December 1989

UILU-ENG-89-2242
CSG-118

COORDINATED SCIENCE LABORATORY
College of Engineering

NETRA—A PARALLEL ARCHITECTURE FOR INTEGRATED VISION SYSTEMS II: ALGORITHMS AND PERFORMANCE EVALUATION

Alok N. Choudhary
Janak H. Patel
Narendra Ahuja

UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION Unclassified	1b. RESTRICTIVE MARKINGS None
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution unlimited
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S) UILU-ENG-89-2242 (CSG-118)	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION Coordinated Science Lab University of Illinois 6b. OFFICE SYMBOL (If applicable) N/A	7a. NAME OF MONITORING ORGANIZATION NASA
6c. ADDRESS (City, State, and ZIP Code) 1101 W. Springfield Avenue Urbana, IL 61801	7b. ADDRESS (City, State, and ZIP Code) NASA Langley Research Center Hampton, VA 23665
8a. NAME OF FUNDING/SPONSORING ORGANIZATION NASA 8b. OFFICE SYMBOL (If applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER NASA NAG 1-613
8c. ADDRESS (City, State, and ZIP Code) see 7b	10. SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO. / PROJECT NO. / TASK NO. / WORK UNIT ACCESSION NO.

11. TITLE (Include Security Classification)
NETRA - A Parallel Architecture for Integrated Vision Systems II: Algorithms and Performance Evaluation

12. PERSONAL AUTHOR(S)
Choudhary, A. N., Patel, J. H. and Ahuja, N.

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15. PAGE COUNT
Technical	FROM _____ TO _____	1989 December 7	43

16. SUPPLEMENTARY NOTATION

17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	multiprocessor architecture, parallel processing, vision, image processing, parallel algorithms, performance evaluation

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

In part 1 of this paper we presented architecture of NETRA [1]. This paper presents performance evaluation of NETRA using several common vision algorithms. Performance of algorithms when they are mapped on one cluster is described. It is shown that SIMD, MIMD and systolic algorithms can be easily mapped onto processor clusters, and almost linear speedups are possible. For some algorithms, analytical performance results are compared with implementation performance results. It is observed that the analysis is very accurate. Performance analysis of parallel algorithms when mapped across clusters is presented. Mappings across clusters illustrate the importance and use of shared as well as distributed memory in achieving high performance. The parameters for evaluation are derived from the characteristics of the parallel algorithms, and these parameters are used to evaluate the alternative communication strategies in NETRA. Furthermore, the effect of communication interference from other processors in the system on the execution of an algorithm is studied. Using the analysis, performance of many algorithms with different characteristics is presented. It is observed that if communication speeds are matched with the computation speeds, good speedups are possible when algorithms are mapped across clusters.

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT	21. ABSTRACT SECURITY CLASSIFICATION
☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT. ☐ DTIC USERS	Unclassified
22a. NAME OF RESPONSIBLE INDIVIDUAL	22b. TELEPHONE (Include Area Code) / 22c. OFFICE SYMBOL

NETRA - A Parallel Architecture for Integrated Vision Systems II: Algorithms and Performance Evaluation

Alok N. Choudhary, Janak H. Patel and **Narendra Ahuja**

Coordinated Science Laboratory
University of Illinois
1101 W. Springfield
Urbana, IL 61801

Index Terms - Multiprocessor Architecture, Parallel Processing, Vision, Image Processing, Parallel Algorithms, Performance Evaluation

Corresponding Author : Prof. Alok N. Choudhary

Electrical and Computer Engineering Department
Science and Technology Center
Syracuse University,
Syracuse, NY 13244

NETRA - A Parallel Architecture for Integrated Vision Systems II: Algorithms and Performance Evaluation

Alok N. Choudhary, Janak H. Patel and Narendra Ahuja

Coordinated Science Laboratory
University of Illinois
1101 W. Springfield
Urbana, IL 61801

Abstract

In part 1 of this paper we presented architecture of NETRA [1]. This paper presents performance evaluation of NETRA using several common vision algorithms. Performance of algorithms when they are mapped on one cluster is described. It is shown that SIMD, MIMD and systolic algorithms can be easily mapped onto processor clusters, and almost linear speedups are possible. For some algorithms, analytical performance results are compared with implementation performance results. It is observed that the analysis is very accurate. Performance analysis of parallel algorithms when mapped across clusters is presented. Mappings across clusters illustrate the importance and use of shared as well as distributed memory in achieving high performance. The parameters for evaluation are derived from the characteristics of the parallel algorithms, and these parameters are used to evaluate the alternative communication strategies in NETRA. Furthermore, the effect of communication interference from other processors in the system on the execution of an algorithm is studied. Using the analysis, performance of many algorithms with different characteristics is presented. It is observed that if communication speeds are matched with the computation speeds, good speedups are possible when algorithms are mapped across clusters.

This research was supported in part by National Aeronautics and Space Administration Under Contract NASA NAG-1-613.

Table of Contents

1. Introduction

In the first part of this paper [1], we described the architecture of NETRA, its features, and presented an analysis of inter-cluster communication strategies. This paper presents performance evaluation of NETRA using several common image processing and vision algorithms. The performance of components of NETRA is illustrated using algorithms with varying characteristics and communication requirements. The results are used to identify the bottlenecks of NETRA and to suggest methods to improve and refine the architecture.

For each algorithm we present one or more mapping strategies, its performance evaluation, and a discussion of the results. The algorithms include two-dimensional (2-D) FFT, convolution, separable convolution, hough transform, sobel edge detection, and median filtering. Some of the algorithms are part of the Image Understanding Benchmark presented in [2]. The approach to performance evaluation has been described in the first paper. To evaluate parallel algorithms on a cluster, we explore alternative mapping strategies, and computation modes. Some of the algorithms have been implemented on a simulated cluster, and we show that the analysis provides very accurate results. We also discuss performance of the algorithms when they are mapped across multiple clusters. The results are used to compare alternative inter-cluster communication strategies.

Table 1 depicts the parameters used for performance evaluation unless specified otherwise. The values of computation and communication speeds are chosen to be conservative. We think that much faster processors and communication links are possible, and available with current technology, and therefore, the performance results presented in this paper are conservative. Since the goal is to study the architecture behavior rather than present raw performance numbers, we have chosen the above parameters.

This paper is organized as follows. Section 2 contains analysis and performance of various algorithms on one cluster. We show mappings in SIMD as well as MIMD modes. In Section 3 we present implementation results for

Table 1 : Parameters for Performance Evaluation

Total No. of Processors N_p	512
Cluster Size P_c	8-128
No. of Processors/Port P_p	4
Image Size $N \times N$	512 X 512
Memory Modules M	128
Processor Speed	5 MIPS, 5 MFLOPS
Network Speed (Block Transfer)	20 Mbytes/Sec.
Traffic Intensity for Interference ($m \times t$)	0.1,0.4,0.8

four algorithms, two of which (median and sobel) are parts of the Image Understanding Benchmark Algorithms [2], and note that the analytical results are comparable to the implementation results. Section 4 discusses the performance of algorithms when they are mapped across multiple clusters. We present results for both multistage interconnection network between cluster processors as well as a global bus interconnection. Finally, in Section 5, summary are presented. For further details, the reader is referred to [3].

2. Parallel Algorithms on a Cluster

In this section we describe how various algorithms with different characteristics can be mapped onto a processor clusters The algorithms are classified according to their computation and communication characteristics as well as their suitability for implementation on SIMD or MIMD architectures. The purpose of this section is manifold : first, we illustrate how SIMD, systolic, and MIMD type algorithms can be efficiently mapped onto the cluster; second, how algorithms that need both types of computations can be mapped; and finally, to show how algorithms from different classes can be mapped onto the cluster. In the evaluation we discuss the computation, communication and storage requirements for the algorithms.

The following is a brief review of the classes of algorithm categorized according to their communication requirements, and is presented in [1].

(1) *Local Fixed* - In these algorithm, the output depends on a small neighborhood of input data in which the neighborhood size is normally fixed.

(2) *Local Varying* - Like the local fixed algorithms, the output at each point depends on a small neighborhood of input data. However, the neighborhood size is an input parameter and is independent of the input image size.

(3) *Global Fixed* - In such algorithms each output point depends on the entire input image. However, the computation is normally input data independent.

(4) *Global Varying* - Unlike global fixed algorithms, in these algorithms the amount of computation and communication depends on the image input as well as its size. That is, the output may depend on the entire image or may depend on a part of image.

2.1. 2-D Convolution

2-D Convolution is a Local varying type of algorithm. A 2-D convolution of an NxN image I(i,j), 0<=i,j<=N, with a kernel W(i,j), 0<=i,j<=w, can be expressed as follows :

$$G(i,j) = \sum_{m=j-w/2}^{m=j+w/2} \sum_{n=i-w/2}^{n=i+w/2} I(n,m) * W((i+w/2-n) \bmod w, (j+w/2-n) \bmod w)$$

In other words, each point in the output is replaced by a weighted sum of a window *wxw* around it.

The convolution algorithm will illustrate how to map SIMD and systolic algorithms onto a processor cluster when the number of processors is much smaller than the problem size. The approach to map the algorithm onto a cluster is to transform 2-D convolution to a 1-D convolution by unfolding the window in one dimension, and without incurring additional steps for unfolding. In other words, the amount of computation in the corresponding 1-D convolution remains the same as that in 2-D convolution. Figure 1 shows a cluster of 64 processors. The interconnection between processors shows all the connections required to perform the convolution operation. However, all the connections are not needed at the same time. We shall observe that only one input and one output connection is sufficient at any time, and that the flexibility of the crossbar can be used to obtain all the desired interconnections efficiently.

Each pixel is logically mapped onto a separate processor (as if there were as many processors available as there are pixels). Actually the image is folded in two dimensions like a torus, and multiple pixels are mapped onto one processor. For a cluster size P, (assume $P = pxp$), each processor has $M = N^2/P$ pixels in its local memory. In general, pixel (i,j) ; $0 \le i \le N-1$, $0 \le j \le N-1$ is mapped to processor ((i mod p), (j mod p)). Therefore, this mapping preserves the adjacency of any two pixels even though the image is folded.

Figure 1 shows the flow of the distribution of data for window size 5×5. A small window is embedded in a larger one and therefore, same connections can be used for a larger window size with the addition of new connections for extra steps. In other words, all the computations and communication needed in a small window can be used for a larger window. For example, 5×5 window requires all the connections that are required by 3×3 window. The algorithm performs the convolution by each processor distributing its pixel values to the neighborhood in a pipelined manner.

In the following algorithm North, South, East and West Neighbors are defined treating the image as a torus. For a processor P(i,j), N,S,E,W neighbors are defined as follows.

