A Multi-DSP Neighbourhood Architecture Using The TMS320C25

P. H. Gregson*, R. Christopher, W. Robertson
Department of Electrical Engineering,
Technical University of Nova Scotia,
Halifax, Nova Scotia, B3J-2X4
e-mail: gregson@tuns.ca

1 Introduction

Algorithms and architectures for neighbourhood-based processing are currently topics of great interest, fuelled in part by recent interest in neighbourhood-based problems such as fluid flow modelling, computer vision and image processing, relaxation-based classification and cluster analysis. These problems are computationally very expensive, requiring generally excessive amounts of computer time for their solution on computers with one or at most a few processors.

Recent advances in VLSI design and fabrication have opened the door to computational architectures specific to these problems. These architectures consist of large numbers of processing elements (PEs), each performing a limited number of simple instructions. The most common architectures have been SIMD (single instruction, multiple data) machines in systolic or wavefront arrays. The PEs within the array or along a 'wavefront' execute the same instructions at each instant in time. One variant of the approach permits each processor to either execute the current instruction, or to do nothing [9].

While these architectures permit applying large numbers of processing elements to a given problem, it is clear that a given algorithm is not necessarily best implemented by all PEs executing identical instructions. In image processing, systolic arrays may be used to implement those spatial filtering and template matching operations which are typically performed on the whole image. There is another class of algorithms, however, for which the size and shape of the neighbourhood as well as the operations performed over the neighbourhood are functions of interrim measurements. These algorithms are not readily or efficiently implementable using systolic arrays, if they may be implemented at all. Examples from computer vision include boundary completion [4], edge-preserving noise filtering [12], edge detection based on gradient angular dispersion [5,7], feature extraction [3], determination of edge detection thresholds based on neighbourhood signal to noise ratios [2,8]. Another example currently being investigated is the use of both the back-projection of the Hough transform [1] for modifying detection thresholds, and the signal to noise ratio of the Hough accumulator array for determining the appropriate neighbourhood size via an iterative optimization strategy [6].

Frequently, neighbourhood computations are not restricted to sequences of similar operations, even when changes in parameters are permitted. Often in computer vision, the results of one operation are used to select or tune subsequent steps. To implement such strategies on systolic arrays, each processing element must either contain the circuitry to permit all the possible operations, or must be programmable. In the former case, the complexity required is too great to permit integration of very large numbers of PEs at this time. In the latter case, the PEs may process words of sufficient width to implement operations in one step, or use words of few bits, implementing operations with sequences of very simple steps [9]. Again, long word-length requires complex PEs. The short word-length approach is initially attractive, but programming is difficult and high execution speed requires very large numbers of PEs.

The availability of very fast digital signal processing (DSP) chips suggests an alternative. An architecture consisting of a relatively small number of DSPs (16 to 64), with local busses connecting ad-

*Research supported by grants from the Canadian Natural Sciences and Engineering Research Council and by a donation from Texas Instruments Inc.

jacent DSPs, local memory for programs and data, dedicated inter-DSP communications hardware and serial inter-processor communications for 'broadcast' messages can address most of the problems discussed. Such an architecture is realizable with current technology. DSPs have sufficiently rich instruction repertoires for data-contingent neighbourhood operations, and generally have modified Harvard architectures to facilitate rapid program execution [10,11]. The issue of programming difficulty is partially addressed through the availability of cross-development tools [13], and may be partially addressed through adequate hardware design. Local interconnection to nearest neighbours and dedicated inter-DSP communications hardware prevents the traditional bus access bottleneck which limits the expandability of shared-bus architectures to about four processors.

2 Architecture Description

The Multi-DSP Neighbourhood Architecture (MDNA) is a 'compute engine' for neighbourhood algorithms, consisting of four major modules and some control logic (Figure 1). It is controlled and its I/O is handled by a host computer (a PC-AT clone) via a (relatively) high speed parallel bus. Programs can be developed and stored on the host and tested on either the host or the MDNA in a nearly seamless development environment.

