmetic mean of five selected results was taken. Table 3 records the projected timings were 21064s to be substituted for i860s, otherwise keeping the system the same. The longer out-of-cache test figures were chosen because the lower resolution clock would otherwise effect the accuracy[2] though in-cache timings indicated a larger scaling particularly for higher values of f, indicating an efficient memory hierarchy. Low-resolution software clocks may be a deterent to the use of a processor in some hard real-time system, though not perhaps for soft real-time systems as herein.

Spatial filtering is an example of an image processing operation commonly performed with integer operations, whereas benchmarking kernels, being derived from the numerical analysis community, usually employ floating-point operations. There would appear to be a need for a set of agreed kernels specifically for image-processing tasks.

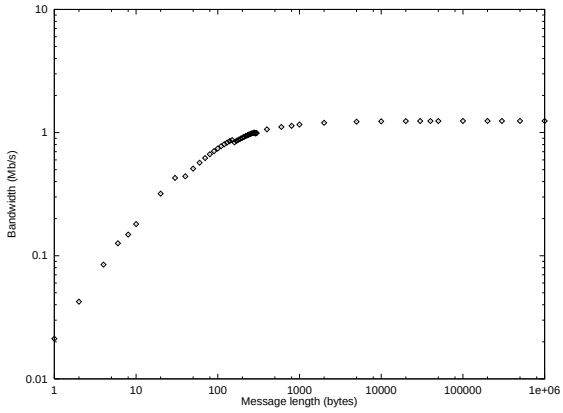

Figure 2: Point-to-point communication performance for the Paramid

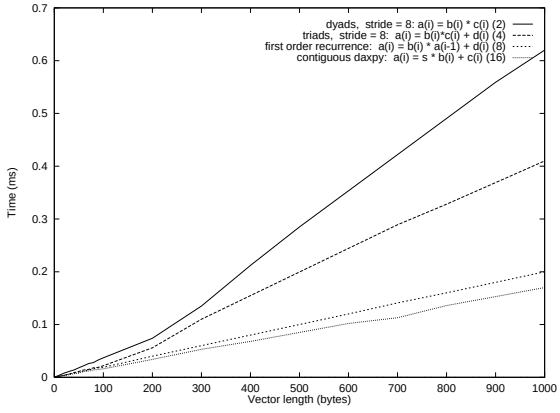

Figure 3: Selected computation performance for the Paramid

Table 4: - Paramid computational performance

f	1	2	4	5	6	8	9	10
r_α^{ic}	25.6	18.3	15.9	15.6	15.4	15.1	13.2	13.3
r_α^{oc}	18.1	16.1	15.7	14.8	14.7	14.9	15.0	13.2

[1] The out-of-cache measurements arise by using vector lengths designed to exceed the cache size and by causing a cache flush between tests.

[2] Clock resolution was $\sim$ 0.01s as opposed to $\sim$ 0.001s on the Paramid, mean timing error $\pm 6.1\%$.

5 Graphical representation

A graphical interface is the user's view of a toolkit, and in terms of person hours of design effort has been the most expensive part of the APTT development. Our design aims to exploit familiar user interface paradigms in terms of navigating data entry screens and utilising simulation and trace tools; thus reducing the user's learning time. The interface was written in the Java programming language, which has enabled a trivial port between Windows NT and Unix operating systems which would not have been possible with X-window software.

A problem with previous analysis visualisation tools, such as ParaGraph [6] written with X-lib calls, is that an animated display occurs, rather like a cartoon film. The user may find it difficult to establish a pattern. Moreover, in seeking generality, with twenty-four ways of presenting data, no structure to the tool's use was provided. An over-animated display also reinforces the sequentiality of the simulation whereas the pipeline represented has local and general parallelism.

