Computers Elect. Engng Vol. 20, No. 1, pp. 51–60, 1994
Printed in Great Britain. All rights reserved

0045-7906/94 $6.00 + 0.00
Copyright © 1994 Pergamon Press Ltd

PARALLEL ALGORITHMS FOR LOW-LEVEL VISION ON THE HOMOGENEOUS MULTIPROCESSOR

R. V. Dantu,† N. J. Dimopoulos,‡ K. F. Li,‡ R. V. Patel and A. J. Al-Khalili

Department of Electrical and Computer Engineering, Concordia University, 1455 de Maisonneuve Blvd, West Montreal, Quebec, Canada H3G 1M8

(Accepted in final revised form 17 February 1993)

Abstract—In this work, we present the parallel implementation of three low-level vision algorithms, namely smoothing, histogram generation and edge detection by using the Sobel operator on the Homogeneous Multiprocessor. These algorithms were run on the simulator specifically developed for the Homogeneous Multiprocessor and the simulation experiments were used to establish the performance of these algorithms on the proposed architecture.

Key words: Multiprocessing, vision, computer architecture, image processing, pattern recognition, MIMO architecture.

1. INTRODUCTION

Many applications in image processing require a large amount of computation, especially if they involve anything more than the most elementary processing techniques. Multiprocessor implementation of image processing and pattern recognition algorithms requiring a large amount of computation offers the possibility of making such algorithms more useful for practical, real-time applications. The complexity of algorithms is in general a function of the size of the problem, and the number of transfers between processors. Thus a hardware structure on which such computations can be performed efficiently requires the availability of communication pathways linking processors in a pattern that matches the one imposed by the algorithm chosen. One such architecture is the Homogeneous Multiprocessor [1,3] which is a closely coupled MIMD architecture providing nearest-neighbor communication.

This work describes the potential applications of the Homogeneous Multiprocessor in low-level image processing. We shall concentrate our discussion in presenting a brief overview of the Homogeneous Multiprocessor, the parallelization of some known low-level vision algorithms and the performance of these algorithms on the Homogeneous Multiprocessor. The performance was obtained through simulation experiments on the existing simulator for the Homogeneous Multiprocessor [2].

2. THE HOMOGENEOUS MULTIPROCESSOR AND THE H-NETWORK

As shown in Fig. 1, the Homogeneous Multiprocessor [3] is a tightly coupled MIMD architecture, composed of N ($N \geqslant 3$) processing elements, N memory modules, $N + 1$ interbus switches isolating the processing elements from each other and the H-network which is a fast local area network used for point-to-point and broadcast mode communications. The architecture is considered to be composed of two parts: namely the Homogeneous Multiprocessor Proper incorporating the processors, memories and interbus switches, and the H-network [1].

Each processing element P_i owns its local memory module M_i and accesses it via its local bus b_i: it also has the exclusive use of the respective network station HS_i. The local buses are separated by the intervening switches s_i. These switches provide each processor P_i with the ability to access the memory modules of either one of its two immediate neighbors by requesting the appropriate switch to close, creating thus an "extended bus". Also, for I/O or data transfers to/from distant

†Bell Northern Research Ltd, Canada.
‡Department of Electrical and Computer Engineering, University of Victoria, Victoria, British Columbia, Canada.

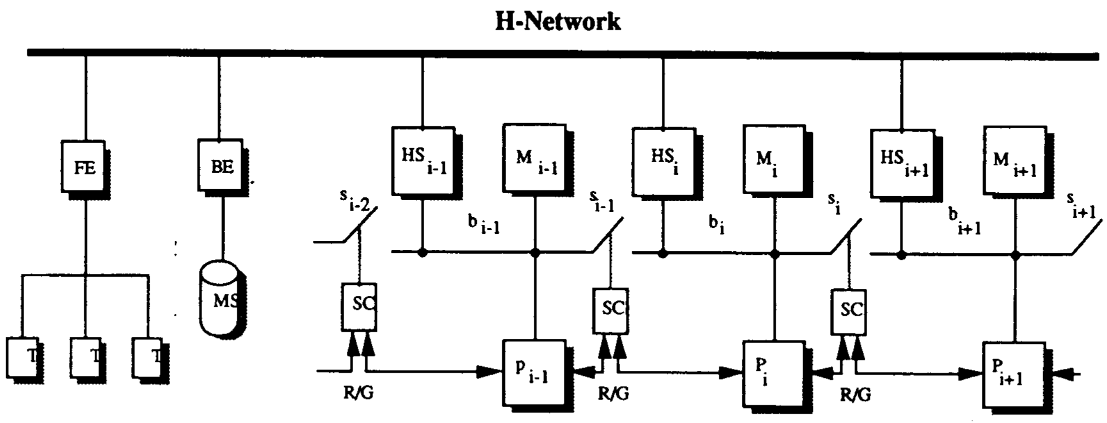


Fig. 1. The homogeneous multiprocessor architecture: P = processor; M = memory module; s = bus switch; FE = front end; BE = back end; SC = Switch controller; b = local bus; T = terminal; MS = mass storage; HS = H-network station; R/G = bus request/grant.