2.1 Digital Signal Processor

The DSP performs all calculations, based on programs stored in the program memory. The architecture is designed so as to permit the maximum DSP throughput. While the ideas behind the architecture may be implemented with a variety of logic families and with one of a number of different types of DSP, this research has concentrated on an implementation using the Texas Instruments TMS320C25 DSP. This DSP has the virtues of having a large variety of cross development tools [13], compatibility with currently available low cost logic families, a sufficiently rich instruction set, an established upgrade path to DSPs with increased performance, availability and relatively low cost.

2.2 Program Memory

This memory consists of both EPROM and RAM. EPROM contains the Resident Services Routines and, in dedicated applications, the program to be executed. Fast RAM (25 nS) is used for RAM-based program storage to permit downloading programs from the host. Programs contained in program EPROM may also be transferred to this RAM for much faster execution by the DSP.

2.3 Data Memory

The data RAM consists of two banks, accessed via a multiplexer. Source data (on which computations are based) are stored in one bank, and results of computations are stored in the second bank. The multiplexer permits easily interchanging the roles of the two data RAM banks, a necessary ability since the execution of long multi-stage algorithms frequently use the results of preceeding stages as source data for succeeding stages. The multiplexer permits interchanging roles without transferring data between banks.

2.4 Write Contention Arbiter

The arbiter contains latches for the address and data of the results generated by the node's DSP and the DSPs of the eight neighbouring nodes. The data portion is the actual data element to be stored. The address portion is the address at which the data element is to be stored expressed in terms of the receiving node's array coordinates.

An internal priority encoder and finite state machine are used to write the results in an arbitrary sequence. While the priority encoder causes data to be written with a strict, fixed priority, the data elements being written will not be needed until the roles of source and result RAM banks are interchanged. Thus, this prioritization need not be considered during algorithm design.

As will be shown, the strength of the architecture rests with the arbiter, since it enables operation of all DSPs at maximum speed for significant problems without incuring inter-communications overhead if properly designed.

3 Architecture Operation

Each 'DSP node' in the architecture is used to compute the neighbourhood operations about each data element in its 'source region' (SR), a section of the input data space. Each node thus implements a large number of 'virtual' processing elements (VPEs) for algorithms requiring a separate PE at each element in the data space to compute a result over its