N = ((i-1),j), if (i-j) < 0, then N = ((i-1 + p), j)

S = ((i+1) mod p, j)

E = (i, (j+1) mod p)

W = (i, (j-1)), if (j-1) < 0, then W = (i, (j-1+p))

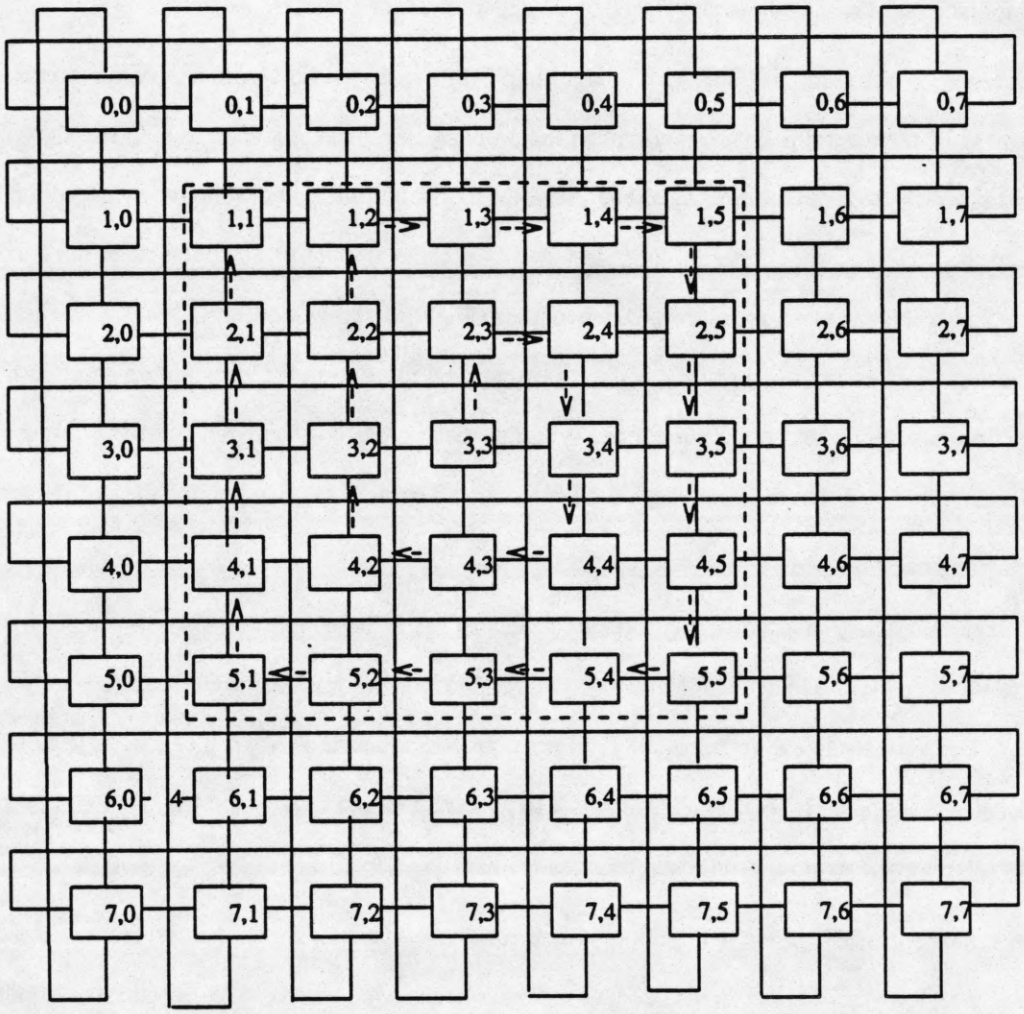


Figure 1 : Mapping on the Cluster for Convolution

At any step in the execution of the algorithm, all the processors have the same neighbor connection. For example, at a given instance, if processor P_i is connected to P_j such that P_j is the North neighbor of P_i, then all other processors will also be connected to their north neighbors. Figure 1 shows how processor (3,3)'s values will be distributed. All the processors follow the same pattern. Note that the above definition of neighbors is for logical neighbors because it uses pixel adjacency rather than processor adjacency. The above definition does not imply any physical connections between processors because the connections are programmed according to pixel adjacency.

The algorithm works as follows (Figure 2): The DSP broadcasts the convolution weights to all the processors. Each processors multiplies its M pixels with the central weight value. In Figure 2 the data values at each processor are stored in a linear array and subscript (i,j) means the data value i in the connection number j. The intermediate values are stored in the running variable for each of the M pixels. The image is then shifted in a spiral manner (as shown in Figure 1). If the image is shifted north then the processors now multiply the pixel values with the south weight. This process is repeated w^2-1 times , i.e., for each weight. The following properties characterize the above algorithm. First, the mapping is independent of problem or cluster size. That is, the mapping will work for all problem sizes. Second, the number of times the interconnection needs to be changed only depends on the convolution kernel size. Furthermore, at any time only one input and one output connection is required. By storing the connection patterns in the crossbar memory the switching time becomes negligible. Third, it is possible to overlap computation and communication by writing the pixel to the output port as soon as it is multiplied by the appropriate weight in the current processor. The above algorithm illustrates that SIMD algorithms can be mapped efficiently on to the processor clusters using the flexibility and programmability of the interconnection.

ALGORITHM CONVOLUTION
All the processors work in SIMD lock-step fashion.
DSP broadcasts the convolution kernel.
Set up Connection_array of size *wxw* in the crossbar memory by choosing.
first *wxw* connections from the set.
{N,E,S,S,W,W,N,N,N,E,E,E,S,S,S,W,W,W,W,N,N,N,N,E,..}.

$M := \left\lceil \frac{N^2}{P} \right\rceil$

For i = 1 to M do (in parallel)
Result(i) := $w_{i,i} * data(i)$
End_For

For j = 1 to *wxw* do (in parallel)
Set up appropriate connections on the crossbar as follows.
connection(j) := connection_array(j)
For i = 1 to M do (in parallel)
Send data (pixels) on the output port to the connected neighbor.
At the same time receive data from its input port.
Result(i) := $Result(i) + w_{i,j} * data(i,j)$
End_for
End_for
END CONVOLUTION

Figure 2 : 2-D Convolution

The computation time decreases as the number of processors increases. The communication time per pixel only depends on the kernel size. The following formulae present the computation and communication times in terms of multiplication and addition operations. The factor t_{fl} denotes the floating point speed of a processors in terms of its normal instruction execution speed.

$$t_{cp} = 2 \times t_{fl} \times \left\lceil \frac{N^2}{P} \right\rceil \times w^2$$

$$t_{comm} = \left\lceil \frac{N^2}{P} \right\rceil \times w^2$$

$$t_{sw} = w^2 - 1$$

$$t_{tot} = \text{Max}(t_{cp}, t_{comm} + t_{sw})$$

Figure 3 shows the performance of the 2-D convolution on a processor cluster. The processing time has been computed assuming a 2 MFLOP processor. The Figure shows two speedup graphs, one with communication overlap and the other with additive communication. The computation time decreases linearly as the number of processors increases. The total communication time per processor also decreases linearly, but the communication time per pixel remains constant. An important observation one can make here is that it is essential that the communication and

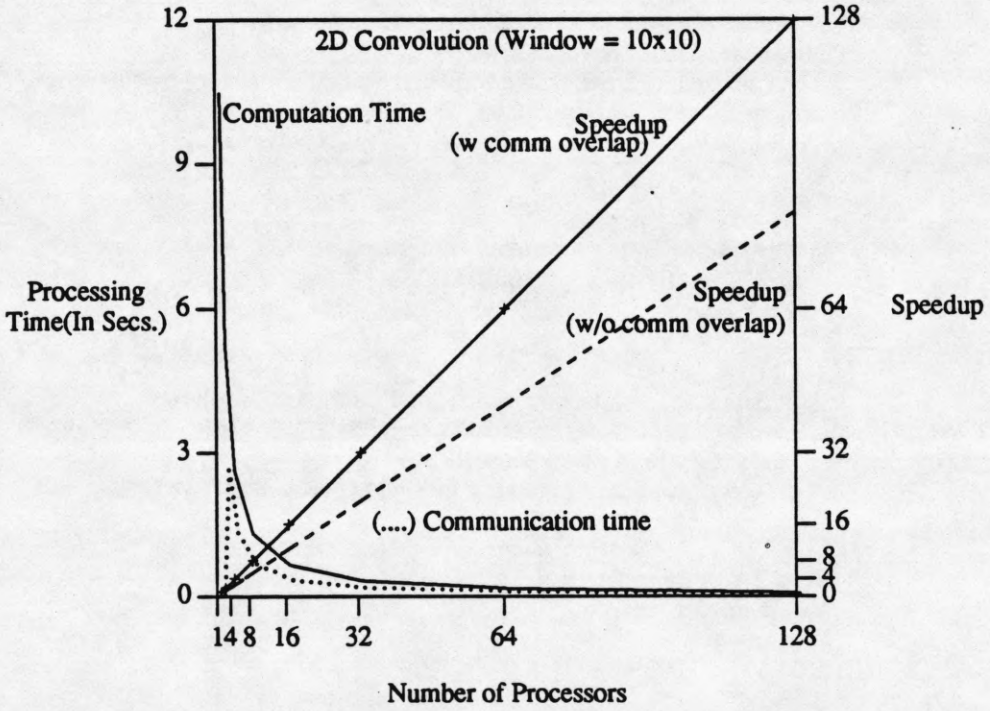


Figure 3 : Performance of 2-D Convolution on a Processor Cluster

computation overlap in order to obtain linear speedups. However, if the interconnection speed is not matched with the computation speed, then overlap will not be possible. Having a fast crossbar without arbitration delays provides the necessary communication speed to obtain linear speedups. Note that since computation and communication can overlap, this mapping is also applicable to systolic algorithms.

2.2. Separable Convolution

A two dimensional convolution is separable if it can be replaced by two one dimensional convolutions. The main advantage of separability is that the computational requirements per pixel are reduced from $2w^2$ to $4w$. We illustrate the mapping by giving an MIMD algorithm to run on a cluster.

The data is partitioned among the processors as follows. Each processor is assigned N/P rows of the data. Processor P_i gets rows $(i-1)\times N/P$ to $i\times N/P - 1$. Each processor computes convolution along the rows using a window of size w. Once processor P_i finishes convolution along the rows, it needs rows $(i-1)\times N/P - w/2$ to $(i-1)\times N/P - 1$, from processor P_{i-1}, and similarly , it needs the bottom $w/2$ rows from $i\times N/P$ to $i\times N/P + w/2 - 1$ from processor P_{i+1}. Therefore, a processor needs to communicate with only two processors to obtain the desired intermediate data. The boundary processors P_0 and P_{P-1} only need to communicate with one other processor. Note that if the granule size with each processor is less than $w/2$ (i.e., $N/P < w/2$) then the processors need to exchange data with the number of processors given below by T_{sw}. Each processor computes convolution along the columns in its granule. The following are computational and communication requirements of the algorithm.

$$t_{cp} = \frac{t_{fl}\times N^2\times 4\times (w/2+1)}{P}$$

$$t_{comm} = 2\times N\times w$$

$$T_{sw} = \left\lceil \frac{w\times P}{N} \right\rceil$$

The amount of computation per pixel in separable convolution is a function of w for a $w\times w$ kernel unlike in 2-D convolution where it is a function of w^2. The amount of communication in separable convolution is fixed as shown in Figure 4. Therefore, the speedup is not as much as in the case of 2-D convolution. There are two reasons for smaller speedup. First, the communication does not reduce with increasing number of processors because each processor needs to exchange $w/2$ rows of intermediate results with two adjacent processors. Secondly, since the computation per pixel itself is small, the communication overhead as a fraction of computation time is large.

