Parallel Reconfiguration in an Image Processing Context

M. Fleury, L. Hayat, and A. F. Clark

Dept. E.S.E., Essex University, Wivenhoe Park, Colchester, CO4 4SQ, U.K

tel: +44 - 1206 - 872795

fax: +44 - 1206 - 872900

e-mail fleum@essex.ac.uk

Summary

Reconfiguration is considered in the context of image processing applications. Dynamic switching is the basis of experiments into hardware reconfiguration described in the article. Implementation issues are detailed. Methods of analysing performance for the reconfigurable case are adumbrated. A discussion ensues on the practical problems of providing a usable system with reconfiguration provision.

1 Introduction

Image processing (IP) is an area where there is much interest in parallel architectures [1, 2, 3, 4, 5] because of the variety of processing requirements for images. Another motivator is the computationally-intensive nature of the processing. A moderately-sized task, to filter and extract features from a 512×512 pixel image at 25 frames per second, requires a throughput of upwards of 600 MOPS. A pre-processing (or low level) stage is usually necessary. Computation at this stage is largely deterministic and only regionalized communication is necessary (if overlapped borders are not used) so that data parallelism might be used. Subsequent (high level) processing might be data-dependent. Less regular communication patterns occur, involving the exchange of semantic or meaningful entities which are held in more complex data structures. An example of a low-level process is finding edges within an image, which might be accomplished by passing a set of Sobel spatial masks over every pixel [6]. This processing is localized to the nine pixels covered by a Sobel mask. Many other IP tasks can also be reduced to localized operations [7]. However, there are some low level activities that cannot be confined to localized communication, notably orthogonal transforms such as the Fast Fourier Transform (FFT), but also coordinate transforms used for image registration such as image rotation. An example of high-level processing is to use any edges found by Sobel masks in order to identify objects, in which case global communication will be necessary, though with reduced data. (It is also possible to distinguish an intermediate stage of processing in which higher level entities are constructed but without involving semantic content. The Hough transform, for example, can be used to identify lines from edges.). [8]

contains an analysis of the *desiderata* of a flexible image processing system, including the provision of the following: an expert system to plan processing; specialist hardware; a range of alternative routines for the same process; and the use of extra processors for speculative computation.

Attention has been given to special-purpose architectures such as the class of pipelined machines, examples of which are the Cytocomputer [9] and PETAL [10]. These computers operate on raster streams of pixels at video rates. Such architectures do not supply general purpose solutions. In [11], the problem is seen as one of being able to switch from SIMD-style processing in the pre-processing stage to MIMD-style computation in later stages. A design is given for a general purpose switch so as to increase the utility of the SIMD stage. It is true that early interest centred on SIMD machines [12], including machines that have combined both SIMD and MIMD components [13]. In our work, we have restricted ourselves in the main to message-passing medium-grained machines because of the problem of portability and the desire to avoid a restricted range of algorithms. The variety of commercially available SIMD machines is limited, an example being the MasPar [14]. Though significant, this article is not concerned with tightly-coupled multiprocessors or machines employing a global address space. Another approach to coping with the data quantities endemic to IP is resolution reduction, for which one might consider pyramidical machines, for instance in [15].

Reconfigurability could provide one solution to the implicit discussion on suitable architectures. The development of a modular processor, the transputer, has stimulated the discussion by providing accessible technology. A similar enterprise is that of a modular switch, for instance the Torus routing chip [16]. It is a truism of parallelism that improvements in processing speed will come from being able to combine processors rather than further technological development of the uniprocessor, which had reached a plateau in 1983 [17]. Though since 1983 there has been renewed activity, as [18] makes clear the boundaries of performance in regard to IP are close to and limited by the finite speed of light. Since the transputer, other modular processors have followed, such as the C40 with six links, which also has an increased per-link bandwidth. Any shortcomings of the hardware can be ameliorated by combining

processors, for instance the combination of transputer and the pipelined superscalar i860 [19] in the Paramid machine [20], or the use of Elan communications co-processor with either a SPARC or Fujitsu μVP vector processor in the Meiko CS-2 [21]. Another level of modularity is the processor board such as the eight-transputer B012. On this board, the transputers are connected by a 32×32 crossbar switch, the ubiquitous C004. Another transputer, usually a lower cost processor such as the T212, is needed to make the crossbar connections. Modularity can go hand-in-hand with reconfigurability. The advent of the virtual parallel machine, employing communication means that are transparent to the user, though convenient, is not necessarily an efficient solution and may not be cost-effective because of the extra hardware requirements, particularly for an embedded system. A case in point [22] is an FFT algorithm on a processor mesh in which the use of cut-through (or its variant wormhole routing) communication is no more efficient than store-and-forward communication. In [23] reconfiguration is approached from a theoretical standpoint, based on the types of parallelism present and employing a scheme based on centralized control of configurations. There should also be room for consideration on a pragmatic basis, which is this article's approach. Briefly, three forms of hardware reconfiguration using transputers can be discerned:

Static reconfiguration Processors are reconfigured at load time, but maintain a configuration throughout the lifetime of a program. An example is POPA [24], a transputer-based machine that uses a single-stage interconection network to form user domains and a three-stage interconnection network to a pool of discs and graphical devices.

Quasi-static reconfiguration The topology of the network or sub-network is changed at pre-determined points or phases during the course of a program's execution. An example is C-NET [25], a programming environment for the Telmat T-node, which has a similar architecture to the Parsys Supernode described in Section 3.2.

Dynamic reconfiguration Reconfiguration of communication links can occur at arbitrary times during a program. In [26] there is a distributed connection scheme, implemented on the ParSiFal T-Rack, a precursor of the Supernode, which also included a program-

mable switch.

There are no transputer examples of a fourth type of reconfiguration, low-level changes to the data-widths of the computer modules and their instruction sets [27]. We employed static reconfiguration and dynamic reconfiguration in our experiments.

The rest of this article is organized as follows. Section 2 considers ways of synchronizing reconfiguration. Various practical aspects and experiences arising from our work are described in Section 3, where a generic image processing model was developed using dynamic switching. Finally, Section 4 summarizes and draws some conclusions.