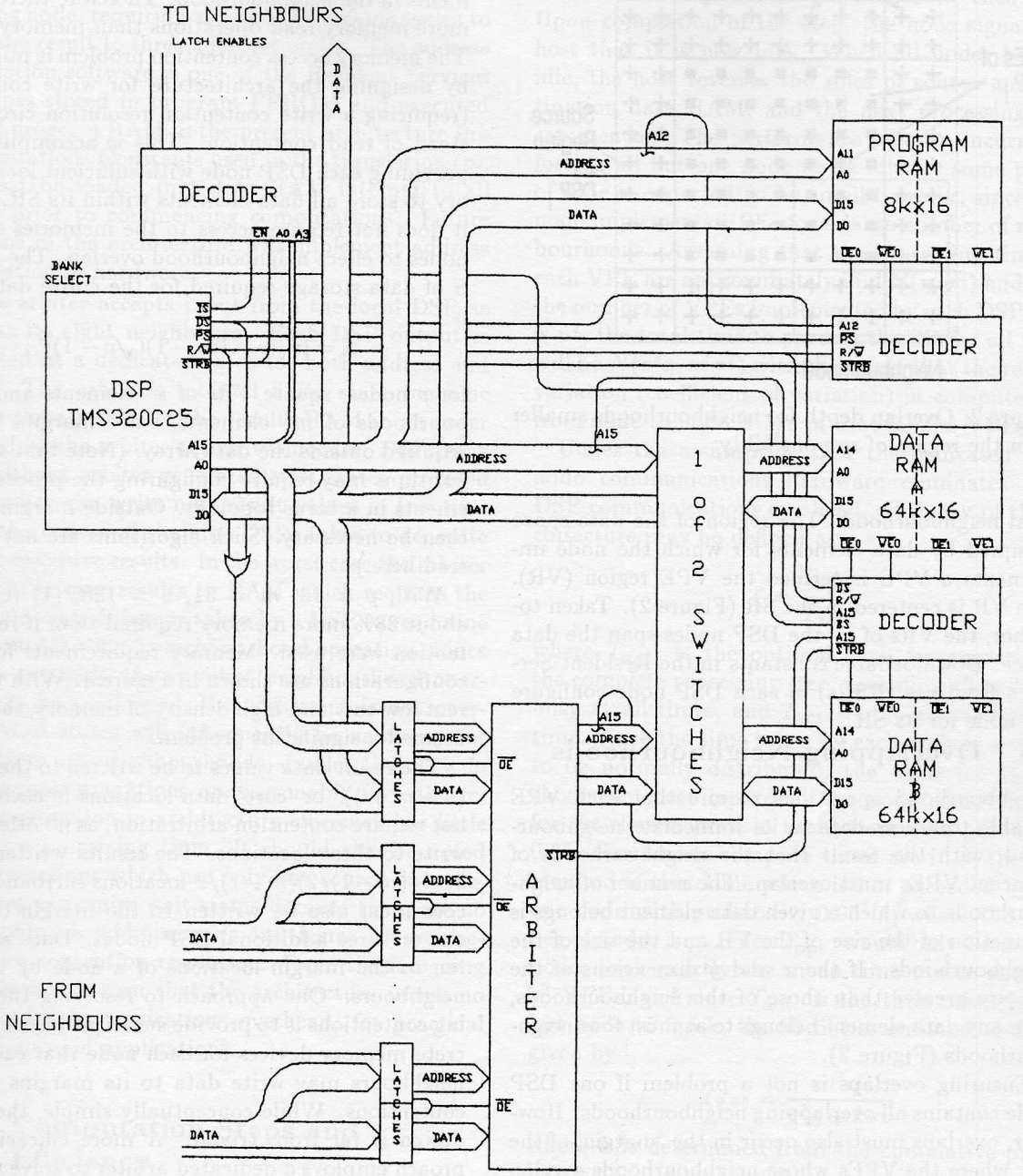

Figure 1: Block diagram of a single DSP node.

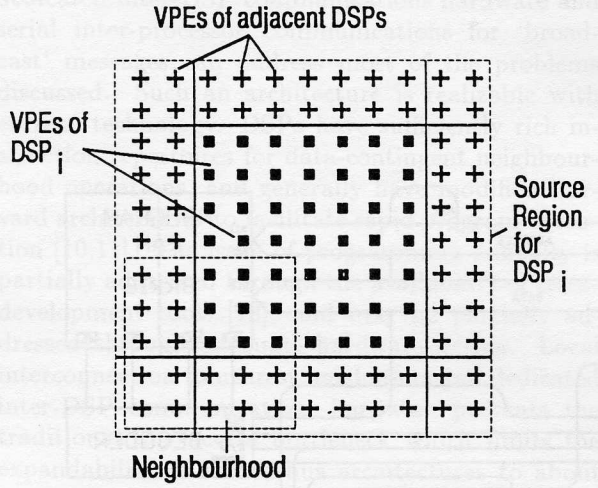

Figure 2: Overlap depth for neighbourhoods smaller than the region of responsibility.

local neighbourhood. The region of the data space occupied by data elements for which the node implements a VPE is termed the VPE region (VR). The VR is centered in the SR (Figure 2). Taken together, the VRs of all the DSP nodes span the data space. Downloadable constants in the Resident Services Routines (RSRs) in each DSP node configure the node for its SR.

3.1 Overlapping Neighbourhoods

Neighbourhood operations require that each VPE be able to access data in its immediate neighbourhood, with the result that the neighbourhoods of adjacent VPEs must overlap. The number of neighbourhoods to which a given data element belongs is a function of the size of the VR and the size of the neighbourhoods. If the x and y dimensions of the VR are greater than those of the neighbourhoods, then any data element belongs to at most four neighbourhoods (Figure 2).

Ensuring overlaps is not a problem if one DSP node contains all overlapping neighbourhoods. However, overlaps must also occur in the 'margin' of the SR, where the VPEs whose neighbourhoods overlap are implemented on two or more DSP nodes. For neighbourhoods of size m^2, the margin of the SR consists of the outer $(m-1)/2$ rows and columns of data elements which form a hollow square.