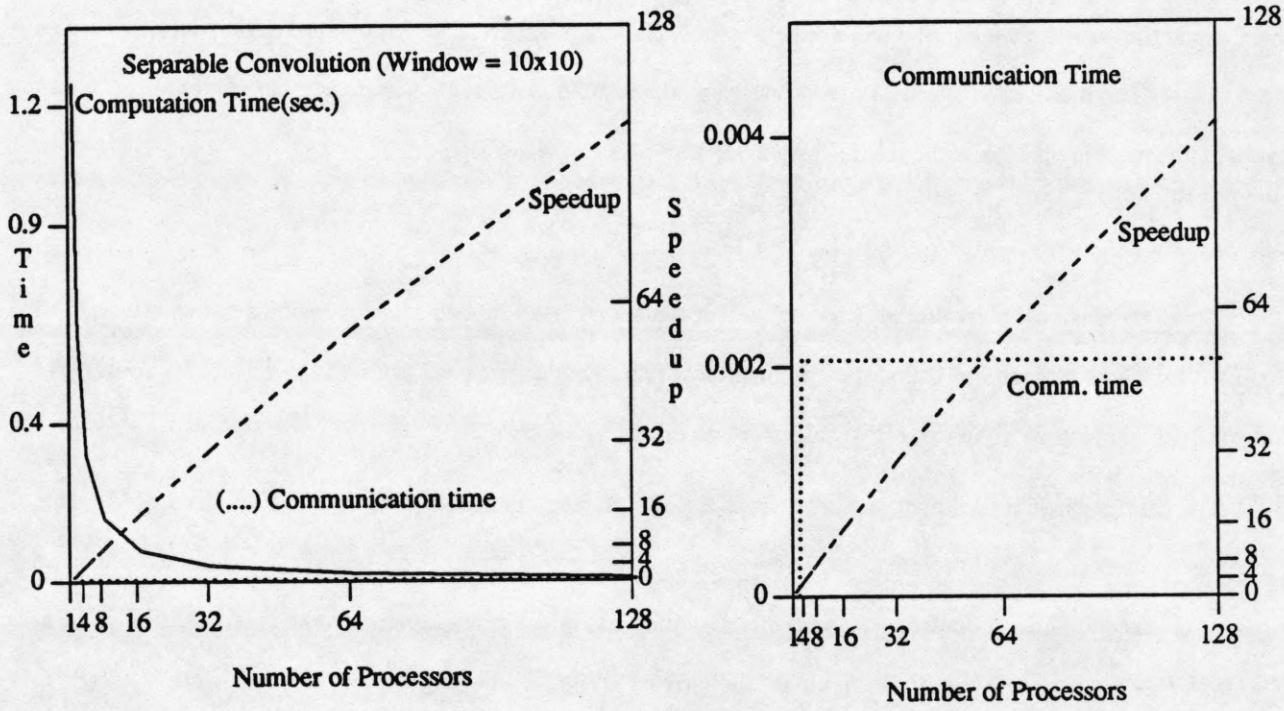


Figure 4 : Performance of Separable Convolution on a Processor Cluster

2.3. 2-D FFT

2-D FFT is a Global Fixed algorithm. For an image I(k,l), 0<=k,l<=N, the corresponding 2-D FFT is given by

$$F(m,n) = \sum_{k=0}^{N-1} \sum_{l=0}^{N-1} I(k,l)\, e^{-2\pi j(km+ln)/N},\ 0<=m,n<=N-1$$

where $j = \sqrt{-1}$. A nice property of the 2-D FFT is that it can be computed in two steps : a one dimensional N point FFT along the rows followed by a one dimensional N point FFT of the columns of row FFT values, or vice versa. We use this property to map 2-D FFT on the cluster processors. The algorithm consists of three phases : 1-D FFT computation along rows, transposing the intermediate results and, 1-D FFT along the columns.

Figure 5 describes the algorithm. In the first phase each processors is assigned N/P rows. Let us denote the sequence of rows with processor P_i as Granule(i). Also, let's divide each granule into P equal blocks of size N^2/P^2 as shown in Figure 6. B(i,j) denotes a block of size N^2/P^2 with processor P_i , 0<=j<=P-1. Each processors computes the 1-D FFT along the rows of its granule. Then in the second phase, the processors communicate with each other in the following manner to transpose the intermediate results. A processor P_i sends block B(i,j) to processor P_j for all 0<=j<=P-1, $j \neq i$. Each processor needs to communicate and exchange a block with every other processor in the cluster. However, by performing the communication systematically, the transpose can be achieved without

any conflicts as described in the algorithm. Finally, each processor computes 1-D FFT along the columns.

ALGORITHM 2-D FFT

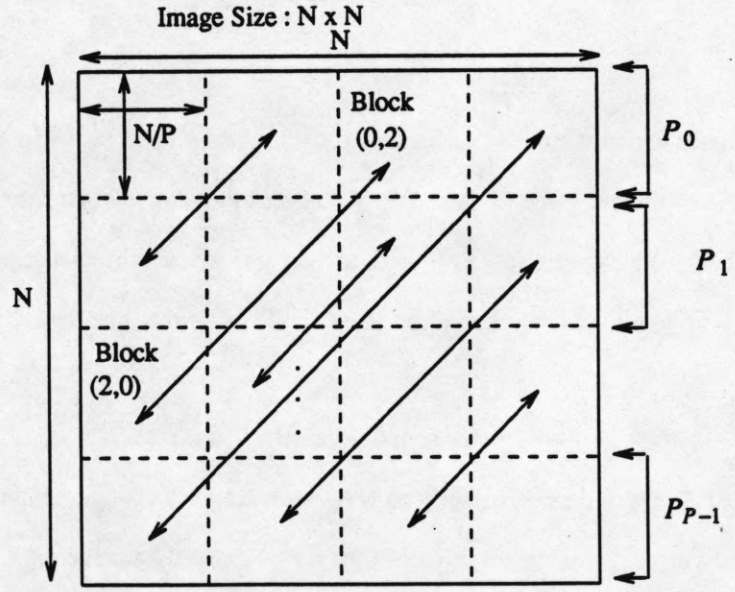
Each processor P_i receives granule(i) of rows.
/* The following description is with respect to processor P_i */
$M := \left\lceil \frac{N^2}{P} \right\rceil$
For k = 1 to M do
compute 1-D FFT of row(k) of granule(i)

For j = 1 to M do (i ≠ j)
k = i+j mod P
connect P_i to P_k
send Block(i,j) to P_k
receive Block(j,i) from P_k

For k = 1 to M do
compute 1-D FFT of row(k) of granule(i)

END 2-D FFT

Figure 5 : 2-D FFT



The figure shows data exchange
needed to transpose intermediate data.

Figure 6 : An Example of Mapping 2-D FFT onto Four Processors

The 1-D FFT of a row of N pixels can be computed[4] The constant of multiplication is 6, i.e., to perform N point 1-D FFT it takes approximately 6NlogN floating point operations. Therefore, the computation time for the above algorithm is (for both row and column steps)

$$t_{cp} = \frac{12 \times t_{fl} \times N \times N \times \log_2 N}{P}$$

The communication time to transpose the intermediate results is

$$t_{comm} = 2 \times (P-1) \times N^2/P^2$$

and the number of switch settings are, $t_{sw} = P-1$.

Even though FFT is a Global Fixed algorithm, in the above mapping both the computation and communication times reduce as the number of processors increases. In other words, both computation and communication are decomposable for parallel processing. Therefore, if the communication is achieved without conflicts (as in our case), we can obtain linear speedups.

Figure 7 and 8 show the performance of 2-D FFT algorithm on a processor cluster. From Figure 7 we can observe that almost linear speedup can be obtained. The variation of the communication time as a function of the number of processors is shown is Figure 8. Note that communication time curve follows the computation time curve in its shape and the communication is completely decomposable.

2.4. Hough Transform

Hough transform is used to detect curves (such as lines, circles, and ellipses) in an input image[5]. The computation is performed in the parameter space of the curve. Consider the example of detecting line segments. computation is performed in the (r,θ) parameter space. If there exists a line whose normal distance from the origin is r, the normal makes an angle θ with the x-axis then if the point (x,y) lies on that line than the following equation is satisfied.

$$r = x\cos\theta + y\sin\theta$$

First, r and θ are quantized. The quantization depends on how much accuracy is required in the final result. Assume that the maximum value of r br r_{max} maximum value of θ be θ_{max} (generally π). Then if r_{res}, θ_{res} are the resolutions used for quantization, the total number of accumulator cells in the computation are $r_{max}.\theta_{max}/r_{res}.\theta_{res}$, the number of rows and columns in the accumulator array being $\theta_c = \theta_{max}/\theta_{res}$ and $\rho_c = r_{max}/r_{res}$, respectively. The algorithm involves two major steps. The first step is to accumulate votes in the accumulator array for various digitized r and θ values. The second step is to compute local maxima in the output of the first step. The first step is

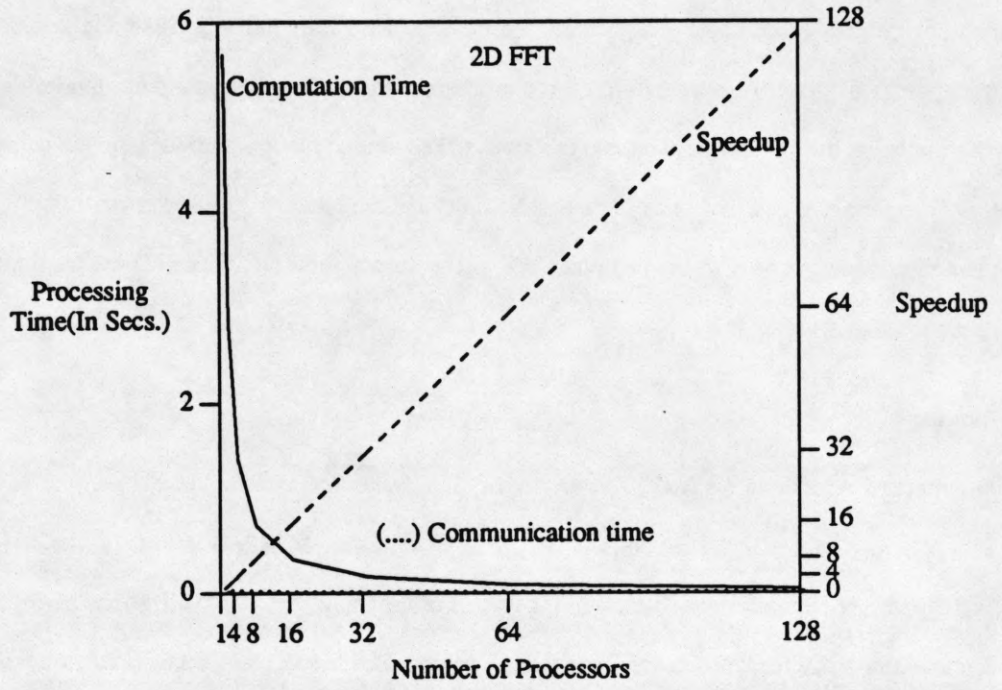


Figure 7 : Performance of 2-D FFT on a Processor Cluster

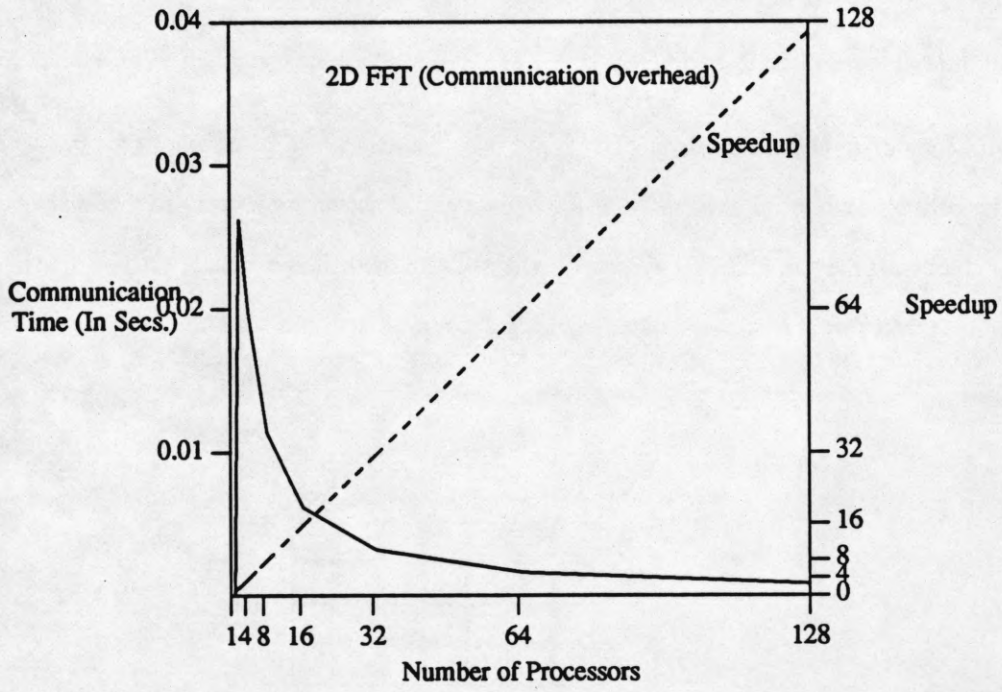


Figure 8 : Communication Time for 2-D FFT on a Processor Cluster

regular and suitable for SIMD implementation. The second step is more suitable for MIMD implementation because the output is global data dependent. For example, an image containing many lines will result in many more maxima

than an image containing a few lines, and therefore, the required computation will vary.