2 Arranging Reconfiguration

Synchronization is the main overhead as well as the main problem to solve. For quasi-static reconfiguration, the problem is related to that of arranging graceful termination in an asynchronous MIMD network [28]. Before a reset can occur all communication must have ceased and all processors must have reached a pre-arranged *rendezvous* in the algorithm. A number of solutions are possible. For instance, when reconfiguration is signalled from a central source all processes should cease communication and retain any expected messages. Each process then signals that it has entered a wait state. A further signal is broadcast subsequently to indicate the end of the reconfiguration phase. Another technique is the circulation of a token between processes [29]. The signal itself may be broadcast on a control bus. The T9000 transputer has a number of control lines, which function independently of the CPU, that might be used for this purpose. A hierarchical arrangement of processes may be a way to manage the synchronization of reconfiguration [30]. This might for instance occur by adding a configuration manager process to sit above a client-server hierarchy. This arrangement is apt when several server processes occupy one processor. A configuration manager process would control a router process which in turn controls all external communication to and from the processor (Figure 1). An object-oriented design methodology, though not necessarily an object-oriented programming language, is suited to this task. The type of reconfiguration

chosen is dependent on the perceived overheads. If parallel languages are to be extended to include reconfiguration, both quasi-static and dynamic reconfiguration should be considered as either could be appropriate to the underlying hardware.

3 Reconfiguration in Practice

This section describes aspects of the authors' experiments into reconfiguration. The algorithms investigated were chosen to be representative of low-level IP tasks rather than necessarily being suitable for parallel implementation. (Our work formed part of a larger project to investigate reconfiguration, with colleagues at other institutions considering other classes of algorithms.)

3.1 Background to our Work

One motivation behind our work was to investigate how usable parallel software software could be provided that would improve the perceived slow transfer of parallelism from academia to industry [31]. A way of doing this is to constrain parallelism to one programming paradigm, that of demand-based farming. In this paradigm, a central task is responsible for distributing (farming out) work to a pool of worker tasks. The returned processed work forms a request for more work. The advantages of the demand-based paradigm are:

- It is particularly suited to low-level and intermediate IP because of data parallelism. (It transpires that the low-level algorithms involving global communication can be performed by our generic arrangement (Section 3.3)).
- It allows the crucial transfer problem potentially to be solved by overlapping the communication of work with its computation. Where efficient I/O is provided to a central source, this can also be overlapped. The time for transferring the program code to the PEs remains as an overhead, but in batch mode this can be amortized over a number of images.

- It avoids the computation of load-balancing algorithms which may require costly global optimization routines, which are not flexible anyway. There is also no guarantee that a load-balancing algorithm will take into account all relevant communication costs, as these may be difficult to determine by theoretical means.

- Software costs are reduced as the software may be written in a generic fashion. Apart from the communication harness, which can remain static, there are only two modules to write: the farmer and the worker.

A disadvantage of demand-based farming for IP is that the granularity of the work portions may be fixed to a quantum for some algorithms so that load-balancing is not ideal. This is the case when applying the method to the orthogonal transforms discussed in Section 3.3. For real-time applications which are not confined to one or two algorithms it is possible to concatenate a number of farms into a pipeline, when convenient system design methods are available [32].

In the case of images, image strips of a suitable granularity were sent out. As arrays are stored in row-major order on the transputer under 3L Parallel C [33], using strips caused the least data movement as well as allowing only the start address to be supplied to the communication primitives. (In one algorithm, convolution in the frequency domain using overlapped blocks, a tiled data decomposition was necessary, but this is a specialised algorithm applicable when very large images are required). While "excessive parallelism" [34] (to coarsen the granularity) might be used where internal concurrency is available and does not cause large overheads, one should consider machines (such as the i860 as used on the Paramid) which cannot support more than one process. In fact, our arrangements were largely one computation task per processor.

3.2 Using Dynamic Link Switching

A particular problem that arises with operating in this mode is loading work onto the farm if more than a few PEs are used. For simplicity, a linear chain was considered. It has been

shown that the optimum topology considering the valency of the transputer, a ternary tree, gives little improvement except for large numbers of PEs [31]. The work of PEs nearer the farmer task is disrupted by messages passing back and forth to PEs further down the chain. If the chain is loaded by giving priority to tasks nearer to the farmer task, then remote tasks may suffer work starvation, while the tasks with first call on any work are disrupted the most. If priority is given to passing work to tasks at greater distances from the farmer task then latency will arise unless work buffering is provided. However, if more than one buffer is employed, work starvation can arise (as worker tasks with higher priority accumulate work to the detriment of other worker tasks). In [35], a similar problem was found when using PE trees, though then it was found necessary to dedicate complete transputers to communication. The solution taken was to link each transputer to a switch, servicing each transputer from a central source.

An alternative hardware solution to the problem of increasing the switching capability on PEs nearer the root of the tree is to employ a fat tree arrangement, whereby more switching elements are used the further up the tree, as in the Meiko CS-2 network [21]. The fat tree also increases the bandwidth nearer the root, while the dynamic switching arrangement will somewhat decrease the bandwidth instantaneously available. However, as indicated subsequently, the use of buffering can alleviate the bandwidth problem. In our experiments, links were not switched between single transputers but between small chains or trees because it was recognized that there existed a balance between the switching time to circulate across the heads of each tree and the overhead involved to send work down the tree. For instance, to process a 3×3 Valley operator (a spatial filter for locating luminance edges in an image [36]) on three processors arranged as in three groups of three took 2.65 sec. but if the processors were arranged singly the application took 3.00 sec. This came at the cost of constructing a simple communication harness. Even if an off-the-shelf harness were to be available that could cope with the reconfigurable case, this solution risks the possibility of deadlock because the internal buffer state is unavailable.