processors, each processor may utilize the H-network. Although the H-network could have been used for data transfers to/from distant processors (especially in the histogram calculation), the results reported in this work were obtained through the utilization of the "extended bus" communication mechanism. It was felt that this mechanism was considerably faster for short to medium distances since no task switching is involved. Nevertheless, we expect further improvement in the performance by employing the H-network for long distance transfers when it becomes operational.

The Homogeneous Multiprocessor is currently under implementation by using the 8 MHz MC68000 processor. The performance results reported in this study pertain therefore to an implementation of the multiprocessor by using the aforementioned processor.

3. IMAGE PROCESSING AND PARALLELISM

The Homogeneous Multiprocessor being a closely coupled MIMD architecture is perfectly suited for context dependent algorithms. Yet, some frequently used low-level image processing algorithms such as smoothing, histogram computation, edge detection by Sobel operator can be efficiently implemented on the Homogeneous Multiprocessor and the computations achieve significant speedups. In the following sections we shall present the method we used in implementing these algorithms on the Homogeneous Multiprocessor as well as their performance.

3.1. Image averaging

Averaging (smoothing) operations [6] are primarily used for diminishing noise. The raw image usually has sharp edges, but it is also noisy and it may contain small spiky artifacts. One of the most commonly used algorithms for smoothing is local averaging. Given an $M \times M$ image $f(x, y)$, the smoothed image $g(x, y)$ is obtained by averaging the grey level values of the pixels of the original image contained in a predetermined neighborhood $S(x, y)$ of (x, y):

$$g(x,y) = \frac{1}{n} \sum_{i,j \in S(x,y)} f(i,j) \tag{1}$$

where n is the total number of points in the set $S(x, y)$.

This operation is performed for each pixel in the image with the possible exception of the edge pixels.

We implement the smoothing algorithm on the Homogeneous Multiprocessor as follows. The $M \times M$ image is divided into N strips with each strip having $(M \times M)/N$ pixel columns. Each processor is allotted a strip of the $M \times M$ image and smooths the pixels in the strip located in its local memory using equation (1) above. Each processor needs to communicate with a neighboring processor only for the calculation of the smoothed image at the boundaries of the strip. This communication is accomplished directly through the "extended bus" mechanism, as described in

Section 2 above, and it is transparent to the programmer. Figure 2 shows the plot of the speedup factor obtained, against processors involved in the computation for two image sizes. The number of grey levels is fixed to 64 for both cases.

3.2. Histogram generation

A histogram of grey level content provides a global description of the appearance of an image. The histogram is frequently used in thresholding an image and interactive histogram modification is useful for enhancing picture quality.

It is assumed that there are N ($N = 2^k$) nodes in the Homogeneous Multiprocessor. The image is divided into N strips, and each strip is loaded in the memory of each of the nodes, which calculate the partial histogram of the region assigned to them in parallel.

The next step accomplishes the merging of the partial histograms. Assuming that there are B grey levels in the image, the partial as well as final histograms are stored in 1-D vectors each containing B elements or bins. Pairs of partial histograms are brought in neighboring memory modules and the corresponding elements are added to form a new partial histogram which supplants the pair of partial histograms. This is repeated until only one histogram remains. Normalization is done on the last remaining histogram. This merging is accomplished through a form of recursive doubling [5,8,9]. Initially, processors P_{2l+1}; $l = 0, 1, 2, \ldots, (N/2 - 1)$ merge the $B/2$ least significant elements of the partial histograms contained in their own as well as the memory of their neighbors to the right. Similarly, processors P_{2l+2} merge the $B/2$ most significant elements located in their own as well as the memory of their neighbors to the left at the end of these operations, nodes P_{2l+1} hold the least significant halves of the merged partial histograms, while their neighbors to the right (nodes P_{2l+2}) hold the most significant halves.