Inter-node neighbourhood overlap makes it necessary for more than one DSP to have access to some data elements, leading to a memory access contention problem. This problem is minimized by recognizing that most neighbourhood algorithms compute one (or at most a few) result data elements on the basis of a much larger number of source data elements in the neighbourhood. Therefore there are far more memory read operations than memory writes. The memory access contention problem is minimized by designing the architecture for write contention (requiring a write contention resolution circuit) instead of read contention. This is accomplished by providing each DSP node with sufficient local memory to store all data elements within its SR, so that it does not require access to the memories of other nodes to effect neighbourhood overlap. The amount S of data storage required for the entire data array is

$$S = [v\sqrt{p} + (m-1)(\sqrt{p}-1)]^2 \qquad (1)$$

for p nodes, square VRs of v^2 elements and neighbourhoods of m^2 elements, since margins are not required outside the data array. (Note that some algorithms may require configuring the processing elements in a torus topology. Outside margins would then be necessary. Such algorithms are not considered here.)

With $p = 16$, $m = 31$, $v = 128$, (1) results in about 38% more memory required than if read contention were used. Memory requirements for other configurations are shown in Figure 4. With the current low cost and high density of memory, this is not deemed a significant problem.

The result data values to be written to the central $(v-m+1)^2$ or 'core' data locations in each VR do not require contention arbitration, as no other DSPs write to these locations. The results written to the $4(v-(m-1)/2)(m-1)/2$ locations surrounding the core must also be written to the margin of either one or three additional DSP nodes. Data are written to the margin locations of a node by its eight neighbours. One approach to resolving the resulting contentions is to provide sufficient physically discrete memory devices for each node that each of its neighbours may write data to its margins without contentions. While conceptually simple, the circuit design is far from trivial. A more efficacious approach employs a dedicated arbiter to solve the contentions.

When a result is to be written to the margins of adjacent nodes, the address at which the result is to be stored must be calculated with respect to the coordinate frame of the receiving node. Since each node generates results for at most three other nodes (given m less than v), this computation will be performed at most three times. At present, this calcula-

tion is performed in software by the node generating the result, as it knows the storage location required in terms of its own coordinate system. Translation of the coordinates is a simple procedure for the transmitting node, requiring at most 13 machine cycles to send the result to three adjacent nodes. The address translation software is one of the Resident Services Routines stored in program EPROM and executed out of program RAM in the present architecture implementation. Constants used in the translation (m, v) must be loaded into data RAM internal to all DSPs prior to commencing computations. Future versions of the architecture will implement address translation in hardware.

The arbiter accepts input from the local DSP as well as its eight neighbours. Each DSP output is accepted at a dedicated latch for both address and data. The arbiter's internal finite state machine writes these results to the results RAM in an order defined by the arbiter's priority encoder. DSPs may run without arbiter-generated wait states provided the arbiter can write nine result values in less time than the fastest of the nine DSPs takes to generate two consecutive results. In the worst case the arbiter must write nine results to RAM which requires the same amount of time as taken by a DSP to read nine locations for a 3×3 neighbourhood operation. Since this is probably the smallest neighbourhood operation to be performed, the constraint is not onerous. Wait states will infrequently be necessary for operations on neighbourhoods smaller than 3×3, but as these operations are reasonably fast, the resulting reduction in architecture efficiency has little effect on processing time. It should also be noted that operations which use only one source data element never require wait states because there are no overlapping neighbourhoods and hence no requirement for contention resolution. From the foregoing discussion it is seen that the architecture incurs no inter-node communications overhead, for practical neighbourhood applications.

3.2 Computation Steps and Efficiency

Each DSP node performs all computations on data in its SR. A 'computation step' is defined as the set of all computations for all VPEs implemented by one node for which all source data are available in one bank of data RAM. Further, all computations performed at each step must use neighbourhood sizes $m \times m$, $m \leq m_{\max}$, where $m_{\max}$ is determined by the configuration constants loaded into the RSR of each node prior to loading the data into each node.