A Parsys Supernode [37] with 24 T805 transputers was used. In Figure 2, the machine is

shown in multi-user mode, when the user master process resides on one of the master PEs. The user domain is formed from a set of raw PEs (transputers with 4–8 Mbytes memory each). A network within the domain is subsequently booted-up with the user application code. The problem of I/O passing through a ring of master PEs before being directed to the network can be solved by reconfiguring so as to make a direct link to a frame-grabber module (not shown in Figure 2). Detailed consideration of the system revealed that centrally controlled switching was the limit of the reconfiguration possibilities as far as 'tweaking' of the hardware was concerned. Unlike the Telmat version of this machine, quasi-static reconfiguration could not easily be arranged without overhauling the operating system (O.S.), though static reconfiguration was available. Nevertheless, it was possible to reconfigure transputer links by link communication to the Switch Control Unit (SCU), also a transputer. Except for large-grained applications, reconfiguration instructions were prohibitively costly if they were made via more than one link. Even when a request was made directly to the SCU, because the machine normally worked in multi-user mode, a number of accesses to check and update connection tables were necessary on the part of the SCU program. Thus we found that a link could be swapped in 400μ sec.. In comparison, transmitting 1 Kbyte over one link in simplex mode took about three times this (duplex mode is more efficient on the transputer). Data packets of at least 3 Kbyte size are commonly used where an image is split into strips for regionalized processing. Due to cross-talk on the back plane, the transputer links were run at 10 MHz and not 20 MHz, the latter being usual for transputer applications.

The PE hosting the farmer on the present system uses two independently rotating links (one for send and one for receive) to connect to a number of small networks hosting the worker tasks (Figure 3). In the absence of the link used to receive processed work packages, a global buffer was provided at the head of each worker network. The farmer effects a reconfiguration by using one of its links to signal by means of a coded command to the SCU, which in turn is connected to a P1085 switch [38] (an electronic switch designed to allow all topologies of valency four to be formed). A look-up-table was used to translate from user to O.S. mapping of the links. The command structure for messages to the SCU had to allow for the possibility

of communication to other Supernodes (via a non-blocking Clos network) as well as other functions. Apart from checking that links were not being moved in another user's domain, another overhead arose because the SCU had to use timed communication over its links so as to allow recovery if a user program 'hung up'. It is this software burden that, it is considered, is the principal obstacle to general-purpose provision of reconfiguration. Apart from the master buffer at the head of the chain, local buffering was used to control in- and out-flows to the application process resident on each processor. A router buffer was also found to be helpful. One stratagem employed was to hold one packet in transit at each PE so that demand for work could in part be satisfied without returning to the farmer. Synchronization was not difficult to arrange because internal concurrency is provided for in firmware on the transputer. A thread (or lightweight process) within the task was polled by the farmer task once the link was in position at the head of the worker network. The thread was rescheduled and transfer of data to the farmer proceeded. No sychronization is necessary for the send as the receiving worker is a passive recipient. Trials determined that it was preferable to transfer small numbers of work messages at any one visit, despite the cost of switching.

A median filter [39] was implemented on the dynamically-switched system, employing individual binary trees, each of just three processors. The execution timings (i.e. without I/O times) are recorded in Table 1. By way of comparison, the same software was used on a single binary tree by simply omitting the link switching. When the third layer of the static binary tree was added, the execution timings saturated because of the burgeoning communication costs. As the size of the tree increased, the processors nearer the data-farmer did less and less work. While there remains the possibility that a scheme might be found to more adequately load-balance the static tree, this problem did not arise with a set of small trees, as used for the dynamically-switched solution.

3.3 Global Communication

Though the generic link switching mechanism was designed with spatial operations in mind, it was found that it was most effective for an image rotation program involving global com-

munication. (The method was also used for polar and log-polar coordinate transforms.) A more complex communication hierarchy was employed in this case whereby messages were sifted according to whether they were intra-tree or to a neighbouring tree (by horizontal connection in the manner of the Berkeley X-tree) or to a distant tree. (The arrangement is shown in Figure 4.) In the latter case, a data-manager, taking the place of the farmer task, used reconfigurable links to pass messages from one tree to another. A large array structure is needed to keep track of messages in readiness for passing onto a remote tree. This and other rotation algorithms are described more fully in [40]. The advantage of the global scheme over the data-farming one is that not all communication is by reconfiguration. Since the messages pass from process to process and not back and forth to a central source, it was also possible to overlap computation with communication to a greater extent.

A number of topologies, including trees, have been used to calculate the FFT [41]. For known architectures such as vector processors, it is important to tune the algorithm to the machine [42]. The hypercube is particularly suitable because each phase of the exchange algorithm can be linked to a dimension of the cube. However, it is not a scalable solution in the transputer case. Practical considerations such as the ability to use a variety of topologies may confine attention to the row-column (RC) algorithm for 2D FFTs, as complete rows or columns can be farmed out. Thus, for these algorithms, the same arrangement, but with dynamic link switching, can be used. Algorithmic interest then centres around providing a flexible solution using a mixed-radix algorithm. A 1D FFT can easily be mapped onto a 2D FFT by factorizing the transform size. Using data-farming for an FFT and other discrete transforms such as the Discrete Cosine Transform does have the detraction that a centralizing phase is necessary. It was found that the cost of transposing the image matrix, so that the last phase of the algorithm could be performed, formed a considerable portion of the processing time. In a concept-proving exercise, an intermediate reconfiguration phase for the transpose was introduced based on the recursive transposition algorithm [43]. The cost of forming a tree and distributing and collecting image blocks was too great. However, overall timings, including a centralized transpose, were at least as quick as solutions using either specialist

topologies or the use of a mesh to execute the FFT transpose algorithm. On the Novi system [44] employing a 4×4 mesh of transputers, a complex 2D-FFT of size 256×256 took a total of 13.63 sec. employing an RC algorithm, while a real-valued 2D-FFT of size 512×512 (when packing rows in the first pass meant 256 complex-valued FFTs of size 512 were performed) with only ten processors on the dynamically-switched system took 7.53 sec. The difference can largely be assigned to data loading and unloading which in the Novi system's case took most of the time (12.05 sec.) as the data were arranged so as to be equally distributed between the processors. On the Novi system, use of the vector-radix algorithm for 2D-FFTs, which reduces calculations (though not communication) by 25% [45], made little change to the timings for the algorithm. The De Bruijn topology of [46] gave a timing of 1.31 sec. for eight processors on a 1D complex image of size 2^{14}. This timing is roughly equivalent to the 1.99 sec. for a 256×256 real-valued image on the reconfigurable system when disparities in attainable transputer link speeds are accounted for. A set of timings for the 2D FFT on the dynamically-switched system can be found in Table 2. Two processors were used in each chain. The one-processor comparison appears to imply that a super-linear speed-up has occurred, but is actually due to the difference between the single transputer compiler and the parallel compiler. The timings for the parallel version do not take into account the considerable time taken to load the program code which, if the program was used for individual images, would be a deterrent to using the parallel version. If batches of images are processed, a pipeline arrangement with four farms will allow the load phase to be overlapped by subsequent phases. The first farm will load the image. The second farm will handle the row phase; the third farm performs the transpose; and the final farm is responsible for the column phase.