Hough transform is Global Varying algorithm. Furthermore, the communication can not be decomposed. We present two mappings of the Hough transform on the processor cluster. The first mapping divides the input image into as many granules as the number of available processors. The second mapping divides the the parameters among the processors. The former is referred to as "data partitioning" and the latter as "parameter partitioning." We discuss advantages and disadvantages of both the mappings and also compare their computation time, communication time and memory requirements.

Data Partitioning

Assume that the input image is NxN, and to simplify the discussion assume that the number of available processors is $P = p^2$. The image is partitioned into N^2/p^2 blocks. Processor P(i,j) works on block i*p + j , where $1 \le i, j \le p$. Each processor computes the vote count for its part of the image for all quantizations of θ values. Figure 9 shows the accumulator array for a processor. Note that each processor has to maintain a complete accumulator array of size $\rho_c \times \theta_c$, and update the appropriate vote count computed from its share of the image. The algorithm ACCUMULATE_COUNT in Figure 9 shows the computation for this step. The computation time to compute the accumulator array is time taken to perform $2 \times \left\lceil \frac{n^2}{p^2} \right\rceil \times f \times \theta_c$ multiplications, and half as many additions, where f is the largest fraction of significant pixels in a block and θ_c is the number of quantizations for θ. The next step is to combine the partial results of all the processor to obtain a global accumulator array so that maxima can be determined. For combining the partial results we propose the tree sum method in which, at each step, twice as many processors combine their partial results, therefore requiring $2 \times \log p - 1$ steps.

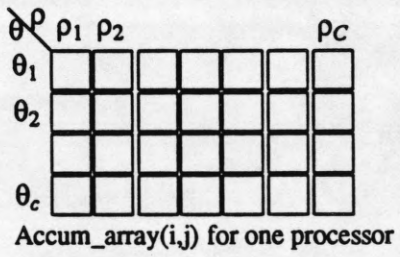


Figure 9 : Accumulator array for Hough Transform

```
ALGORITHM ACCUMULATE_COUNT
    Each processor P_i, 1≤i≤p^2 does the following (in parallel)
     For j = 1 to θ_c do
    For each (x,y) in the subimage such that (x,y) is significant do
     /*significant means black pixel or edge element*/
       compute r(θ_j) = x cosθ_j + y sinθ_j
       Accum_array(θ_j,r(θ_j)/r_res) = Accum_array(θ_j,r(θ_j)/r_res) + 1
     End_For
    End_For
END ACCUMULATE_COUNT
```

Figure 10 : Algorithm to Compute Votes in Hough Transform

The algorithm ACCUMULATE_SUM in Figure 11 performs the merging of partial results. The processors are numbered from 0 to p^2-1. A processor with number k, $0 \le k \le p^2-1$ corresponds to a processor (i,j) such that $k=i*p+j$.

Following this step, processor P_0 has the entire accumulator sum. The next step is to distribute this global accumulator sum to all the processors so that computation for local maxima can be performed in parallel. This step needs only one step. Processor P_0 broadcasts the entire array to all the processors using the broadcast facility of the

```
/* Accum_array_k(i,j) denotes the accumulator cell (i,j)
  the Accumulator array of processor k. */

ALGORITHM ACCUMULATE_SUM
  For i = 0 2×log_2 p−1 do
    For all processors P_j do in parallel (0≤j≤p^2−1)
        If j mod 2^(i+1) = 2^i then
          Connect P_j --> P_(j−2^i)
          For k = 1 to θ_c do
           For l = 1 to ρ_c do
                 Send Accum_array_j(k,l) P_(j−2^i)
                 Accum_array_(j−2^i)(k,l) := Accum_array_(j−2^i)(k,l)
                  + Accum_array_j(k,l)
           End_For
          End_For
        End_If
    End_For
  End_For
END ACCUMULATE_SUM
```

Figure 11 : Algorithm to Accumulate the Vote Count

crossbar. After the broadcast step, each processor performs a search for local maxima on its share of the accumulator rows $\left\lceil \frac{\theta_c}{p^2} \right\rceil$. In this algorithm, for each entry in its block of the accumulator array, the processor determines whether the entry represents a local maxima.

In summary, the total computation and communication time requirements for the entire hough transform algorithm using the data partitioning are as follows.

$$t_{cp} = 3 \times t_{fl} \times \left\lceil \frac{N^2}{p^2} \right\rceil \times f \times \theta_c + \theta_c \times C \times logP + \theta_c \times \rho_c \times w^2/p^2$$

where, the first term is for computing the votes, the second term is to sum the accumulator array, and the third term is for looking for local maxima in a window of size w^2. The communication time for this algorithm is

$$t_{comm} = (logP+1) \times \theta_c \times \rho_c$$

and the number of switch settings are $t_{sw} = \log P+1$.

Unlike 2-D FFT, the communication is not decomposable. In other words, the communication increases as the number of processors increases in a cluster. Figure 12 shows the computation and communication time along with the speedup for hough transform. Even though the computation time for hough transform decreases as the number of processors increases, the computation is not completely decomposable. The second term (to combine partial results) of t_{cp} increases as a log function of the number of processors. Furthermore, the communication overhead to combine accumulator arrays also increases logarithmically with the number of processors. Consequently, for a large number of processors, the communication time becomes comparable to the computation time (as shown in Figure 12), and that results in degradation in the speedup.

Parameter Partitioning

In this mapping, instead of partitioning the data among the processors, the parameters space is partitioned. Each processor works on the entire image but computes the vote count for only few θ values. Each processor computes all ρ values for its share of θ values. If there are p^2 processors, then each processor works on $n = \theta_c/p^2$ values of θ. There are several advantages to this mapping, both in terms of communication and implementation at each processor. First, when looking for local maxima later, a processor needs to communicate with only two other processors to obtain the upper and lower boundary rows of the Accumulator array. Second, we introduce additional data structures to make the search for local maxima efficient, where instead of searching for the local maxima in the

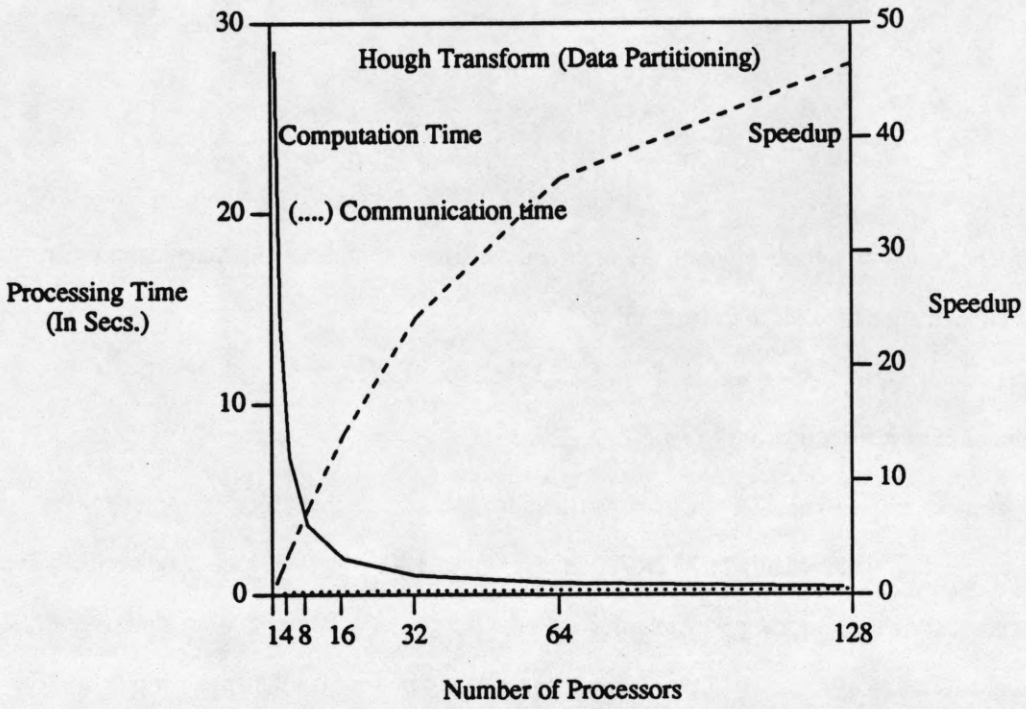


Figure 12 : Performance of Hough Transform (Data Partitioning)

entire accumulator array, only a fraction indicating possible local maxima need to be searched. Furthermore, the processor can store $\sin\theta$, $\cos\theta$ values for its allocated n values of θ in its registers, since only a few values need to be stored. This results in saving on local memory access delays which would occur if all quantized $\sin\theta$ and $\cos\theta$ values are stored with each processor in its local memory. The algorithm to compute the accumulator array at each processor is similar to that in the case of data partitioning except that each processor works on the entire image but only on its own part of the parameters.

A brief explanation of the algorithm is as follows. In the first step (computing votes), the algorithm computes value of ρ for each significant pixel for all θ values. It then increments the appropriate count in the Accumulator array. If the count increases beyond a certain threshold value, there exists a possibility of this being a local maxima. Therefore, another array called the Link_array is updated marking this fact. This step reduces the search space when looking for local maxima since normally a very small fraction of the image contributes to lines and entire Accumulator array need not be searched when looking for local maxima. Once the above computation is finished for the entire image, processor P_i communicates with P_{i+1} and P_{i-1} to obtain the boundary rows of the Accumulator array. Then the local maxima are computed in the Accumulator array using the information available in

Link_array. There is a need to search only those entries in the Accumulator array for a local maxima which are marked by the Link_array. The computation, communication and memory requirements for this mapping are as follows.

$$t_{cp} = 3 \times t_{fl} \times \left\lceil \frac{N^2}{p^2} + 1 \right\rceil \times f \times \theta_c + \theta_c \times \rho_c \times w^2/p^2$$

where the first term is for computing the votes and the second term is to for local maxima in a window of size w^2. The communication time for this algorithm is

$$t_{comm} = 2 \times \rho_c,$$

and the number of switch settings are $t_{sw} = 2$.

The memory requirements of the two partitionings are comparable. For example, for an image size of $512x512$, value of ρ_c will typically be $512 \times \sqrt{2}$, and θ_c will be 180. However, each pixel normally is a byte where as each accumulator cell is an integer. Assuming a 4 byte integer, in data partitioning a processor has to store the entire accumulator array of size 521 Kbytes (approximately), and in the second mapping a processor has to store the entire image (256 K bytes), and its part of the accumulator array.

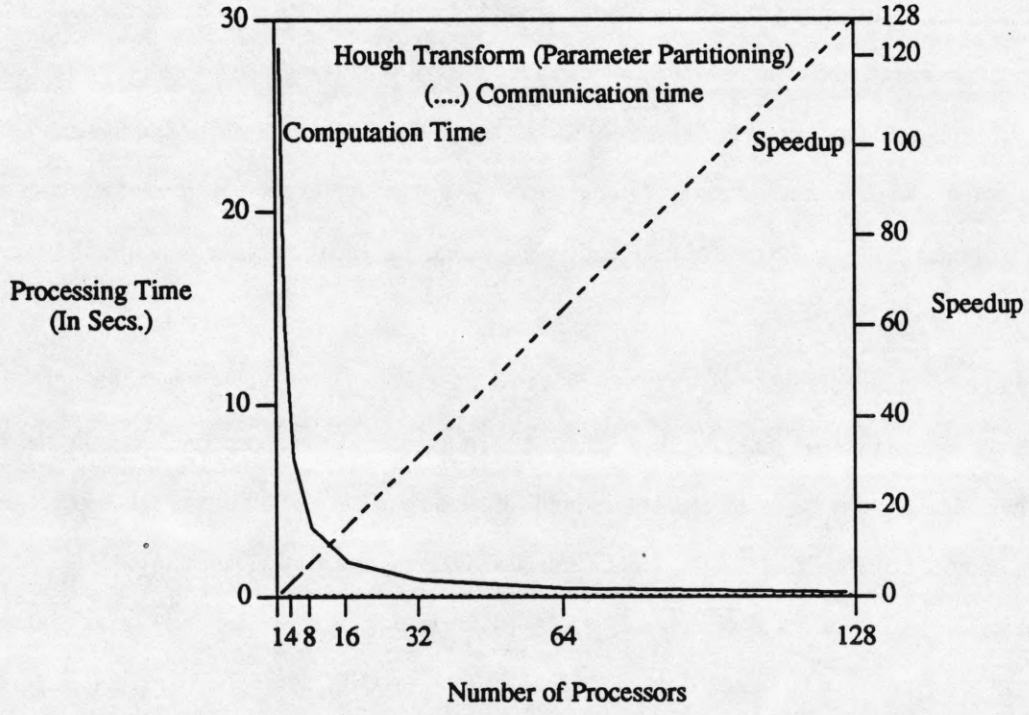


Figure 13 : Performance of Hough Transform (Parameter Partitioning)

Another way in which the parameter partitioning mapping can be performed is as follows. Instead of storing the image in all the processors, a controller processor, such as a DSP can store the image and broadcast each significant pixel value and its location while processors compute the votes in an SIMD lock-step fashion. This results in saving the memory, because now only one processor need store the image. The communication requirement for this mapping is $f \times N^2$, where f is the fraction of significant pixels. However, the communication can be overlapped with computation because while processors are computing the vote count for a location in the image, the next location can be broadcast. Therefore, the time to compute the Accumulator_array in this case will be MAX(t_{cp} , Broadcast time for $f \times N^2$ pixels locations).