Next, nodes P_{4l+1}; $l = 0, 1, 2, \ldots, (N/4 - 1)$ transfer the $B/2$ least significant elements of their merged histograms to nodes P_{4l+2}, and similarly nodes P_{4l+4} transfer the $B/2$ most significant elements to nodes P_{4l+3}. At this point, nodes P_{4l+2} and P_{4l+3} contain partial B-element histograms, and the process is repeated. The final completed histogram is to be found in node $P_{N/2}$. Under this algorithm, the partial histograms are merged on a tree structure of processors embedded on the Homogeneous Multiprocessor as depicted in Fig. 3.

Observe that as the process progresses, partial histograms are located in nodes that are progressively further away from each other. Their merging requires the transfer of data between distant processors. Pipelining is then used to efficiently transfer long vectors between distant

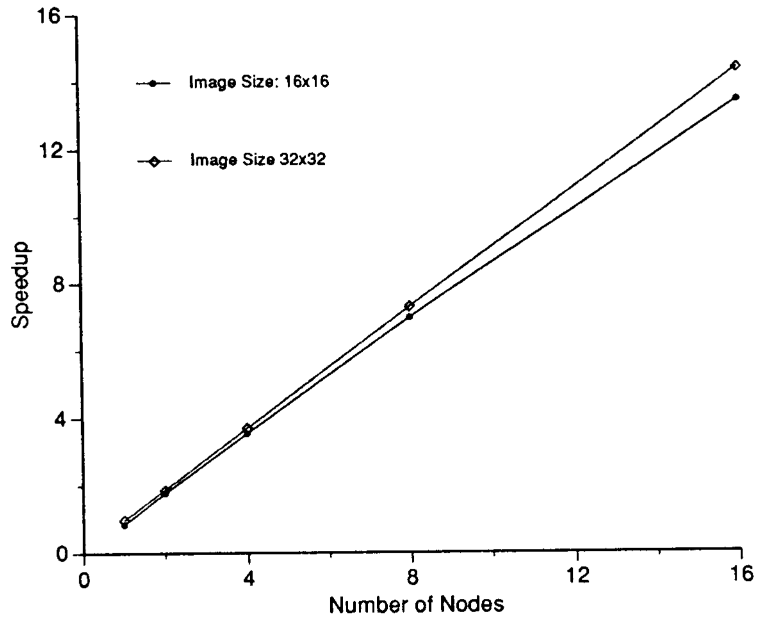


Fig. 2. Speedup vs the number of processors for the distributed averaging algorithm as obtained through simulation.