A further experiment was constructed to see whether it would be possible to load an image onto one processor, through a substitute farmer task, while the other farmer process managed work distribution and collection. The intention is that the two farmers alternate their roles in batch processing mode. Apart from the detailed synchronization needed between the two farmer tasks, it is also necessary to ensure that the process responsible for link reconfiguration suspends any process using the link as due to the limited valency of the transputer different

processes must use the same link. It was concluded that the management needed to complete the synchronizations was too complex unless it was provided as part of a reusable system.

An algorithm involving global communication which, though simple, remains difficult to effectively parallelize is image histogramming combined with equalization. If data-farming is used for histogramming then if the workers equalize the *same* data upon receiving a new grey-level mapping on broadcast from the farmer, then there is the possibility of an imbalanced load. In practice, it was found that a uni-ring arrangement [47] was as quick as a recursive doubling arrangement while avoiding centralization and preserving scalability. If the data are loaded from a single source, no arrangement using reconfigurable links proved quicker than a static topology, while loading costs overshadowed the time to execute the algorithm.

3.4 Performance Models

A performance model is possible for a dynamically switched network by extending existing theory. A method of finding the upper bound for communication on static n-ary trees employing store-and-forward communication is given in [48] by reference to the limiting bandwidth approaching the root. In [49], this method is extended to non-deterministic workload distributions by a mean-value analysis and can be used as a form of load-balancing if all communication costs are accounted for. Given the throughput from this method, it is possible [50] to find the per-visit waiting time (which is also the per-packet overhead if only one packet is removed at a time) at each sub-network as

$$W = \frac{mt_{switch}}{1-(m-1)r}$$

where m is the number of sub-networks, t_{switch} is the time to move a link from one sub-network to the next sub-network and r is the rate work approaches the head of each sub-network. r is given by $T(t_{setup} + t_{comm})$, where T is the message throughput, t_{setup} is the time to set a message up (after which the CPU can proceed independently to the communication engine). If mean-value analysis is not employed then queueing theory is a suitable choice when there is a stochastic character to the workload (or the workload is not known in precise terms). One

analysis of switching networks [51, pp. 137–151] used an extension of the Pollaczek-Khintchine formula which accounts for server vacation, but this depends on knowledge of the distribution of vacation times (*i.e.*, absence from the head of a sub-network). A solution for the rotation program test case described in Section 3.3 uses a polling type formula [52]

$$W = \frac{m\lambda E[S^2]}{2(1-m\rho)} + \frac{(1+\rho)mt_{switch}}{2(1-m\rho)},$$

where $E[S]$ is the expectation of the per packet computation time and ρ is the ratio of arrival rate to departure rate from the head of the sub-network.

It would seem that, as only one link can be moved at a time, the polling-type models will also form the basis for performance models of other dynamically-changed networks as future investigation will determine. In the case of phased reconfiguration, if the rearrangement of communication links can be overlapped with calculation, the computational cost of each phase gives the allowable overhead from switching the network. Figure 5 is an example of the analysis applied to our image rotation program. For the period when the processors were exchanging image line segments, measurement of the throughput was made. With the known message numbers, along with switching times, this allowed the estimate to track the actual times to rotate an image, provided the communication links were not approaching saturation.

3.5 Discussion

Reconfiguration by modification of the way the hardware is used is a solution that could be considered by software developers for the types of algorithms discussed in preceding sections. It is also a facility that computer manufacturers might consider providing as an explicit feature. In our experiments it was necessary to employ subterfuge to make it appear to the SCU program that a consistent static network was being established, before sending commands to alter the position of links during the program run. The main obstacle to dynamic or phased reconfiguration in electronically-switched networks is the absence of a hardware synchronization mechanism. This should not be a difficulty as monitoring on a global or regional basis is needed for fault tolerance purposes. A simple wired-OR arrangement, such

as that used in an early SIMD design [7], might be adequate. A higher-level facility could also provide global broadcast, which those machines employing wormhole routing already have the potential for by the replication of header flits (flow control digits). A user-accessible bus is adequate for phased reconfiguration. Designing user access into the operating system for such machines is necessary, as otherwise the link topology may be left in a disordered state after the user's low-level access. Finally, a feature of the Supernode architecture employed in our work is that the connections that can be made are restricted by the arrangement of the electronic switch. This causes difficulties in going from one topology to another, though not in setting up the initial topology.

4 Conclusion

As image-processing (IP) applications consist of several phases with different computer architectural needs, hardware reconfiguration is an important provision. We have shown that reconfiguration can also be employed within the low-level processing stage by means of dynamic switching. The performance can be predicted by various means and in particular by a polling-type queueing model. The algorithms we used were not selected because they were especially applicable to reconfiguration but because they were to form part of an IP software library. While the generic reconfigurable set-up developed proved useful in allowing data-parallel type algorithms to become more scalable, it was also useful for algorithms involving global communication, such as image rotation. The hardware employed is based around the P1085 switch design, which forms the backbone of the Supernode parallel processor. Though other machines continue to use electronically programmable switches (for instance the Paramid), unfortunately they have not chosen to make the switching easily accessible to the user. We do not make a case that dynamic link reconfigurability is essential but we do show that it can be used and would make a more attractive system. However, a future system should also include synchronization means in order that reconfiguration can be used effectively. The notion of reconfiguration is not confined to hardware: it is also important to make software

as flexible as possible in terms of the possible target architectures. Aspects of this work are ongoing. It is also hoped to use the performance modelling ideas started here in the context of parallel embedded systems, which have a cost sensitive component making some aids to transparent communication less attractive. Where there are pipelines of processor farms it may well be necessary to dynamically reconfigure the partitioning of the farms as the balance of the workload alters.