By using parameter partitioning, the overhead of combining partial results is eliminated, and for each processor the communication is reduced to exchanging one row of the accumulator array with two other processors. Therefore, the communication remains constant as the number of processors increases. Figure 13 shows the speedup, computation time and communication time for hough transform using parameter partitioning. Figure 14 compares the communication overhead and the speedup for the two types of partitioning. Notice that using parameter partitioning it is possible to obtain almost linear speedup.

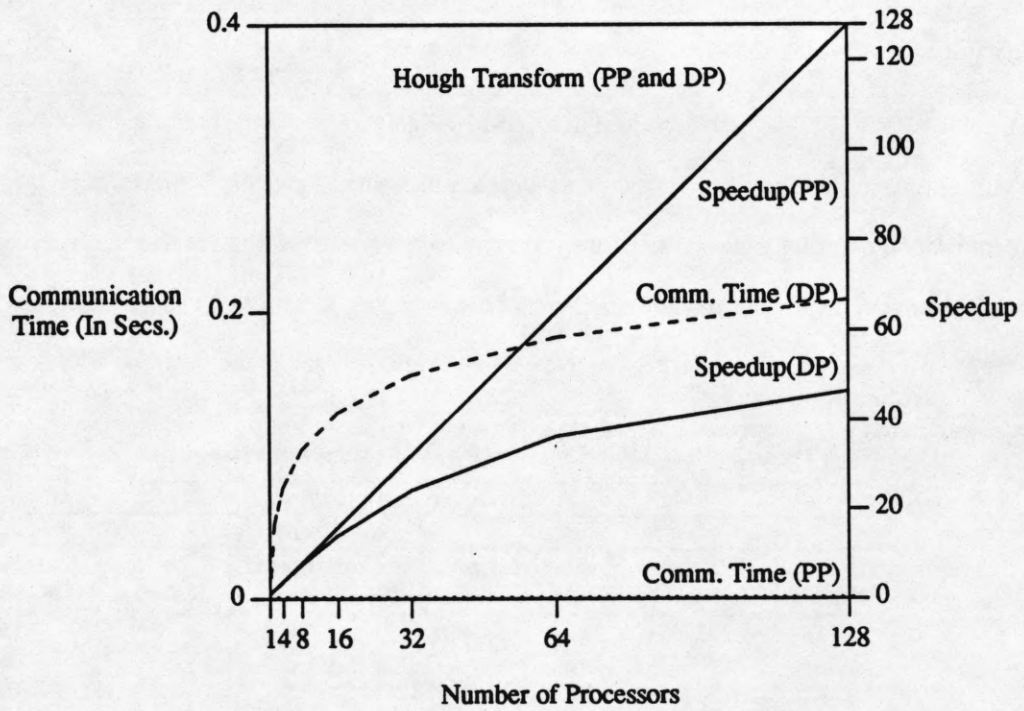


Figure 14 : Comparison of Performance of Parameter Partitioning and Data Partitioning for Hough Transform

3. Parallel Implementation Results

This section contains implementation of some algorithms on a simulated processor cluster. A cluster was simulated on an intel iPSC/2 hypercube multiprocessor. The performance results capture all the overheads associated with parallel programming, and therefore, the results are very accurate. Also, we show through the example of 2-D FFT algorithm that the analysis presented in the previous section is very close to the implementation results. We present performance results for four algorithms in this section. Two algorithms are 2-D FFT and separable convolution. The other two algorithms are parts of the Image Understanding Benchmark Algorithms developed by Weems et al [2]. The two algorithms are sobel edge detection and median filtering. The performance of the algorithms has been evaluated using the test data provided with the benchmark algorithms [2].

Table 2 shows the performance for separable convolution implementation on a 256×256 image with window size 10×10. The table shows the major computation operations in the algorithm which include floating point operations as well as integer operations. The fifth column shows the number of times connection in the crossbar needs to be changed during the algorithm execution, and column 6 contains the rounded value of the amount of data communicated in KBytes. The table shows that the communication time is very small compared to the computation time, and therefore, good speedups are obtained.

3.1. 2-D FFT

A mapping of 2-D FFT has been described in section 2. Figure 15 shows the performance of 2-D FFT on a 16 processor cluster (image size 256x256). Other parameters are the same as given in Table 1. Solid lines in the graph show the computation times for analysis (symbol +) and implementation. We observe that the analytical results are very accurate. However, the implementation times are a little more than that given by analysis because implementation captures the overhead of index management, etc., which is not included in the analysis. The graph also shows

Table 2 : Separable Convolution Implementation Results

	Separable Convolution					
	Window 10x10					
No. Proc.	Fl. Point K. Ops	Other K. Ops	Comp. Time (ms.)	Comm. Setup	Comm. K Bytes	Comm. Time(ms.)
1	3932	3932	2607	0	0	0
2	1966	1966	1310	2	20	4.09
4	983	983	658	3	20	4.09
8	492	492	332	3	20	4.09
16	246	246	169	3	20	4.09

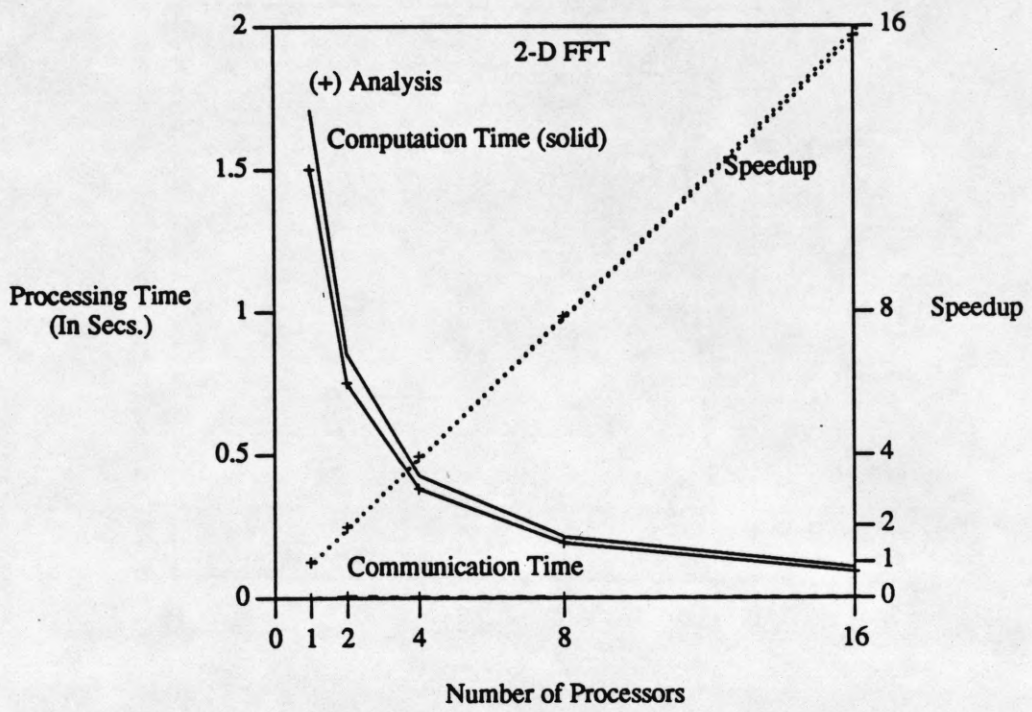


Figure 15 : Performance of 2-D FFT on a cluster (Analysis and Implementation)

the corresponding speedups for both cases. Note that speedups obtained through analysis and implementation are almost the same and are practically indistinguishable. Figure 16 shows graphs for the communication time. Again, implementation and analytical results are very close to each other.

3.2. Separable Convolution

3.3. Benchmark Algorithms

The Image Understanding Benchmark provided the serial version of the programs and the data [2]. We implemented sobel edge detection and median filtering algorithms.

3.3.1. Sobel

Sobel edge detection is a two-dimensional convolution operation with a 3×3 mask. The implementation used medium grain parallelism in an MIMD mode, and mapping was similar to that of separable convolution. Table 3 illustrates the performance results for sobel edge detection algorithm. There were six data sets but here we present results using only one data set (test, size 256×256). The results obtained on other data sets were similar. The table

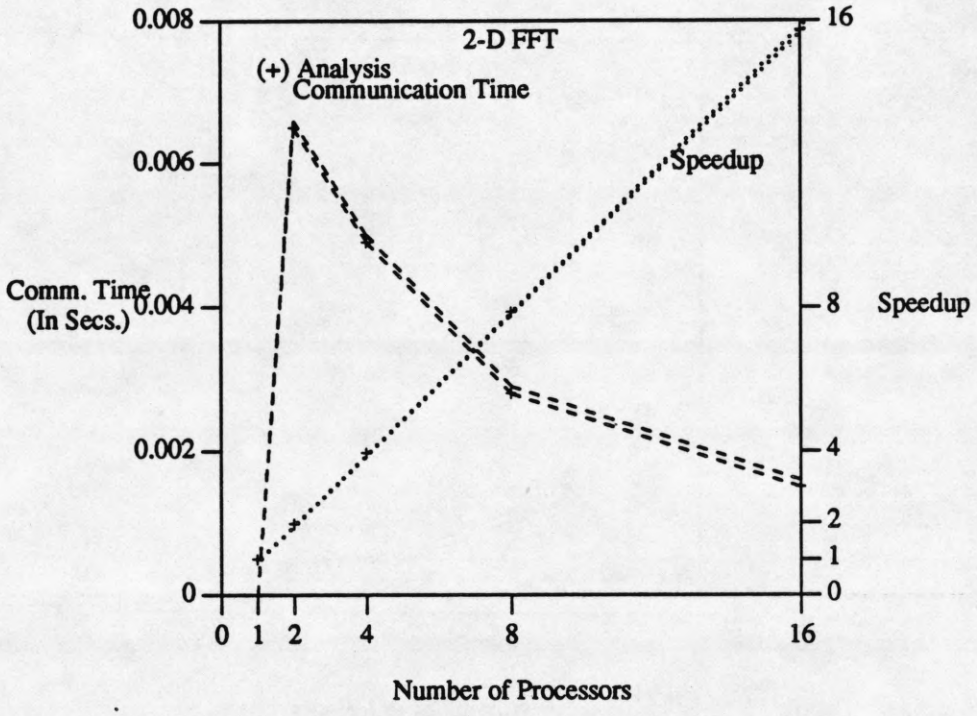


Figure 16 : Communication Time for 2-D FFT

includes all overheads, including program load time, data load time, data input time (from global memory), and time to gather results. If all the overhead is included, then the performance for larger cluster size is sublinear. There are two main reasons for this performance. First, amount of computation per pixel is very small (3×3 convolution), and second, all the overhead is included in the computation of the speedup. The parameters for communication bandwidth are conservative (20 MBytes/sec.), and if the bandwidth is assumed to be larger, then the performance is expected to be much better.