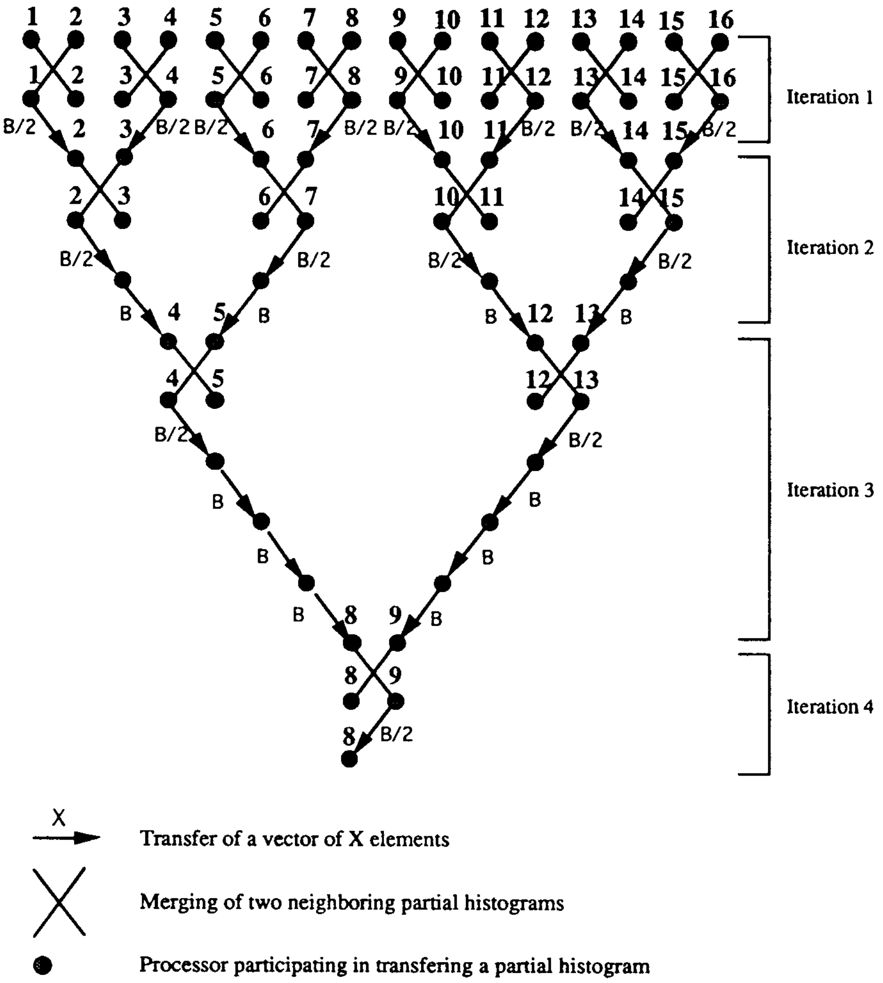

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
B/2
B
2 3 6 7 10 11 14 15
4 5 12 13
8 9
8
Iteration 1
Iteration 2
Iteration 3
Iteration 4
X
Transfer of a vector of X elements
Merging of two neighboring partial histograms
Processor participating in transfering a partial histogram

Fig. 3. An example of the distributed merge algorithm on 16 processors.

processors. Thus, in order to transfer a B-element vector from processor P_i to processor P_j, the intervening processors form a pipeline through which the B-element vector is transferred in $O(j-i+B)$ time units. Since neighboring nodes communicate by accessing each other's memory modules, it is possible to form a pipeline consisting of alternate nodes. Each node in the pipeline moves data from the memory nodule of its left neighbor to the memory of its right neighbor. The following example can be used to clarify the strategy. Consider the transfer of a vector of B elements from node 5 to node 8 (cf. Fig. 3). The transfer is accomplished by forming a two-stage pipeline consisting of nodes 6 and 8. Thus node 6 transfers data from the memory module of node 5 to the memory module of node 7 while node 8 completes the pipeline by transferring the data from the memory module of node 7 to its own. Assuming that transferring an individual element between two memory modules which can be accessed by the same node requires t_{tr} time units, then the transfer of the B-element vector as described previously, requires $(B+2-1)t_{\mathrm{tr}}$ time units [4]. This result can be generalized as $(B+\lceil (j-i)/2 \rceil - 1)t_{\mathrm{tr}}$ for transfers between nodes i and j.

The histogram merging algorithm outlined above is carried out in $\log N$ iterations. Each iteration consists of a partial merging step requiring $B/2$ additions and $B/2$ transfers, plus $B/2$ transfers needed to locate the merged histogram in the appropriate node. Observe that after iterations 2 to $\log N - 1$ the resulting partial histograms are located in nodes which are separated by $2^{(m-2)}$ intervening nodes, where m is the iteration number. Thus, the resulting pairs of B-element partial histograms are transferred through the $2^{(m-2)}$ intervening nodes to pairs of adjacent nodes and become ready for the subsequent merging iteration. These transfers use the pipelining method as discussed earlier, and are accomplished in $(B+2^{(m-2)}-1)t_{\mathrm{tr}}$ time units. Thus, if we denote by t_{a}

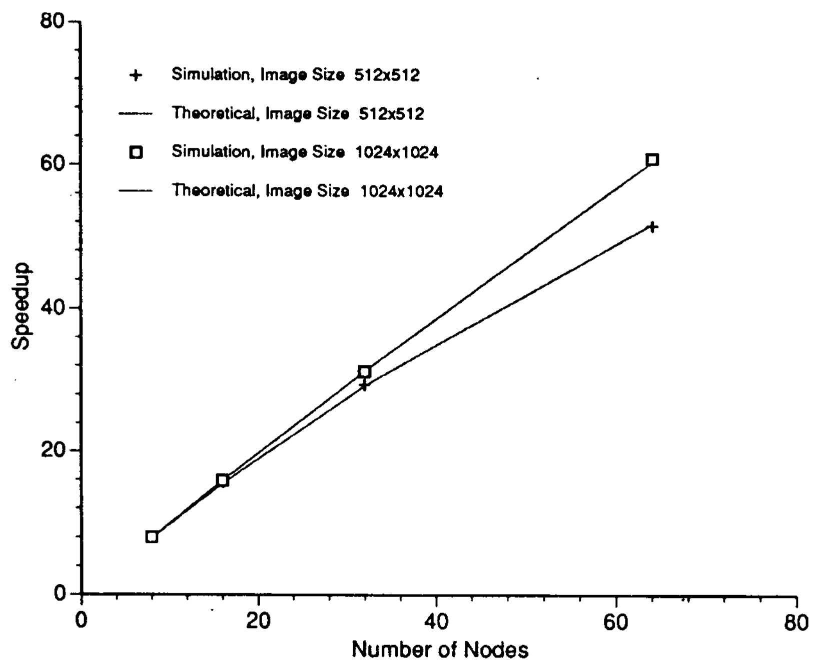


Fig. 4. Speedup vs the number of processors for the distributed histogram algorithm as obtained through simulation and analysis.