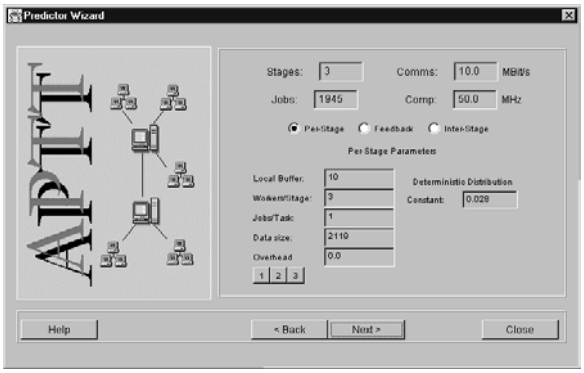

Figure 4: APTT data-entry 'wizard'

Fig. 4, showing a summary of statistics entered, is taken from the APTT data-entry 'wizard' which has a familiar look-and-feel to ease user adaptation. Fig. 5 shows a snapshot of the predictor running the postcode simulation. The pipeline backplane occupies the main window with details of the stage activity such as buffer and processor usage available from subsidiary windows. Processor activity is shown using colour by analogy with stop/go displays. Again using the linguistic associations of colour, the communication arrows change colour from black, through red to white to highlight 'hotspots'. The arrows also widen and contract. However, the cumulative mean bandwidth, not instantaneous bandwidth is displayed. The colour scaling is adjustable to centre on critical data rates as otherwise the variation across the whole bandwidth range is too low to show up. Latency is also indicated in a persistent display. Jobs are marked off at task boundaries, with the task latency determined by the slowest job. Though persistent displays convey more information, they need to be balanced with features marking progress, which is why the processor activity diagram and message motion arrows are included.

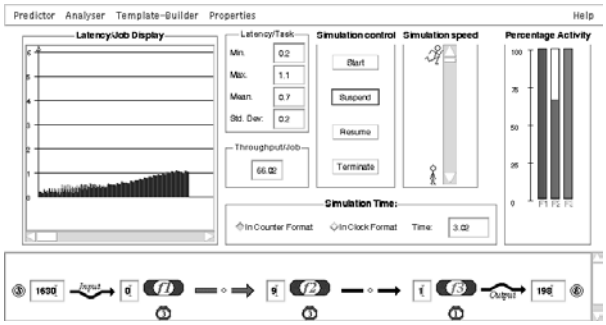

Figure 5: APTT predictor window

Fig. 6 shows the analyser window with the postcode trace running and the configuration set-up from an intermediary format file. As instrumentation is inserted transparently the display does not try to infer performance statistics other than run-times.

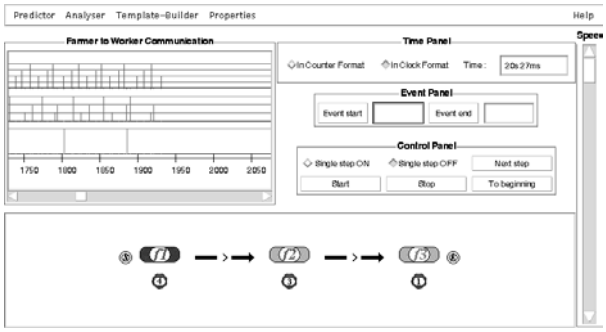

Figure 6: APTT analyser window

6 Constructing templates

The prevalence of PC networks has prompted work on constructing a suitable data-farm template implementing our generic abstract model [7] for real-time systems. Remote method invocation (RMI) is a high-level Java communication class library [8] suitable for large-grained applications. Using the Java vector class, which can be made to transparently grow and shrink as work requests are serviced, could remove the need to provide concurrent access management. Though a peer-to-peer communication mode is possible, the underlying semantics of RMI are client-server utilising remote procedure call (`rpc`). Our implemented design requires the data farmer to poll the remote worker processes acting as servers. Conveniently, this fits a polling queueing model [9] of performance estimation. Java's pre-emptive priority-based thread scheduling can be

adapted to provide a responsive structure, Fig. 7. However, `rpc` always comes with an overhead by reason of stub processes acting as intermediary communication interfaces, and additionally it is necessary to arrange that the farmer is not blocked until the remote invocation completes. RMI is also suitable for linking JavaBean software components within a framework and as such our template is consistent with a trend within industry-standard distributed software [10] towards standardised middleware and high-level object-oriented software architectures.