Table 3 : Sobel Edge Detection

Sobel (Test)							
No. Proc.	**Proc. Time(sec.)**	**Data load Time(Sec.)**	**Result Output Time(sec.)**	**Prog. Load Time(sec.)**	**Data Input Time(sec.)**	**Total Time(sec.)**	**Speed up**
1	4.04	0	0	0	0.008	4.05	1
2	2.02	0.056	0.014	0.001	0.008	2.1	1.92
4	1.01	0.056	0.014	0.001	0.008	1.09	3.70
8	0.51	0.056	0.014	0.001	0.008	0.589	6.91
16	0.26	0.056	0.014	0.001	0.008	0.33	12.13
32	0.13	0.056	0.014	0.001	0.008	0.21	19.71

3.3.2. Median Filtering

Table 4.4 shows the performance results for the median filtering algorithm. The algorithm was evaluated on the same data set. Size of the median filter was 5×5. Data is partitioned along the rows. Each processor is allocated an equal number of rows and two boundary rows in each direction. There is no need for communication during the algorithm execution. Median filtering does not involve any floating point multiplication or addition operations (only comparison operations are needed). Table 4.4 shows that we can obtain good speedups on a cluster for median filtering.

4. Performance of Parallel Algorithms on Multiple Clusters

The extent of inter-cluster communication depends on the type of algorithms, how they are mapped in parallel, frequency of communication and amount of data to be communicated. As discussed in the first paper [1], these requirements vary for algorithms belonging to different classes.

We are mainly interested in the performance evaluation of parallel algorithms when mapped across clusters. The performance of an algorithm will be affected by interference from other processors in the system which are not executing the particular algorithm under study.

Consider a parallel execution of an algorithm across clusters. Suppose the computation time is t_{cp}, intra-cluster communication time is t_{cl}, inter-cluster communication time is t_{icl}, and the execution time when the algorithm is executed on a single processor is t_{seq}. Then the speed up in the best case is given by

$$Sp = \frac{t_{seq}}{t_{cp} + t_{cl} + t_{icl}} \tag{1}$$

That is, assuming there is no interference while accessing the network or the global memory. Under the conditions in which there are conflicts while accessing the network, the inter-cluster communication time will be given by

Table 4 : Median Filtering

Median Filtering (Test)							
No. Proc.	Proc. Time(sec.)	Data load Time(Sec.)	Result Output Time(sec.)	Prog. Load Time(sec.)	Data Input Time(sec.)	Total Time(sec.)	Speed up
1	60.36	0	0	0	0.008	60.37	1
2	30.17	0.056	0.056	0.001	0.008	30.30	1.99
4	15.19	0.056	0.056	0.001	0.008	15.31	3.94
8	7.72	0.056	0.056	0.001	0.008	7.85	7.70
16	3.99	0.056	0.056	0.001	0.008	4.11	14.68
32	1.90	0.056	0.056	0.001	0.008	2.02	29.93

$w \times t_{icl}$, and therefore, the speed up will be given by

$$Sp' = \frac{t_{seq}}{t_{cp} + t_{cl} + w \times t_{icl}}. \tag{2}$$

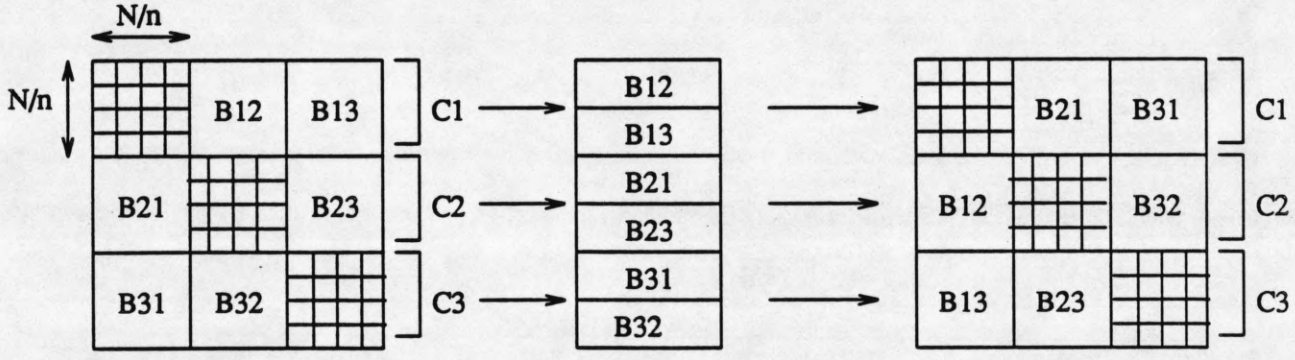
Hence, degradation in speed up with respect to the best case speed up will be

$$\frac{Sp - Sp'}{Sp} = \frac{(w-1) \times t_{icl}}{t_{cp} + t_{cl} + w \times t_{icl}}. \tag{3}$$

This section discusses the performance of various algorithms when mapped across clusters. The algorithms are selected according to their communication requirements. We have chosen one algorithm from each of the following categories; Local Varying, Global Fixed and Global Varying. Algorithms in each of these categories exhibit different communication characteristics, and therefore, the analysis will provide the performance of the architecture for a wide range of algorithms.

4.1. Two-Dimensional Fast Fourier Transform (2-D FFT)

From Section 2 we know that a 2-D FFT can be performed in two steps : a one-dimensional N point FFT along the rows followed by a one-dimensional N point FFT along the columns, or vice versa. We use this property to map the algorithm across clusters. Hence, dividing the data along rows will not require communication when computing one-dimensional FFT. However, communication is needed to obtain transpose of the intermediate results. Figure 17 shows an example of the two steps and communication for three clusters.



(a) Row FFT (b) Transpose using global Memory (c) Column FFT

The shaded area denotes data which remains within the cluster

Figure 5 : An Example of Mapping 2-D FFT on Three Clusters

Clusters are allocated rows in proportion to their size. A cluster C_i of size $P_c(i)$ (i.e., containing $P_c(i)$ processors) is allocated $\frac{N \times P_c(i)}{\sum_{i=1}^{i=n} P_c(i)}$ rows, where n is the total number of clusters executing the algorithm. Within a cluster rows are equally divided among processors. In the first phase processors compute N point FFT of all the rows in their granule. In the second phase, to obtain transpose of the intermediate data, processors write the intermediate results into the designated global memory locations, which is read by other processors. Data remaining within a cluster is transposed using the cluster crossbar.

The computation time in terms of number of instructions is given by the following. The total number of processors are given by P, and we assume all clusters have the same size (P_c).

$$t_{cp} = \frac{12 \times N^2 \log_2(N) \times t_{fl}}{P} \tag{4}$$

where t_{fl} is the number of instruction per floating point operation. The intra-cluster communication time (t_{cl}) and the inter-cluster communication time (t_{icl}) are given by

$$t_{cl} = \frac{2 \times N^2 (P_c - 1)}{P^2} \tag{5}$$

$$t_{icl} = \frac{4 \times N^2 \times (n-1) \times P_p \times R}{n \times p} \tag{6}$$

where P_p is the number of processors per port and R is the communication speed of the network in terms of number of instructions/per word transfer.

Using these parameters for 2-D FFT traffic intensity, computation times, and the parameters from Table 1, we evaluate the performance using the analysis presented earlier. Figure 18 shows the speedup obtained for the 2-D FFT algorithm. The X-axis shows the number of processors (cluster size is 16). For example, value 48 means that the algorithm is executed on 3 clusters, each containing 16 processors. The four different graphs in the Figure show speedups for no conflict (best case), low conflict, medium conflict and high conflict cases through the global interconnection network (multi-stage interconnection). We will present similar results when bus is used as the global interconnection network later in this section. We observe that speedup obtained under varying degrees of conflicts through the network is comparable to that obtained in the best case. However, the best case speedup itself is not linear because of the delays through the network and the global memory.

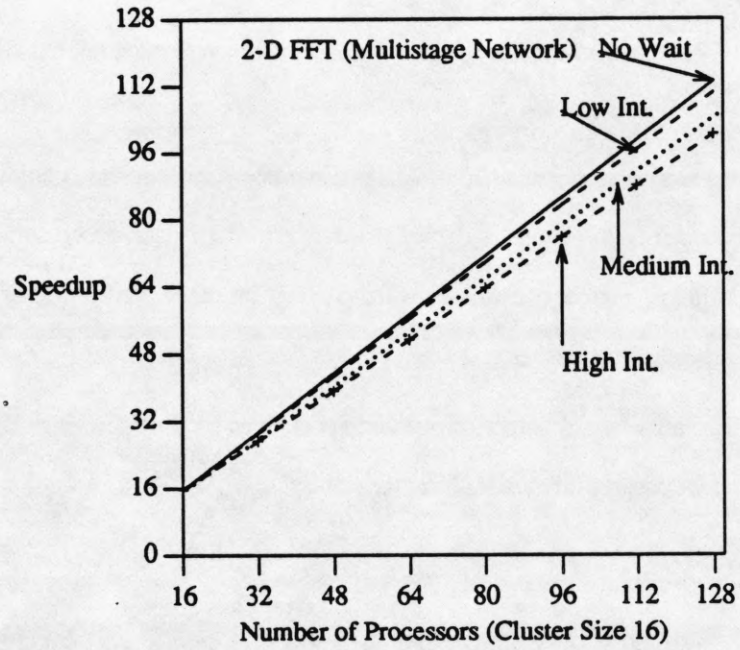


Figure 18 : Speedup for 2-D FFT (Multistage Network)

Figure 19 shows the computation and communication time for 2-D FFT as a function of number of processors. Figure 20 shows a blown-up graph for the communication times. The communication time is much smaller than the computation time. Furthermore, the communication time also decreases as the number of processors (clusters) increases. Also note that the intra-cluster communication time is much smaller than the inter-cluster communication time. Figure 21 shows percentage degradation in speedup, as defined in Equation (3), for different levels of conflict in the network. The degradation in the speedup levels off after increasing initially because the communication time decreases as the number of processors increases.