Acknowledgement

This work was carried out as part of project IED3/1/2171 ("Parallel Reconfigurable Image Processing Systems").

References

[1] M. J. B. Duff and S. Levialdi, editors. *Languages and Architectures for Image Processing.* Academic Press, London, 1981.

[2] K. Preston and L. Uhr, editors. *Multicomputers and Image Processing–Algorithms and Programs.* Academic Press, London, 1982.

[3] J. Kittler and M. J. B. Duff, editors. *Image Processing System Architectures.* Research Studies Press, Letchworth, UK, 1985.

[4] L. Uhr, K. Preston, S. Levialdi, and M. J. B. Duff, editors. *Evaluation of Multicomputers for Image Processing.* Academic Press, London, 1985.

[5] A. F. Clark, K. Martinez, and W. Welsh. Parallel architectures for image processing. In D. Pearson, editor, *Image Processing*, pages 169–189. McGraw-Hill, London, 1991.

[6] R. O. Duda and P. E. Hart. *Pattern Classification and Scene Analysis.* Wiley, New York, 1973.

[7] S. R. Unger. A computer oriented toward spatial problems. *Proceedings of IRE*, 46:1744–1750, 1958.

[8] A. Samal. Design of a dynamically reconfigurable integrated parallel vision system. In *IEEE Computer Vision and Pattern Recognition*, pages 521–523, 1990.

[9] S. R. Sternberg. Bio-medical image processing. *IEEE Computer*, pages 22–34, 1983.

[10] K. Martinez and D. Pearson. PETAL: a parallel processor for real-time primitive extraction. In *Proceedings of SPIE Symposium on Architectures and Algorithms for Digital Image Processing*, volume 596, 1985.

[11] N. K. Kasabov. Functionally reconfigurable general purpose parallel machines and some image processing and pattern recognition applications. *Pattern Recognition Letters*, 3:215–223, 1985.

[12] T. J. Fountain. A survey of bit-serial array processor circuits. In M. J. B. Duff, editor, *Computing Structures for Image Processing*, pages 1–14. Academic Press, London, 1983.

[13] D. Antonsson, B. Gudmundsson, T. Hedblum, B. Kruse, A. Linge, P. Lord, and T. Ohlsson. PICAP–a system approach to image processing. *IEEE Transactions on Computers*, 31(10):997–1000, 1982.

[14] M. Maresca and T. J. Fountain. Scanning the issue: Massively parallel computers. *Proceedings of the IEEE*, 79(4):395–401, April 1991.

[15] G.R. Nudd and N.D. Francis. Architecture for image processing. In $3^{\underline{rd}}$ *International Conference on Image Processing*, pages 445–451. IEEE, 1989.

[16] D. A. Reed and R. M. Fujimoto. *Multicomputer Networks: Message-Based Parallel Processing*. MIT, Cambridge, MA, 1987.

[17] E. R. Davies. Image processing—its milieu, its nature, and constraints on the design of special architectures for its implementation. In M. J. B. Duff, editor, *Computing Structures for Image Processing*, pages 57–76. Academic Press, London, 1983.

[18] A. F. Clark and K. Martinez. Image-processing architectures. In D. Pearson, editor, *Image Processing*, pages 141–155. McGraw-Hill, London, 1991.

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

[20] Transtech Parallel Systems Corporation, 20, Thornwood Drive, Ithaca, NY, USA. *The Paramid User Guide*, 1993.

[21] J. Beecroft, M. Homewood, and McLaren M. Meiko CS-2 interconnect Elan-Elite design, 1993. Technical Note.

[22] A. Gupta and A. V. Kumar. The scalability of FFT on parallel computers. *IEEE Transactions on Parallel and Distributed Systems*, 4(8):922–932, 1993.

[23] I. R. Greenshields. A dynamically reconfigurable architecture for image processing. In P. M. Dew, R.A. Earnshaw, and T. R. Heywood, editors, *Parallel Processing for Computer Vision and Display*. Addison-Wesley, Wokingham, UK, 1989.

[24] J. Y. Lee, C. S. Jeang, J. Y. Lee, and W. M. Jeang. An implementation of dynamic reconfiguration in POPA (POhang PARallel) computer. In T.S. Durrani, W.A. Sandham, J.J. Soraghan, and S.M. Forbes, editors, *Applications of Transputers 3*, pages 38–47. IOS, Amsterdam, 1991.

[25] J. M. Adamo, J. Bonneville, C. Bonello, P. Moukeli, N. Alhafez, and L. Trejo. Developing a high level programming environment for Supernode. In T.S. Durrani, W.A. Sandham, J.J. Soraghan, and S.M. Forbes, editors, *Applications of Transputers 3*, pages 632–637. IOS, Amsterdam, 1991.

[26] A. Murta. Support for network-wide synchronous communication via the active reconfiguration of transputer links. In R. Grebe, editor, *Transputer Applications and Systems*, pages 772–778. IOS, Amsterdam, 1993.

[27] S.I. Kartashev, S.P. Kartashev, and C.V. Ramamoorthy. Adaptation properties for dynamic architectures. In *NCC*, pages 543–555. AFIPS, 1979.

[28] P. H. Welch. Graceful termination — graceful resetting. In Bakkers A., editor, $10^{\underline{th}}$ *Occam User Group Technical Meeting*. IOS, Amsterdam, 1989.

[29] G.R. Andrews. Paradigms for process interaction in distributed programs. *ACM Computing Surveys*, 23(1):49–90, 1991.

[30] N.J. Edwards. *Dynamically Reconfigurable Systems*. PhD thesis, Bristol University, 1989.