Data-farm multicast is present in our original model for applications such as the H.263 hybrid video encoder [11] where per image-row distribution of quantisation levels takes place. Broadcast is also a synchronisation mechanism for pipeline reconfiguration if workloads vary over time between the parallel stages. Reliable broadcast can be employed as an efficient method of implementing shared objects [12] which would allow co-operative forms of parallelism to take place within a pipeline stage, extending the range of applications suitable for parallel pipelines.

7 Conclusion

Data flow rather than control flow is the key issue in the design of real-time image-processing applications. Understandably, algorithmic designers do not wish to be concerned with the added problem of understanding how the details of a number of different implementation platforms affect the flow but do require the specification to be met. APTT is an integrated environment which has the potential to close this loop. An extended machine-vision example has demonstrated that accurate results are predicted for a constrained parallel-pipeline construction system. Running a simulation for sufficient time will establish whether maximal events, exceeding the real-time constraints, will occur in a way that a prototype may not because of physical limitations on test file-size. Cross-architectural comparisons requiring relatively simple machine/algorithm characterisations are possible by adapting computer benchmark methodology. A graphical display, provided it gives meaningful information, is a vital element in giving a feel for the application, building confidence in eventual implementation on a range of target systems.

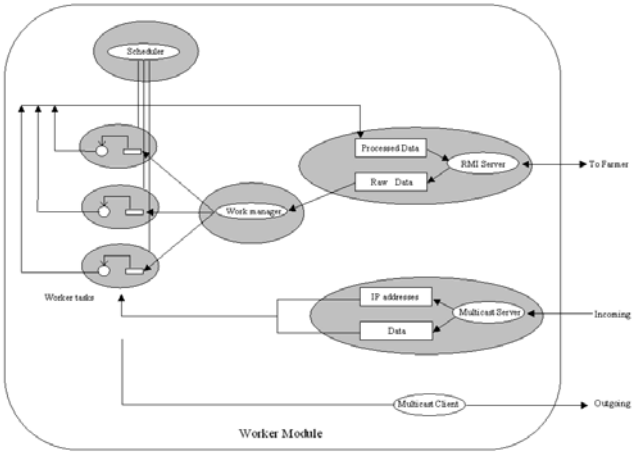

Figure 7: Java RMI Worker Module

References

[1] M. Fleury, A. C. Downton, and A. F. Clark, 1998, Co-design by parallel prototyping: Optical-flow detection case study. "IEE Colloq. High Performance Architectures for Real-Time Image Processing", 8/1–8/13

[2] M. Fleury, A. C. Downton, and A. F. Clark, 1997, Modelling pipelines for embedded parallel processor system design, Elec. Lett., 33(22), 1852–1853

[3] A. Çuhadar, A. Downton, and M. Fleury, 1997, A structured parallel design for embedded vision systems: A case study, M. & Microsys., 21, 131–141

[4] T. Delaitre, *et al.*, 1998, EDPEPPS: A toolset for the design and performance evaluation of parallel applications, Euro-Par'98, LNCS 1470 113–135

[5] R. W. Hockney, 1996, "The Science of Computer Benchmarking", SIAM, Philadelphia.

[6] M. T. Heath and J. A. Etheridge, 1991, Visualizing the performance of parallel programs, IEEE S/W, 8(5), 29–39

[7] M. Fleury, H. P. Sava, A. C. Downton, and A. F. Clark, 1996, A real-time parallel image-processing model. IPA'97, i174–i178.

[8] E. Burris. The RMI package, 1997, "Java Unleashed", 426–451. Sams, Indianapolis, IN

[9] M. Fleury and A. F. Clark, 1994, Performance prediction for parallel reconfigurable low-level image processing. ICPR, iii349–iii351

[10] R. Orfali, D. Harkey, and J. Edwards, 1996 "The Essential Distributed Objects Survival Guide", Wiley, New York

[11] H. P. Sava, M. Fleury, A. C. Downton, and A. F. Clark, 1996, A case study in pipeline processing farming: Parallelising the H.263 encoder. UK PARALLEL '96, 196–206. Springer, London

[12] H. E. Bal, M. F. Kaashoek, and A. S. Tannenbaum, 1992, Orca: A language for parallel programming of distributed systems. IEEE Trans. SE, 18(3), 190–203