Algorithms must be structured to maximize the number of computations performed at each step. Upon completion of the step, the node signals the host that it is now idle. When all nodes become idle, the host reverses the roles of source and destination data RAM, and the next processing step commences. The performance penalty incurred in forcing all but one node to be idle for some period of time is expected to be small however, since each node implements VPEs for a large number of neighbourhoods. Assuming that the processing times at each VPE are approximately i.i.d $N(\eta, \sigma^2)$ and since the number of VPEs implemented by each DSP node is v^2, the total time to process the SR for all nodes will be $N(v^2\eta, v^2\sigma^2)$ with the result that the relative variation (coefficient of variation) in compute time from node to node is $1/v$ of that for the VPEs.

Under the assumption that the dedicated inter-node communications hardware eliminates inter-DSP communications overhead, efficiency of the architecture may be defined as

$$\zeta = \frac{T_{\text{opt}}}{T_{\text{act}}} \tag{2}$$

where T_{opt} is the optimal time for execution of the complete processing step, assuming all nodes are busy at all times, and T_{act} is the actual compute time. Since the time taken by each node is assumed to be normally distributed, the value for T_{act} is chosen to meet or exceed the node compute time for the slowest node in the system with probability 0.95. Since node compute times are i.i.d, the time required is that time which is not exceeded by any one node with probability $0.95^{1/p}$, where the number of nodes is p. Since T_{opt} would only occur if all nodes took exactly the mean time $v^2\eta$ to simulate v^2 VPEs, and since T_{act} is the time taken by the last node to finish, the 95-th percentile efficiency is given by

$$\zeta_{0.95} = \frac{v^2\eta}{v^2\eta + k\sigma v} \tag{3}$$

where k is determined from the cumulative probability

$$P\left(\frac{T_{\text{act (worst)}} - T_{\text{opt}}}{\sigma} \leq k\right) = 0.95^{1/p} \tag{4}$$

using tables. Figure 3 shows the worst-case 95-th percentile efficiency as a function of numbers of nodes for an image 256 pixels square. This worst-case efficiency occurs if each node must either do

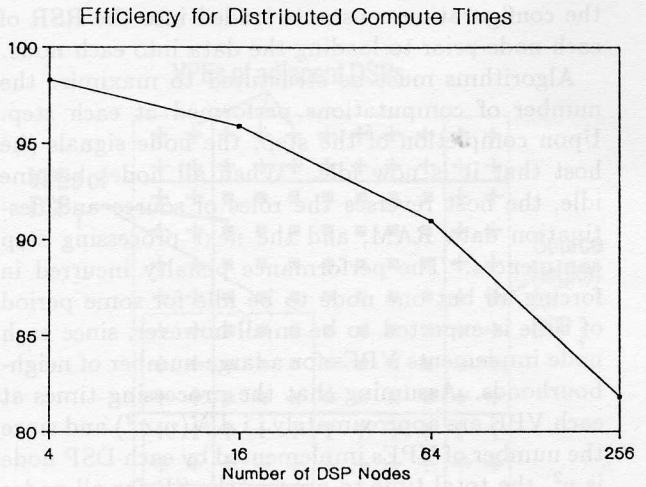

Figure 3: 95-th percentile efficiency as a function of number of nodes for an image of 256 pixels square.

nothing or compute for time 2η, with equal probability. In this case, $\sigma/\eta = 1$. Thus the figure depicts a lower bound on efficiency.

3.3 Downloading

Programs and data may be downloaded to all nodes in the MDNA at the same time since one program is run by all nodes at the same time. (This is still an MIMD structure since the processors may be executing different instructions at any one time.) Downloading is performed using a high speed (2 Mbit/s) serial communications link. Programs are reasonably short since they consist of only the 'inner loops' as discussed below, so download time is short.

Data may also be downloaded via the communications link but the time taken is significant for large data arrays, taking over one second for a 512 × 512 byte array. For complex algorithms of many compute steps, this may still be much less than the compute time, however.

A faster alternative is to download data through one 'edge' of the data array, thereby permitting the movement of data at v data elements in time t_A, where t_A is the access time for data RAM. For the current implementation and for $v = 4$, 2.0×10^7 data elements could be loaded per second. A 512 × 512 element image would then load in about 13 milliseconds.

Software for all downloading resides in the Resident Services Routines.