[31] R. G. Tregidgo. *Parallel Processing and Automatic Postal Address Recognition*. PhD thesis, Essex University, May 1992.

[32] 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.

[33] 3L Ltd., Peel House, Ladywell, Livingston, Scotland. *Parallel C User Guide*, 1991.

[34] W. F. McColl. Scalable computing. *Lecture Notes in Computer Science*, 1000:46–61, 1995.

[35] G. Hall and T.J. Terrell. Implementation of the Radon transform using a dynamically switched transputer network. In D.J. Pritchard and C.J. Scott, editors, *Proceedings of the 2nd International Conference on Applications of Transputers, Southampton*, pages 156–163. IOS, Amsterdam, 1990.

[36] D. E. Pearson and J. A. Robinson. Visual communication at very low data rates. *Proceedings of the IEEE*, 73(4):795–812, 1985.

[37] Parsys Ltd., Boundary House, Boston Road, London. *Hardware Reference Manual for the Parsys SN1000 Series*, 1989.

[38] D. A. Nicole. ESPRIT project P1085: Reconfigurable transputer processor architecture. Technical report, Dept. of Computer Science, Southampton University, 1988.

[39] L. Hayat, M. Fleury, and A. F. Clark. A two-dimensional median filter algorithm for parallel reconfigurable computers. *IEE Proceedings Part I (Vision, Image and Signal Processing*, 143:345–350, 1996.

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

[41] A. Averbuch, E. Gabber, B. Gordissky, and Y. Medan. A parallel FFT on an MIMD machine. *Parallel Computing*, 15:61–74, 1990.

[42] P. N. Swartzrauber. Multiprocessor FFTs. *Parallel Computing*, 5:197–210, 1987.

[43] J. O. Eklundh. A fast computer method for matrix transpose. *IEEE Transactions on Computers*, pages 801–803, 1972.

[44] H. Kunieda and K. Itah. Parallel 2D-FFT algorithm on a practical multiprocessor system. In D. May and K. L. Kunii, editors, *Proceedings of the* $3^{\underline{rd}}$ *Transputer-Occam International Conference, Tokyo 1990*, pages 77–89. IOS, Amsterdam, 1990.

[45] G. E. Rivard. Direct fast Fourier transform of bivariate functions. *IEEE Transactions On Acoustics, Speech, and Signal Processing*, 25(3):250–252, 1977.

[46] R. Creutzburg, T. Minkwitz, M. Roggenbach, and T. Umland. Parallel FFT-like transform algorithms on transputers. In I. Pitas, editor, *Parallel Algorithms for Digital Image Processing, Computer Vision and Neural Networks*, pages 52–89. Wiley, Chichester, UK, 1993.

[47] L. Sousa, J. Burrios, A. Costa, and M. Picdate. Parallel image processing for transputer-based systems. In *IEE International Conference on Image Processing and its Applications*, pages 33–36, 1992.

[48] D. J. Pritchard. Mathematical models of distributed computation. In $7^{\underline{th}}$ *Occam User Group Technical Meeting*. IOS, Amsterdam, 1987.

[49] H. V. Sreekantaswamy, S. Chanson, and A. Wagner. Performance prediction modelling of multicomputers. In $12^{\underline{th}}$ *International Conference on Distributed Computing Systems*, pages 278–285. IEEE Computer Society, Los Alamitos, CA, 1992.

[50] M. Fleury, L. Hayat, and A. F. Clark. Performance estimation for a dynamically reconfigurable multiprocessor system applied to low-level image processing. In *Proceedings of the Conference on the Performance Evaluation of Parallel Systems (PEPS '93)*, pages 209–213. University of Warwick, 1993.

[51] E. Gelenbe. *Multiprocessor Performance*. Wiley, Chichester, UK, 1989.

[52] H. Takagi. Queueing analysis of polling models. *ACM Computing Surveys*, 20:5–28, 1988.

Reconfigurable System		Static Tree	
No. of processors	Time (sec)	No. of processors	Time (sec)
6	14.33	3	28.53
9	9.65	5	17.21
12	7.44	7	12.35
15	6.22	9	12.58
18	5.47	11	12.23
		13	11.91
		15	11.70

Table 1: Execution Timings for a Median Filter Using Various Set-ups, Image Size 1024×768

Image size	Factorization	No. of Processors					
		1	4	6	8	10	12
128	4,4,4,2	4.18	0.63	0.56	0.53	0.54	0.56
144	6,6,4	5.01	0.87	0.68	0.67	0.66	0.69
216	6,6,6	11.54	2.01	1.49	1.40	1.38	1.40
256	4,4,4,4	18.02	2.89	2.23	1.99	1.95	1.98
288	6,6,4,2	25.20	4.18	2.97	2.55	2.50	2.47
384	6,4,4,4	43.40	7.30	5.16	4.41	4.29	4.23
432	6,6,6,2	60.40	9.74	6.84	6.72	5.43	5.23
512	4,4,4,4,2	88.36	13.43	12.02	8.30	7.65	7.53
576	6,6,4,4	105.33	16.18	12.02	10.05	9.59	9.13

Table 2: Execution Timings for a Mixed-Radix Real-Valued 2D-FFT on the Dynamically-Switched System, Time in Sec.