the time required to perform a single addition, then the time required for the histogram merging algorithm can be calculated as†

$$
\begin{aligned}
T_{\mathrm{MRG}} &= \sum_{m=1}^{\log N} (t_a + 2t_{\mathrm{tr}})(B/2) + t_{\mathrm{tr}} \sum_{m=2}^{\log N - 1} [B + 2^{(m-2)} - 1] \\
&= \left(t_a \frac{B}{2}\right)\log N + t_{\mathrm{tr}}\left[\frac{N}{4} + (2B-1)\log N - 2B + 1\right]
\end{aligned}
\tag{2}
$$

Given now an $M \times M$ image, the time required to obtain the N partial histograms on a Homogeneous Multiprocessor consisting of N nodes is given as $T_{\mathrm{HIST}} = M^2/N\, t_a$. Thus, the total time required for the parallel algorithm is obtained as:

$$
T_{\mathrm{PAR}} = T_{\mathrm{HIST}} + T_{\mathrm{MRG}} = t_a\left(\frac{M^2}{N} + \frac{B}{2}\log N\right) + t_{\mathrm{tr}}\left[\frac{N}{4} + (2B-1)\log N - 2B + 1\right]. \tag{3}
$$

Similarly, the total time required to obtain the histogram of an $M \times M$ image on a uniprocessor is given as $T = M^2 t_a$. The speedup factor is therefore obtained as:

$$
\begin{aligned}
S = \frac{T}{T_{\mathrm{PAR}}} &= \frac{M^2 t_a}{t_a\left[\dfrac{M^2}{N} + \dfrac{B}{2}\log N\right] + t_{\mathrm{tr}}\left[\dfrac{N}{4} + (2B-1)\log N - 2B + 1\right]} \\
&= \frac{M^2}{\left[\dfrac{M^2}{N} + \dfrac{B}{2}\log N\right] + \gamma\left[\dfrac{N}{4} + (2B-1)\log N - 2B + 1\right]},
\end{aligned}
\tag{4}
$$

where $\gamma = t_{\mathrm{tr}}/t_a$ is the ratio of the transfer over the add times. The efficiency of the algorithm is given as:

$$
e = \frac{S}{N} = \frac{M^2}{\left[M^2 + \dfrac{B}{2}N\log N\right] + \gamma\left[\dfrac{N^2}{4} + (2B-1)N\log N + N(1-2B)\right]}. \tag{5}
$$

†It is assumed that the time spent to merge each of the B/2 elements of two partial neighboring histograms is $t_a + t_{\mathrm{tr}}$ which corresponds to the transfer of one element from the neighboring memory module plus the addition to the corresponding element in the local memory module. In addition, $B/2$ elements need to be transfered to a neighboring memory module at the conclusion of the addition step.

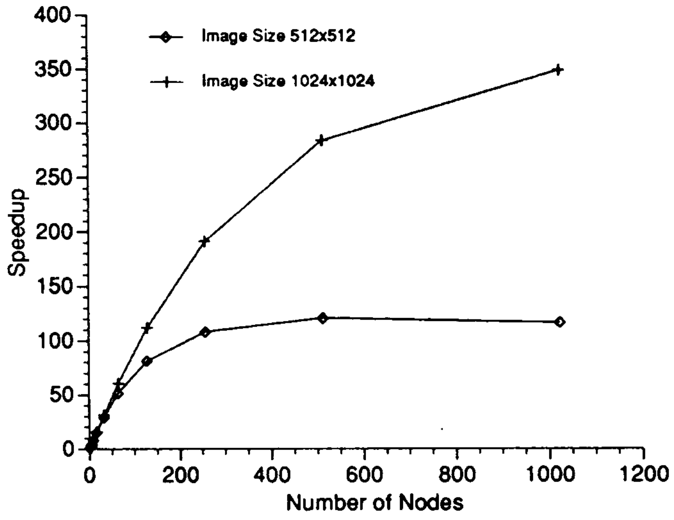


Fig. 5. Speedup vs the number of processors for the distributed histogram algorithm as obtained through analysis.

$$E = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \qquad W = \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix}$$

$$N = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} \qquad S = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

Fig. 6. The Sobel operator matrices for east, west, north and south.