Figure 22 shows the sensitivity of the speedup to the network bandwidth. The network bandwidth is normalized to computation speed. For example, value 1 on the X-axis means that it takes the same amount of time (amortized or in block transfer mode) to write/read a word to/from global memory as it takes to execute one instruction. The region on the left of 1 indicates faster communication network, and to the right of 1 indicates slower communication network. It is evident from the Figure that degradation in speedup occurs very fast as the communication becomes slower. Therefore, in order to obtain any significant speedups from parallel computation, it is important to have matched computation and communication speeds; otherwise, increasing the number of processors or the processor speeds will not improve the performance as expected. Figure 22 also illustrates that the four graphs diverge as the communication becomes slower meaning that slower performance under heavy traffic suffers more in the

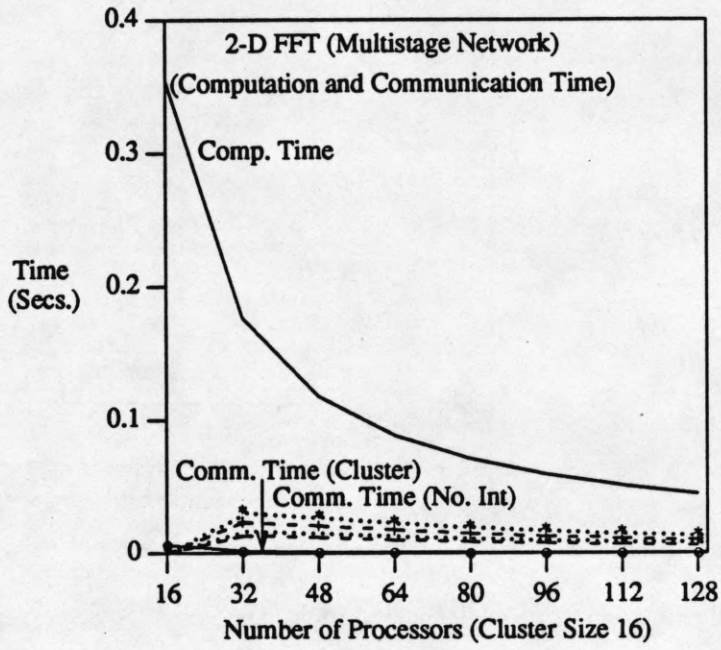


Figure 19 : Computation and Communication Times for 2-D FFT (Multistage Network)

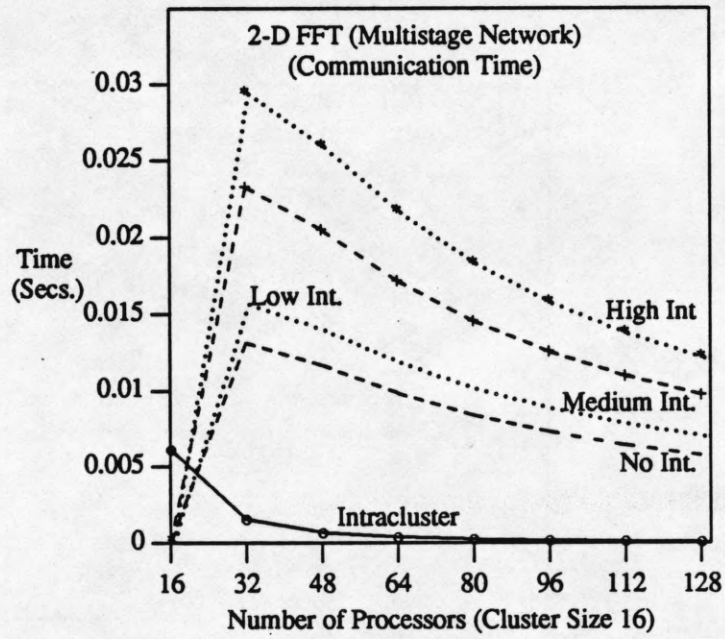


Figure 20 : Communication Times for 2-D FFT (Multistage Network)

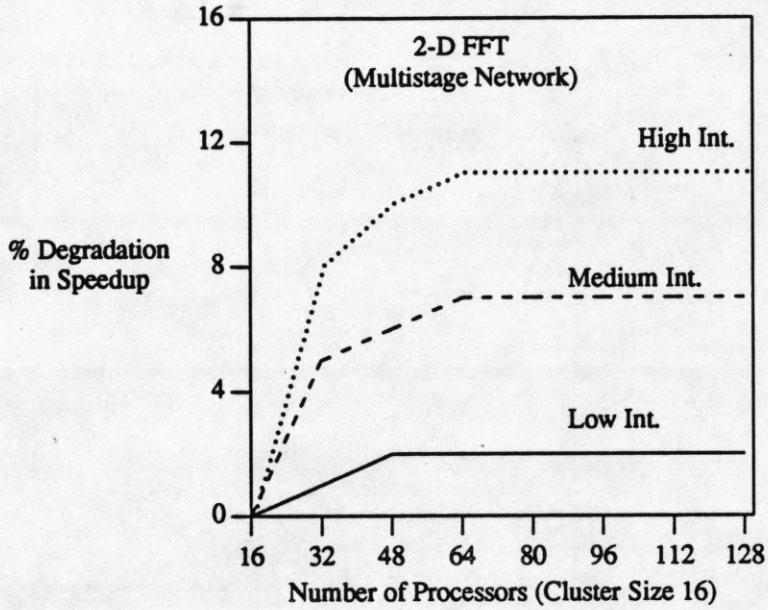


Figure 21 : Degradation in Speedup Due to Conflicts for 2-D FFT (Multistage Network)

slower network than under light traffic.

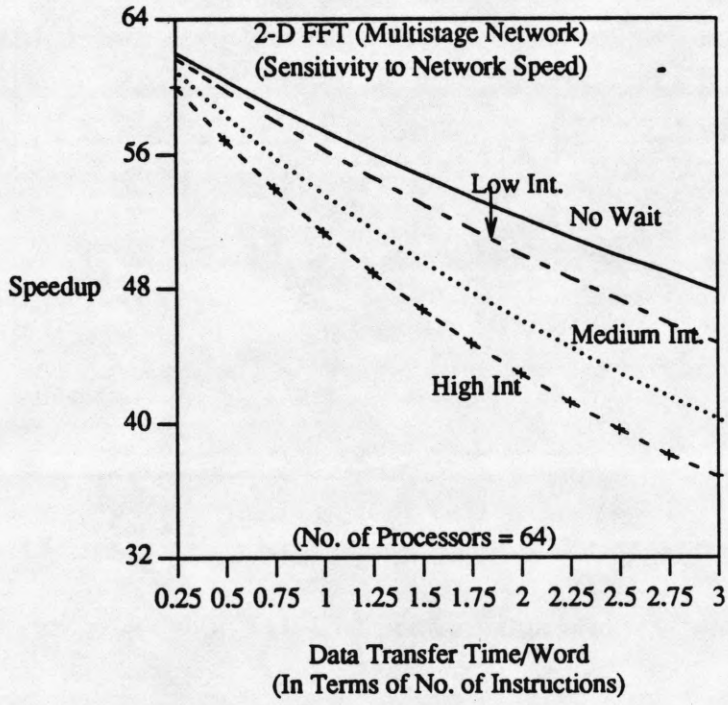


Figure 22 : Speedup vs. Network Speed

The following is a discussion of the performance of 2D-FFT when a bus is used as a global interconnection network. The algorithm is mapped as described above. Since global bus can be accessed by only one processors at a time, the inter-cluster communication time becomes additive as the number of clusters is increased. Therefore, the performance is expected to be worse than that in the case of the multistage interconnection network. The total computation time remains the same as in the previous case and is given by

$$t_{cp} = \frac{12 \times N^2 \log_2(N) \times t_{fl}}{P}. \tag{7}$$

However, the inter-cluster communication time becomes

$$t_{icl} = \sum_{i=1}^{i=n-1} \frac{2 \times R \times N^2}{n^2} = \frac{2 \times R \times (n-1) \times N^2}{n^2}. \tag{8}$$

In other words, each cluster needs to send $\frac{(n-1)}{n}$ fraction of its data to transpose the intermediate results. This is achieved by a designated processor in each cluster, which collects the data and broadcasts it on the bus to be read by other cluster processors. Hence, there is an additional overhead of collecting and distributing the intermediate data. The intra-cluster communication time in this case is given by

$$t_{cl} = t_{cl1} + t_{cl2} + t_{cl3}$$

where,

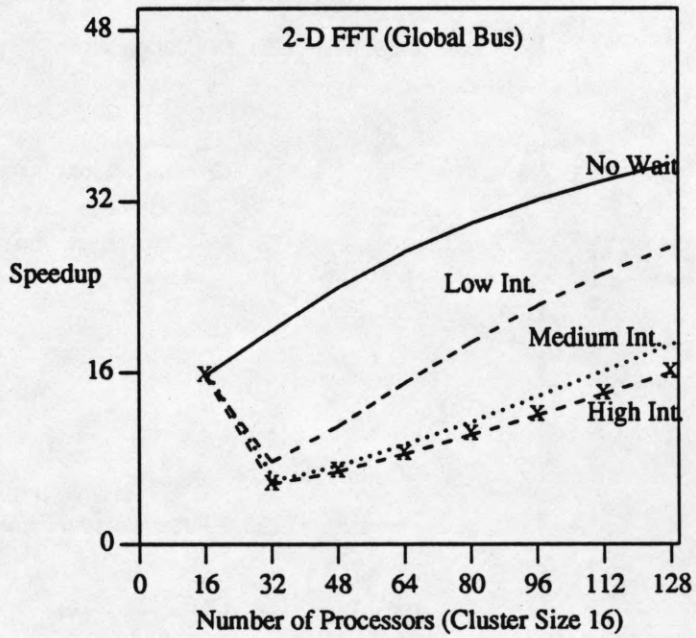


Figure 23 : Speedup for 2-D FFT (Global Bus)

$t_{cl1} = \frac{2\times(P_c-1)\times N^2}{P_c^2\times n}$, for within cluster transpose, and

$t_{cl2} = t_{cl3} = \frac{2\times N^2\times(n-1)}{n^2}$, for sending, receiving and redistributing the intermediate data.

Using these parameters, we evaluate the performance of 2-D FFT under varying degrees of conflicts on the bus. Figure 23 shows the speedup for 2-D FFT as a function of the number of processors (cluster size 16). When there is no conflict on the bus, the speedup increases with the number of processors. However, under conflicts, the speedup first decreases and then increases slowly. In fact, for medium and high conflicts, the speedup obtained on one cluster is better than that obtained using multiple clusters. The reason for such poor performance is that even though the communication is decomposable in 2-D FFT, the inter-cluster communication time becomes additive due to the bus and increases as the number of clusters executing the algorithm increases as shown in Figure 24.

Figure 25 shows the relative performance degradation in the speedup. The degradation is very significant. However, the degradation itself decreases as the number of processors (clusters) increases because more clusters execute the algorithm, and consequently, less number of clusters interfere. Figure 26 shows the sensitivity of the speedup to the bus speed. Again, the Figure shows that performance degrades rapidly as the bus becomes slower. In order for a bus to be viable global interconnection network it is essential that the bus bandwidth be much greater

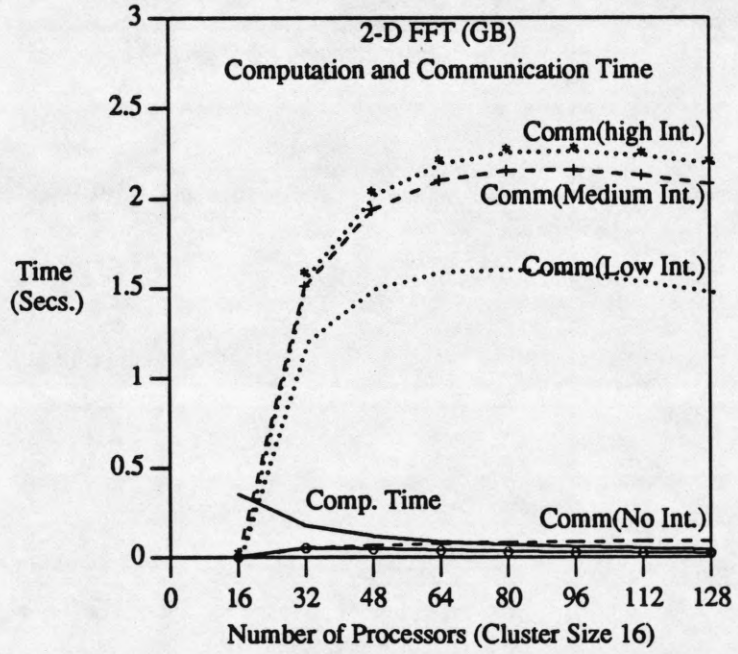


Figure 24 : Computation and Communication Times for 2-D FFT (Global Bus)

than the processor speed.