3.4 Expansion of the Architecture

Architecture throughput may be readily improved merely by installing more DSP nodes and changing some RSR constants, as the use of local inter-DSP communications busses and hardware arbitration result in an architecture whose throughput is linear in the number of DSP nodes. Application-specific cost/performance trade-offs are then made easily.

Problems of higher dimensionality may be accomodated using conventional techniques. For three dimensional problems, the inner loops are programmed to treat each data element as one element of a data vector. Programming is simplified if the elements of each vector are arranged to cover a square patch of the data space rather than a strip, since neighbourhoods are made to overlap more easily with this data organization. Routines to access data elements within data vectors can easily be made part of the Resident Services Routines.

4 Software Development

Software development for parallel architectures is generally difficult as algorithms must be explicitly structured by the programmer to take advantage of a particular architecture's parallelism. Algorithms are therefore highly dependent on the number of processors, and the internal architectural details of message passing and contention resolution. Porting algorithms is then difficult.

Neighbourhood algorithms frequently may be structured so as to hide these details however. For simple operations such as convolution, it is clear that the operation performed at every VPE is identical. Such algorithms may be implemented as two nested loops in which the loop counters provide the x and y coordinates of the result data element of interest. These loops can be implemented in the Resident Services Routines, to use values of m and v previously downloaded from the host.

The processing necessary to generate a result at a given location x_i, y_i is termed the 'elemental function' and is programmed with all data element references relative to that location. This usually entails two nested loops for non-separable neighbourhood operations. The elemental function is independent of the coordinates of the point of interest, and so may be written by the application programmer without knowledge of the number of DSPs currently installed in the architecture. The programmer must ensure that his problem will fit the architecture, by ensur-

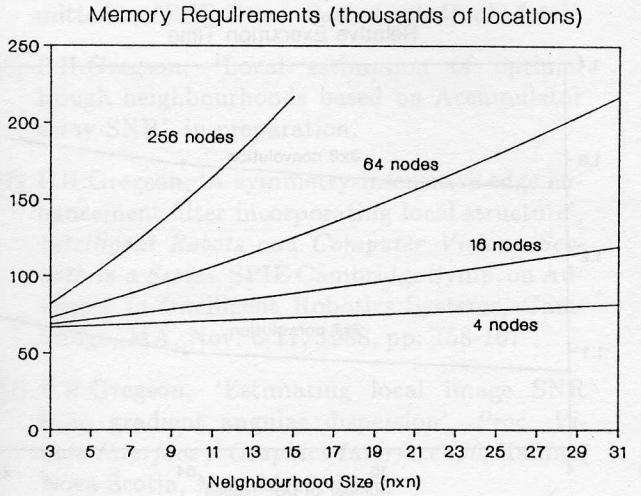

Figure 4: Memory requirements as a function of neighbourhood size.

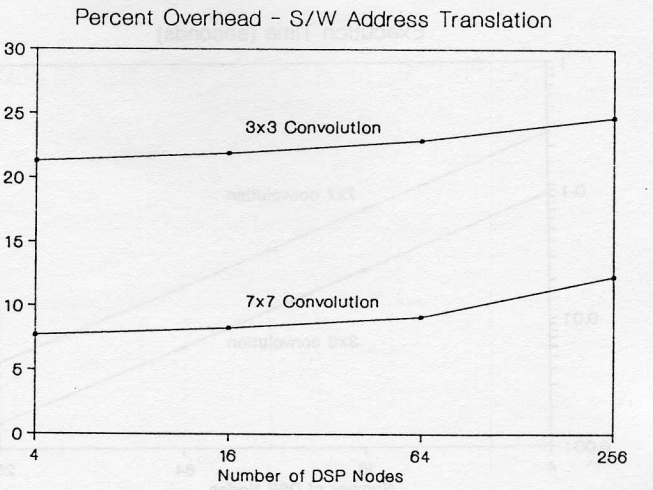

Figure 5: Performance of software address translation with respect to hardware-based translation for a data array of 256 × 256 elements.

ing that the installed memory exceeds the memory indicated in Figure 4 for the m required by the algorithm. While larger problems may be accomodated, the programmer must take care of problem partitioning.