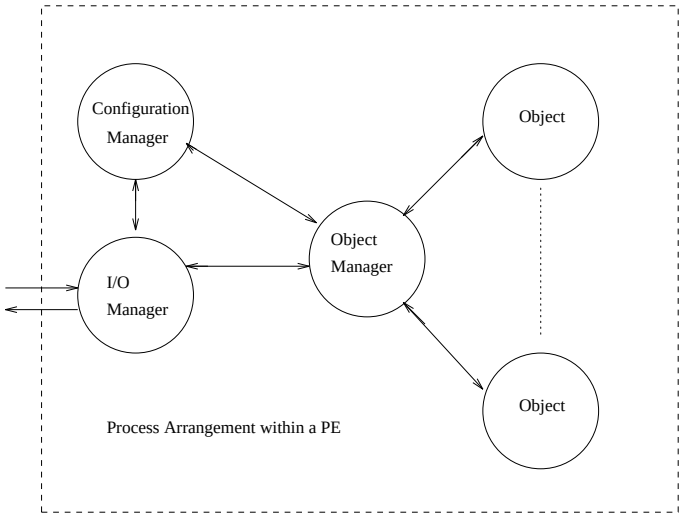

Figure 1: Object Management Incorporating Reconfiguration

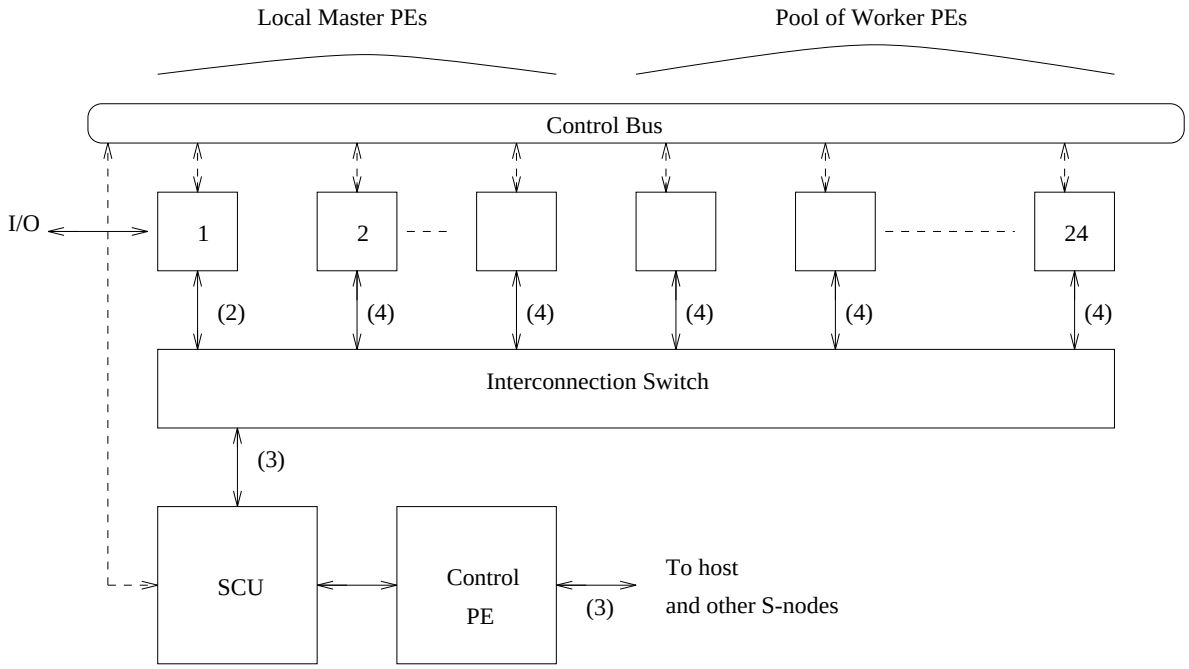

Figure 2: Parsys SuperNode Architecture

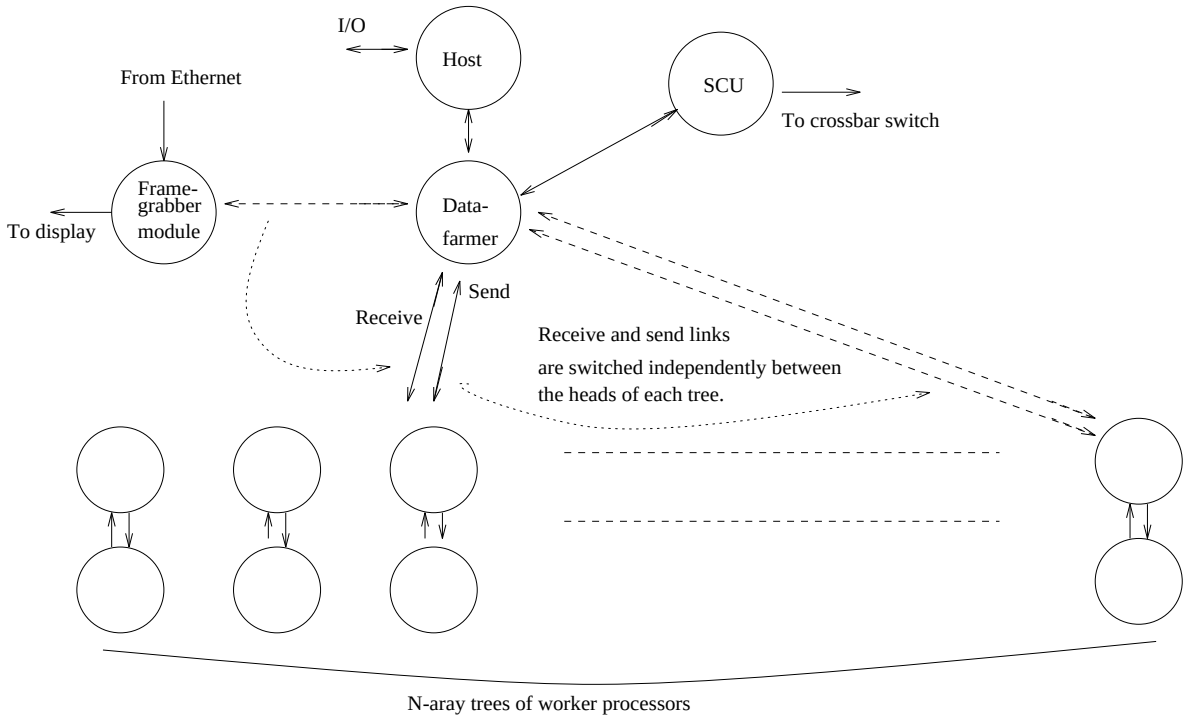

Figure 3: Set-up for Dynamically-Switched Network with Local Communication