Figure 4 shows the speedup factor, obtained through both simulations and equation (2) against the number of processors, for histogram calculation for images of varying sizes. The number of grey levels is set to 32.

3.3. *Edge detection using the Sobel operator*

Edge detection plays an important role in segmenting an image. Some of the edge detection algorithms are very attractive for parallel implementation. Mask operations are popular among them. The Sobel mask operator for example, was found effective in automatic visual inspection of spring assemblies [7] among other applications.

The Sobel operator was designed to approximate a discrete gradient function by appropriate directional mask operators [6]. The Sobel operator estimates the partial derivatives in four directions and is given in Fig. 6.

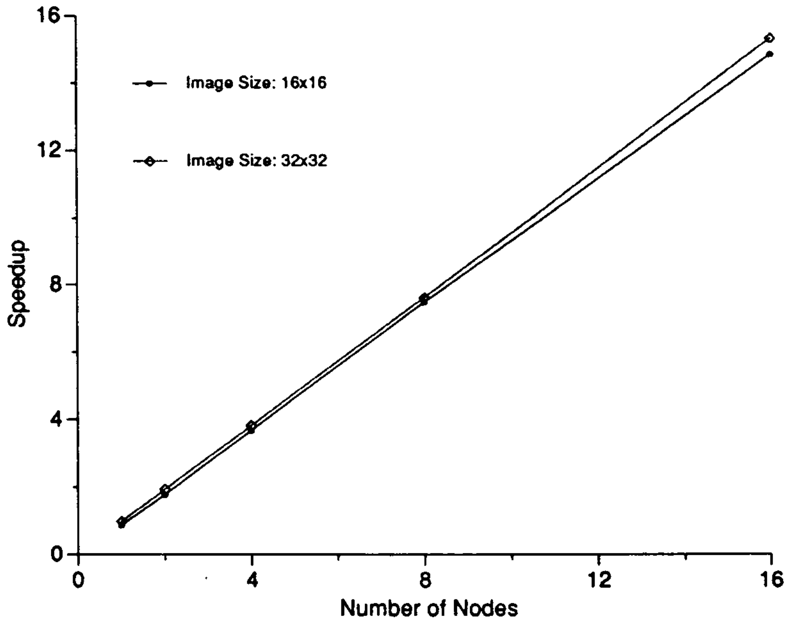


Fig. 7. Speedup vs the number of processors for the distributed edge detection algorithm (using the Sobel operator) as obtained through simulation.

Each edge detector is described by a set of templates whose application on an intensity function $f(x, y)$ results in a set of gradient arrays. The above procedure can be implemented in the multiprocessor as described below. The $M \times M$ image is divided into N strips with each strip having $(M \times M)/N$ pixel columns. Each processor calculates the gradient arrays of its strip, i.e. the corresponding edge enhanced values of each pixel. In a similar fashion as in smoothing, the edge enhanced values of the strip boundaries are calculated based on the values of the pixels in the neighboring processors. This is repeated for all the directional mask operators.

The partitioning algorithm is similar to the one used in smoothing. Figure 7 shows the speedup factor against the number of processors for two image sizes. The number of grey levels is fixed to 64 for both cases.

4. SPEEDUP CALCULATION FOR LOCAL OPERATIONS

Several image processing algorithms can be carried out by repeating a set of operations on well-defined neighborhoods of each pixel. Examples of such algorithms are the local averaging and edge detection algorithms mentioned previously. These algorithms can be effectively ported on the Homogeneous Multiprocessor, by assigning to each of the N nodes a strip of size $M \times M/N$. For this analysis, we assume that the width of the neighborhood is smaller than twice the width of the strip of pixels assigned to each processor. This condition guarantees that only neighboring strips are participating in the local calculations. Since individual pixels belong to several neighborhoods, it is most economical to replicate the values of the pixels from the two adjacent strips needed for the local calculations. If w were to denote the width of a neighborhood, then one would replicate the pixels contained within strips of width $(w - 1)/2$ adjacent to the current slip. Thus, each node would transfer $[(w - 1)/2] \times M$ pixels from each adjacent node. Denote now by t_{op} the time necessary to perform the set of operations on the pixel neighborhood, and by t_{tr} the time needed to transfer a single pixel from a neighboring memory module, then the achievable speedup is calculated as follows.