Various languages are suitable for software development, requiring only that they have some construct to permit execution at a location specified by a vector. Candidates include Forth and C, since integrated cross-development environments (editor, compiler, linker, debugger, libraries etc.) already exist for TMS320C25 application development in these languages. Development environments must be augmented with a very simple facility to download the binary program to the nodes. In Forth, the development environment is easily tailored to permit running TMS320C25 code on the host machine to facilitate debugging. With adequate Resident Services Routines, the programmer sees a virtually seamless development and debugging environment.

5 Architecture Simulations

The architecture was simulated to test aspects of the foregoing. The time taken to execute 3 × 3 and 7 × 7 convolutions were determined by writing appropriate elemental functions and finding instruction execution times in the TMS320C25 specifications. Convolution kernels larger than 7 × 7 were not studied because the overhead incurred in address calculation becomes insignificant since it is fixed while the time to access source data is linear in the number of kernel elements. All simulations are based on programs being executed from program RAM without wait states, data fetched from data RAM with one wait state, and all DSPs operating wih a 40MHz clock oscillator. The convolution routines could be optimized to take better advantage of the TMS320C25 internal data RAM, but the simulation results would then be highly specific to convolution and would not indicate architecture performance when executing neighbourhood operations in general.

The first simulation explores the impact of using software address translation as opposed to hardware translation as a function of the number of DSP nodes and the size of the kernel. Since all address translation variables are passed to the RSR and thus are known to the DSP prior to executing this computation step, for this simulation only the program and data were assumed to be internal. Figure 5 shows the relative performance of software versus hardware translation as the inverse ratio of the processing times, indicating that efficiency increases with kernel size. Further, overhead increases as more DSP nodes are added because the ratio of margin pixels to core pixels has increased.

The second simulation (Figure 6) indicates that the execution time of the architecture is inversely related to the number of DSP nodes installed, or equivalently, throughput is linearly related to the number of nodes. This illustrates the effectiveness of the contention arbitration approach used.

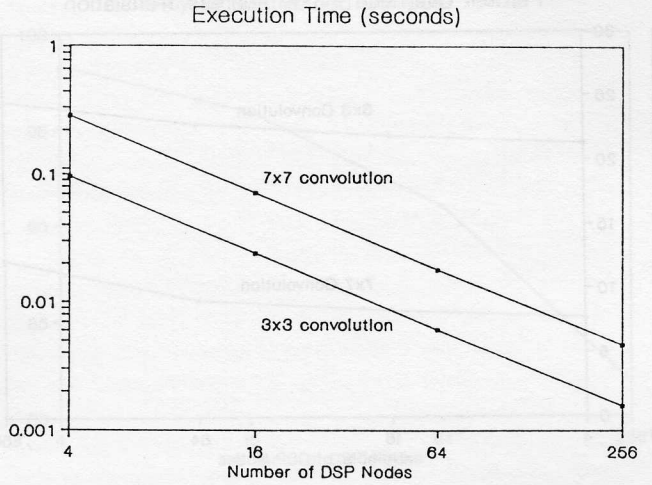

Figure 6: Execution time as a function of the number of DSP nodes.

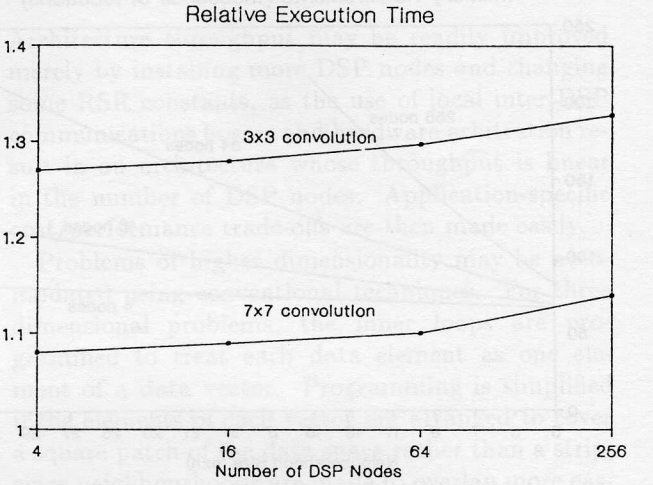

Figure 7: Architecture execution time relative to the theoretical minimum time.