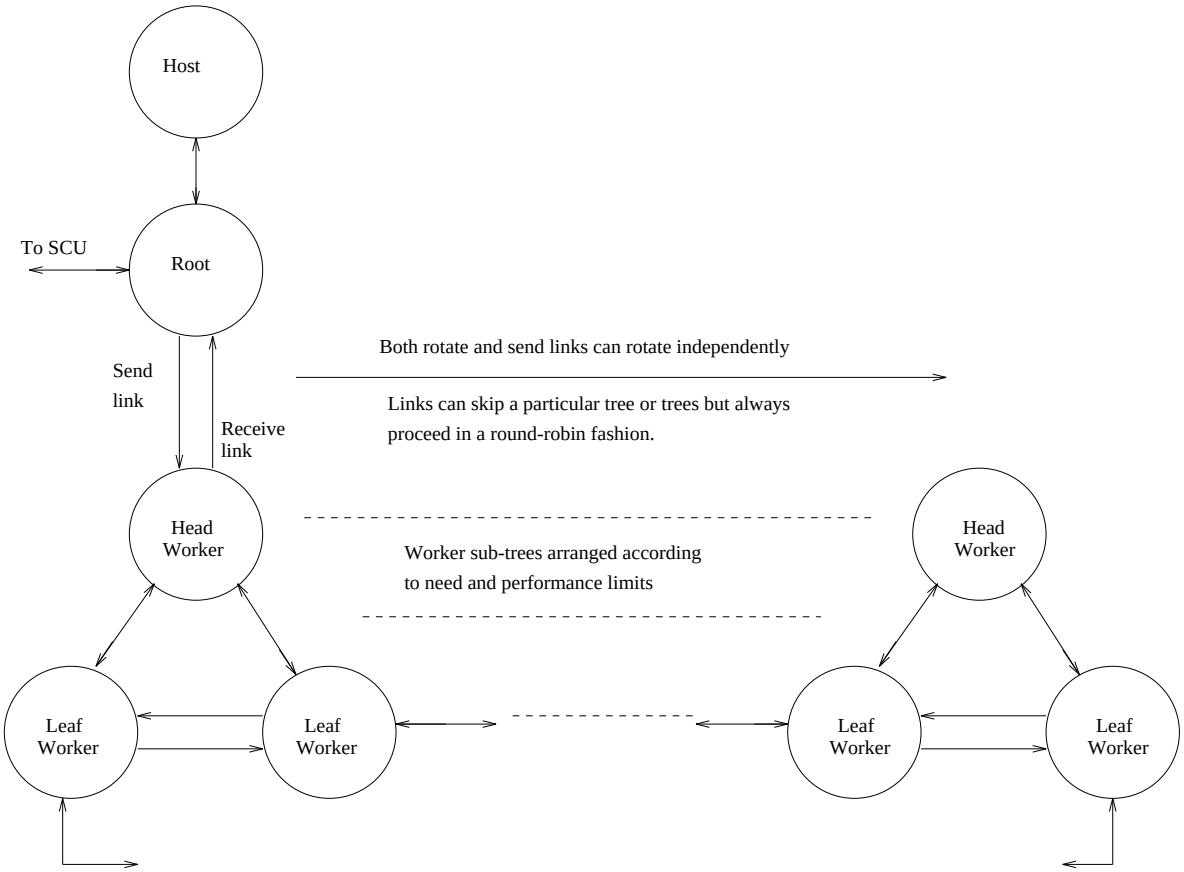

Figure 4: Set-up for Dynamically-Switched Network with Global Communication

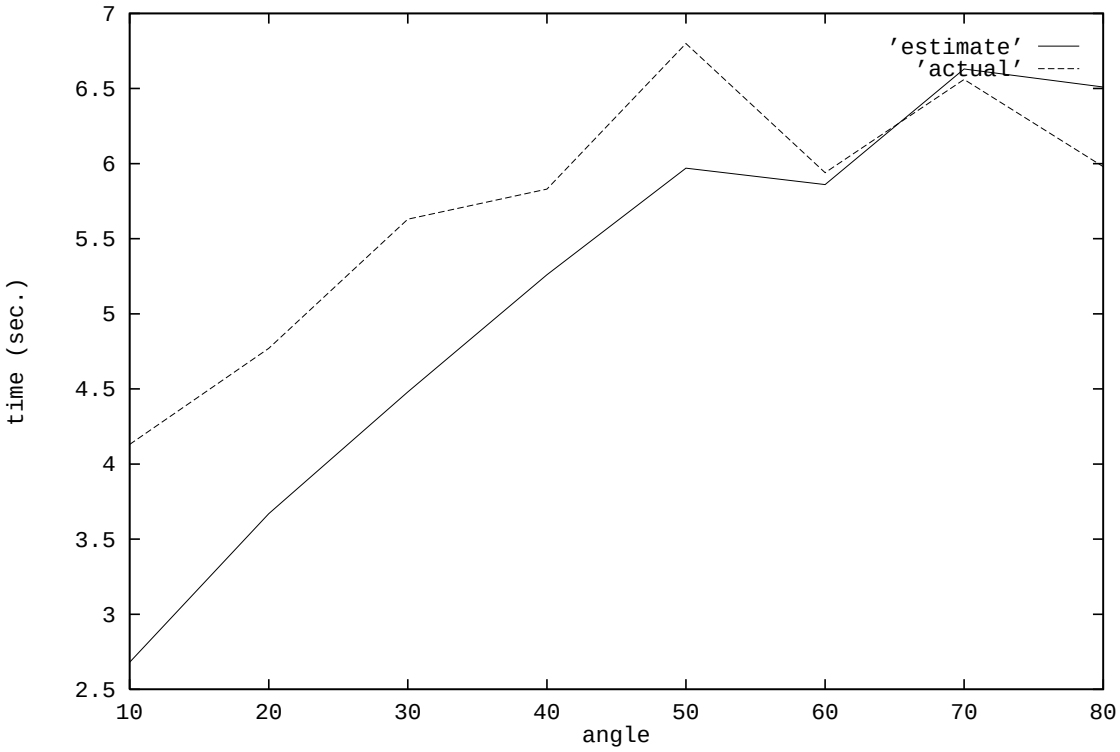

Figure 5: Performance Estimate for 12 Worker Processors in the Topology of Figure 3