The time to carry out the algorithm on a uniprocessor is given as $T = M^2 t_{\mathrm{op}}$ while the execution time for the Homogeneous Multiprocessor is:

$$T_{\mathrm{PAR}} = \frac{M^2}{N} t_{\mathrm{op}} + 2\frac{w-1}{2} M t_{\mathrm{tr}} = \frac{M^2}{N} t_{\mathrm{op}} + (w-1) M t_{\mathrm{tr}}.$$

Therefore, the speedup is given as:

$$S = \frac{T}{T_{\mathrm{PAR}}} = \frac{M^2 t_{\mathrm{op}}}{\frac{M^2}{N} t_{\mathrm{op}} + (w-1) M t_{\mathrm{tr}}} = \frac{N}{1 + \frac{(w-1)N}{M} \frac{t_{\mathrm{tr}}}{t_{\mathrm{op}}}}. \tag{6}$$

Since $M \geqslant N$ and $t_{\mathrm{op}} \sim (w-1)^2 t_{\mathrm{tr}}$ it is easy to see that the speedup is larger or equal to $N/1 + (w-1)^{-1}$ and it approaches N for sufficiently large M and w. This has been confirmed experimentally as can be seen by the linear behavior of the speedup vs the number of processors in Figs 2 and 7.

If the number of processors N is large, it is possible that for large local operators, pixels in one strip will need to access pixels which are located in strips on processors which are not immediate neighbors to the current processor. This happens when N exceeds $2M/(s - 1)$. In such cases, one would simply replicate the strips from the neighboring memory modules at an extra cost of $2Mwt_{\mathrm{tr}}$ and use the same algorithm. The speedup would still be linear with the number of processors, albeit with a slightly lower slope than before.

5. CONCLUSIONS

In this work, we presented the performance of the following low-level vision algorithms: (i) smoothing; (ii) histogram generation; and (iii) edge detection by using the Sobel operator. The results were obtained through simulation experiments run on the simulator developed for the Homogeneous Multiprocessor, and are presented in Figs 2, 4–7.

As can be seen in Figs 2 and 7, both the smoothing and edge detection algorithms show an almost linear speedup with the number of processors involved. This was expected, since these algorithms require very little interaction between processors, apart from the occasional exchange of the values of the pixels located at the boundaries of the strips allocated to each of the processors, and it agrees with equation (6) predicting the behavior of the speedup for local algorithms implemented on the Homogeneous Multiprocessor.

Also, almost linear speedup was obtained for large images of the histogram presented in Section 3 above. Figures 4 and 5 present substantial agreement of the theoretical calculation [as given by equation (4)] for the speedup with that obtained through simulation.

As can be seen in Fig. 5, the speedup for the histogram generation algorithm slows down, and as a matter of fact it reverses itself when a large number of processors are used. This was also expected, since the algorithm used for the distributed merging of the partial histograms, requires $O(n)$ transfers. Hence given the number of grey levels in the histogram, there exists an optimum number of processors beyond which no substantial speedup gain can be achieved.

Our distributed merge algorithm could be improved if the pipelined transfer of the partial histograms started immediately after the merging of the neighboring histograms. Further improvements may be attained if the H-network were to be used instead of the pipelining mechanism for long range transfers for which the transfer time over the H-network is less than the transfer time obtained through the pipelining mechanism.

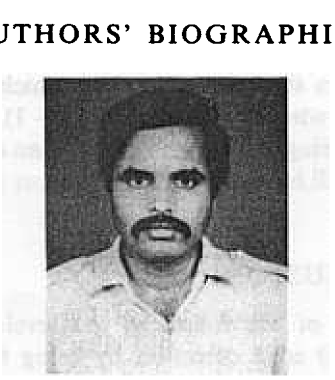
Nevertheless we were able to show substantial speedups for both cases of local algorithms (such as local averaging and edge detection) as well as nonlocal ones (e.g. histogramming) and for standard image sizes.

Acknowledgements—This research was supported by grants from the Natural Sciences and Engineering Research Council of Canada. A preliminary version of this paper was presented at the *Computer Vision and Pattern Recognition Conference*, Miami Beach, June 1986.