A single DSP node would take 0.301 seconds to convolve a data array of 256 × 256 elements with a 3 × 3 kernel and 1.05 seconds for convolution with a 7 × 7 kernel. Figure 7 indicates compute times of the architecture relative to the theoretical optimum $T_{1\text{ node}}/p$, where $T_{1\text{ node}}$ is the time taken for one DSP to perform the entire convolution and p is the number of DSP nodes used. Compute times approach the theoretical optimum with larger kernels. This occurs because fewer instruction fetch operations per data element are necessary for larger kernels as the TMS320C25 has a facility for repeating instructions without re-fetching them.

6 Summary and Conclusions

Analysis of the neighbourhood processing problem has lead to a new, easily programmed, scalable multiprocessor architecture with performance linear in the number of processors. The architecture may be implemented with readily available, low cost components, and permits use of available software cross development products. By using dedicated interconnection between adjacent nodes and by implementing large numbers of VPEs with each DSP node, the architecture reflects the inherent structure of neighbourhood problems to the degree that enables the DSP chips to operate at maximum speed without inter-node communication wait states.

While the architecture has not yet been constructed, the detailed design has been completed and verified. It is anticipated that the first MDNA machine will be completed and operational in the very near future.

Future research in high speed neighbourhood architectures will attempt to perform MDNA address translation and write contention arbitration in VLSI, explore optical interconnection between adjacent nodes, and define an appropriate instruction set and internal architecture for a neighbourhood processing DSP chip that incorporates facilities for multi-directional interconnection. A high resolution video interface for direct display of the result data RAM will also be designed.

References

[1] D.H.Ballard, 'Generalizing the Hough transform to detect arbitrary shapes', *Pattern Recognition*, vol. 13, no. 2, 1981, pp. 111-122

[2] J.Canny, 'A computational approach to edge detection', *IEEE Trans. Patt. Anal. Mach. Intell.*, PAMI-8 no. 6, 1986, pp. 679-697

[3] D.S.Chen, 'A data-driven intermediate level feature extraction algorithm', *IEEE Trans. Patt. Anal. Mach. Intell.*, vol. 11, no. 7, 1989, pp. 749-758

[4] C.W.K.Critton, E.A.Parish Jr., 'Boundary location from an initial plan: the bead-chain algorithm', *IEEE Trans. Patt. Anal. Mach. Intell.*, PAMI-5 no. 1, 1983, pp. 8-13

[5] P.H.Gregson, 'Using angular dispersion of gradient direction for detecting edge ribbons', sub-

mitted to *IEEE Trans. Patt. Anal. Mach. Intell.*

[6] P.H.Gregson, 'Local estimation of optimal Hough neighbourhoods based on Accumulator array SNR', in preparation.

[7] P.H.Gregson, 'A symmetry-insensitive edge enhancement filter incorporating local structure', *Intelligent Robots and Computer Vision: Seventh in a Series*, SPIE Cambridge Symp. on Advances in Intelligent Robotics Systems, Cambridge, MA, Nov. 6-11, 1988, pp. 158-167

[8] P.H.Gregson, 'Estimating local image SNR from gradient angular dispersion', *Proc. Vision Interface / Graphics Interface '90*, Halifax, Nova Scotia, May 14-18, 1990

[9] B.A.Kahle, W.D.Hillis, 'The Connection Machine model CM-1 architecture', *IEEE Trans. Syst. Man, Cyb.*, vol. 19, no. 4, 1989, pp. 707-713

[10] E.A.Lee, 'Programmable DSP architectures: part 1', *IEEE ASSP Magazine*, vol. 5, no. 4, 1988, pp. 4-19

[11] E.A.Lee, 'Programmable DSP architectures: part 2', *IEEE ASSP Magazine*, vol. 6, no. 1, 1989, pp. 4-14

[12] J.-S.Lee, 'Digital image smoothing and the sigma filter', *Comp. Vis., Graphics, Im. Proc.*, vol. 24, 1983, pp. 255-269

[13] Texas Instruments Inc., *TMS320 Family Development Support Reference Guide*, document no. SPRU011, 1986