REFERENCES

1. N. J. Dimopoulos, K. F. Li, E. C. W. Wong, R. V. Dantu and J. W. Atwood, The homogeneous multiprocessor system: an overview. *Proc. 1987 Int. Conf. on Parallel Processing*, pp. 592–599 (1987).
2. K. F. Li and N. J. Dimopoulos, The performance analysis of the homogeneous multiprocessor proper. *Can. Elect. Engng Jl* **Jan,** 3–10 (1987).
3. N. J. Dimopoulos, On the structure of the homogeneous multiprocessor. *IEEE Trans Comput.* **C-34,** 141–150 (1985).
4. K. Hwang and F. Briggs, *Computer Architecture and Parallel Processing*. McGraw Hill, New York (1984).
5. P. M. Kogge and H. S. Stone, A parallel algorithm for the efficient solution of a general class of recurrence equations. *IEEE Trans. Comput.* **C-22,** 786–793 (1973).
6. M. D. Levine, *Vision in Man and Machine*. McGraw–Hill, New York (1985).
7. W. A. Perkins, Using circular symmetry and intensity profiles for computer vision inspection. *Comput. Graphics Image Process.* **17,** 161–172 (1981).
8. H. J. Siegel *et al.*, PASM: a partitionable SIMD/MIMD system for image processing and pattern recognition. *IEEE Trans. Comput.* **C-30,** 934–947 (1981).
9. L. J. Siegel, H. J. Siegel and P. H. Swain, Performance measures for evaluating algorithms for SIMD machines. *IEEE Trans. Software Engng* **SE-8,** (1982).

AUTHORS' BIOGRAPHIES



Ramanamurthy V. Dantu—Ramanamurthy Dantu received the Ph.D. degree from Concordia University, Montreal in 1990 in Electrical and Computer Engineering. Since April 1990, he has been a Member of the Scientific Staff in Bell-Northern Research Limited, Ottawa. His research interests include multiprocessor systems, computer vision, and congestion control in high-speed computer networks.

Nikitas J. Dimopoulos—Nikitas Dimopoulos received the B.Sc. degreee in Physics from the University of Athens, Greece in 1975 and the M.Sc. and Ph.D. degrees in electrical engineering from the University of Maryland, College Park in 1976 and 1980, respectively. From 1980 to 1988, he was an Assistant and then Associate Professor at the Electrical and Computer Engineering Department, Concordia University in Montreal. During the Summer of 1986 he was on leave with the Centre de Recherche Informatique de Montreal. From November 1986 to December 1987 he was on leave at the Jet Propulsion Laboratory in Pasadena, CA. In 1988, he joined the Electrical and Computer Engineering Department, University of Victoria, BC, where he is currently an Associate Professor. His research interests include multiprocessor systems, neural networks, low-level vision and expert systems.

Kin F. Li—Kin Li received his B.Eng. and Ph.D. degrees in electrical engineering from Concordia University, Montreal in 1982 and 1987, respectively. He is an Associate Professor in the Department of Electrical and Computer Engineering at the University of Victoria, BC, Canada. His research interests include distributed systems, expert systems and computer architecture.

R. V. Patel—Dr Patel received the B.Eng. degree in electronics from the University of Liverpool, U.K., in 1969 and the Ph.D. degree in electrical engineering from the University of Cambridge, U.K. in 1973. He has held post-doctoral and visiting faculty positions at the University of Cambridge, U.K., Lund Institute of Technology, Sweden, NASA Ames Research Center, U.S.A., University of Waterloo, Canada, Delft University of Technology, Holland and the Control Systems Centre, UMIST, U.K. At present, he is Professor of Electrical and Computer Engineering at Concordia University, Montreal, Canada. His current research is concerned with various computational and practical issues in control, robotics and neural networks. He has published over 130 technical papers in these areas and has co-authored two books: *Dynamic Analysis of Robot Manipulators: A Cartesian Tensor Approach* (Kluwer Academic Publishers, Boston, 1991); and *Multivariable System Theory and Design* (Pergamon Press, Oxford, 1982). He was an Associate Editor of the *IEEE Transactions on Automatic Control* from 1987 to 1990, and of *Automatica* from 1985 to 1991. Dr Patel was elected a Fellow of the IEEE in January 1992.

Asim J. Al-Khalili—Asim Al-Khalili received the M.Sc. and Ph.D. degrees in systems engineering from Strathclyde University, Glasgow U.K. in 1974. He has previously worked with GEC-Ellio Automation and Parson's Brown International as a Systems Engineer. Currently he is a Professor of Computer Engineering at Concordia University, Montreal, Canada. His interests include microcomputer systems architecture and applications. VLSI architecture VLSI design.