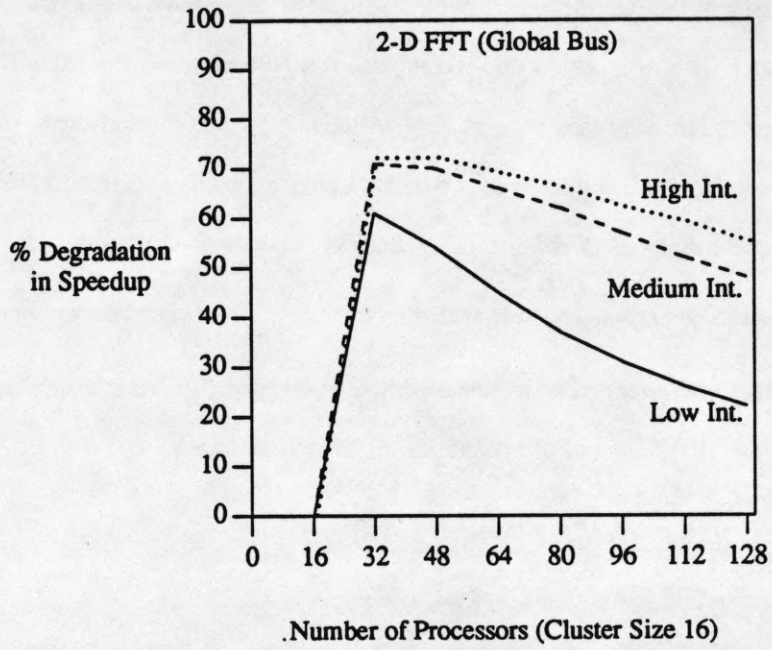


Figure 25 : Degradation in Speedup Due to Conflicts 2-D FFT (Global Bus)

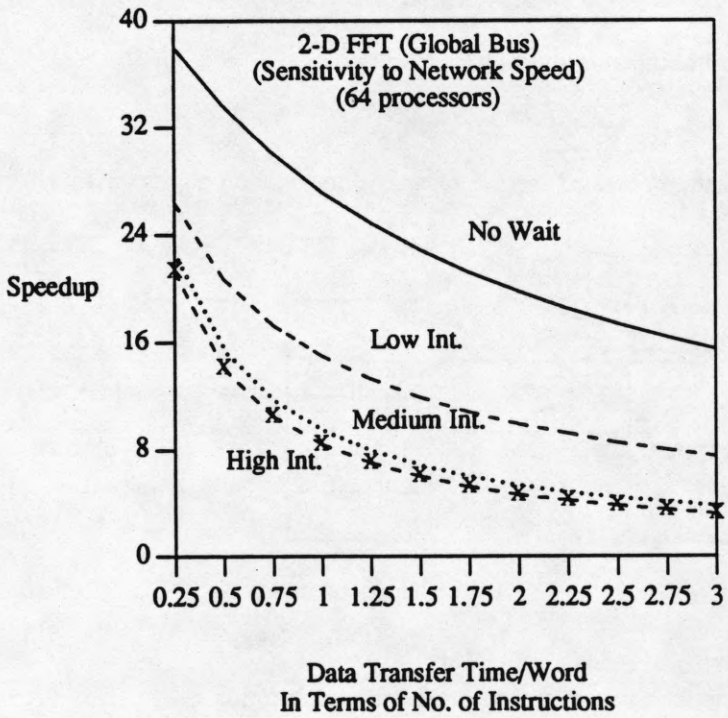


Figure 26 : Speedup vs. Network Speed (Global Bus)

4.2. Separable Convolution

This algorithm consists of two steps. First convolution along rows using two one-dimensional masks and then convolution along columns of the intermediate results. Partitioning along rows in clusters, therefore, avoids communication in the first step. However, before the second step can be performed, boundary rows with each cluster need to be communicated to other clusters. Figure 27 shows the mapping on three clusters. Note that unlike in 2-D FFT, a cluster needs to communicate with at most two other clusters to obtain the upper and lower boundary rows of the intermediate results. The number of rows to be exchanged depends on the kernel size. For a kernel size of $w \times w$, the number of rows to be exchanged along each direction is $\frac{w}{2}$. The amount of communication is fixed and is independent of the number of clusters on which the algorithm is mapped. The same mapping will work for regular 2-D convolution except that the amount of computation per pixel will be larger.

The computation time for the two steps is given by

$$t_{cp1} = t_{cp2} = \frac{2 \times t_{fl} \times N \times (\frac{w}{2}+1)}{P} \tag{9}$$

the intra-cluster communication is given by

$$t_{cl} = 2 \times N \times w \tag{10}$$

and, the inter-cluster communication is given by

$$t_{icl} = 2 \times w \times N \times R \tag{11}$$

Figure 28 depicts the speedup obtained for the separable convolution algorithm as a function of the number of

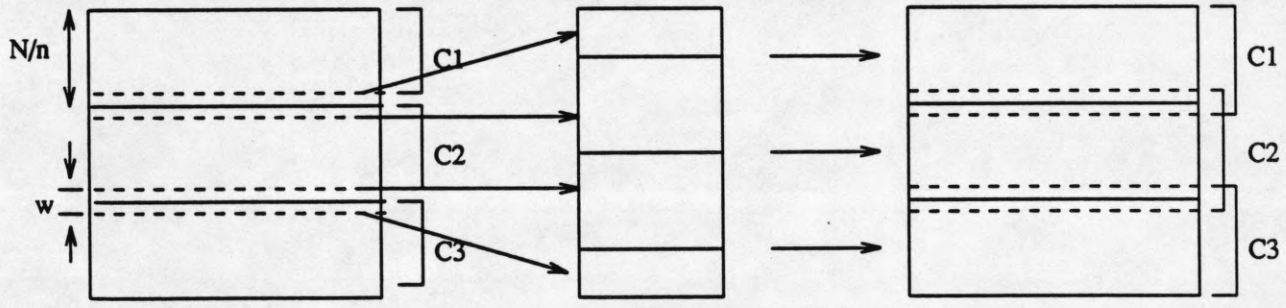


Figure 27 : An Example of Mapping Separable Convolution on Three Clusters

clusters (cluster size = 16). The speedup increases sublinearly as the number of clusters increases. The reason for not obtaining better speedup is that the computation per point of the input is small, and the computation per processor decreases as the number of clusters increases, but the communication remains constant (as long as the granularity per processor is at least $\frac{w}{2}$ rows). Hence, the ratio of computation and communication decreases as the number of processors increases. The computation and communication times are shown in Figures 29 and 30. Figure 29 compares the two times whereas Figure 30 shows only the communication time.

Note that inter-cluster communication can be avoided completely if clusters are assigned overlapped rows to perform the first step. That is, if a cluster is responsible to compute the convolution for R_i rows, then its is assigned $w + R_i$ rows. Therefore, each cluster has to perform additional computation to obtain 1-D convolution of w additional rows. If the extra computation time is less than the communication time then overlapped data partitioning is better.

Figure 31 shows a performance comparison of the two partitioning methods. When the number of processors executing an algorithm is small, the performance is almost the same. For smaller window sizes the difference is marginal and becomes apparent only when the number of processors becomes large. However, as the window size increases (40x40 in Figure 31), the performance with overlapped computation becomes poor because the overhead

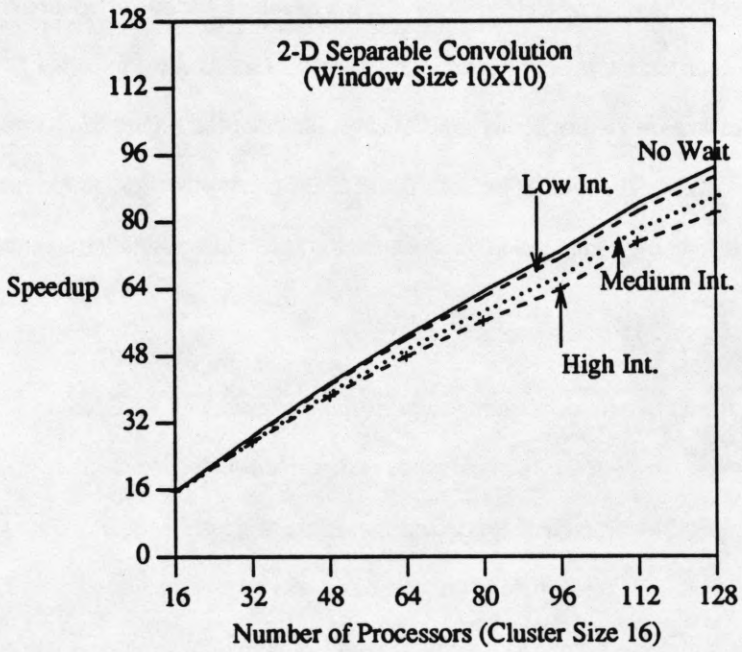
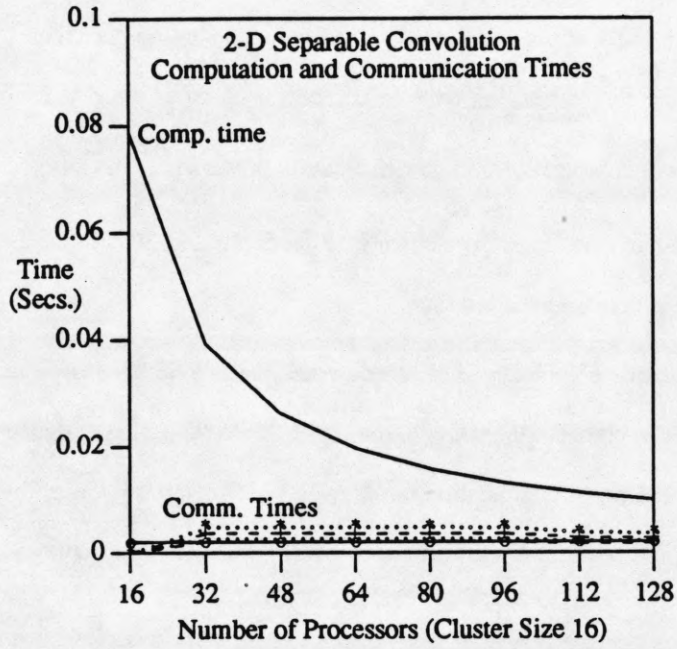


Figure 28 : Speedup for Separable Convolution (Multistage Network)



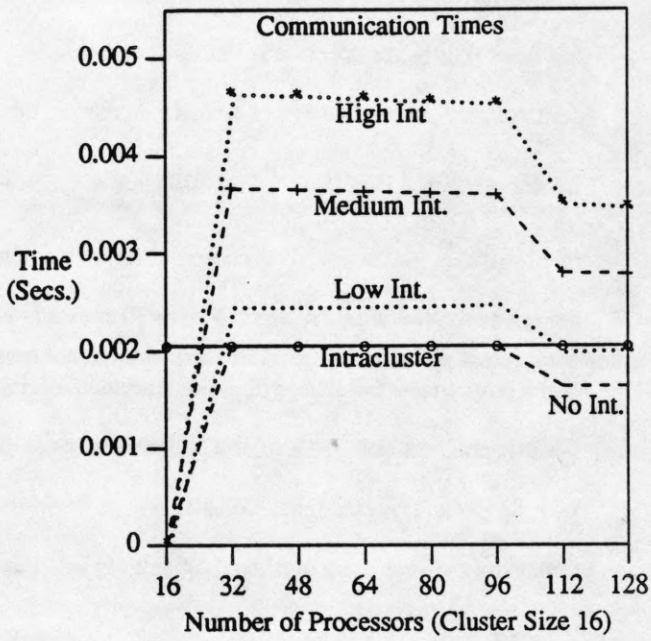


Figure 29 : Computation and Communication Times Separable Convolution (Multistage Network)

Figure 30 : Communication Times

of extra computation becomes larger than the communication overhead.

Figure 32 shows the performance of the algorithm when the bus is used as a global interconnection network. The speedup increases as the number of clusters increases but eventually levels off. Though inter-cluster communication time per cluster is constant, total communication time increases as the number of clusters increases, because only one cluster can send data on the bus at any time. This is illustrated in Figure 33 where the communication time (with no interference) is a linear function of the number of clusters. Another reason for speedup to level off is that for a larger number of clusters the computation time becomes comparable or smaller than the communication time.

4.3. Hough Transform

We have evaluated two mappings for hough transform, namely, Data Partitioning (DP) and Parameter Partitioning (PP). The difference between the two mappings is described in section 2. Briefly, in DP the data is decomposed among clusters and in PP parameters are decomposed across clusters. In DP, Data is allocated to clusters in proportion to their size. Within a cluster data is distributed equally among the processors. The algorithm consists of three phases. In the first phase, each processor computes and accumulates the count contributed by its data for all the parameter values. Note that each processor maintains the entire accumulator array. In the second phase, partial

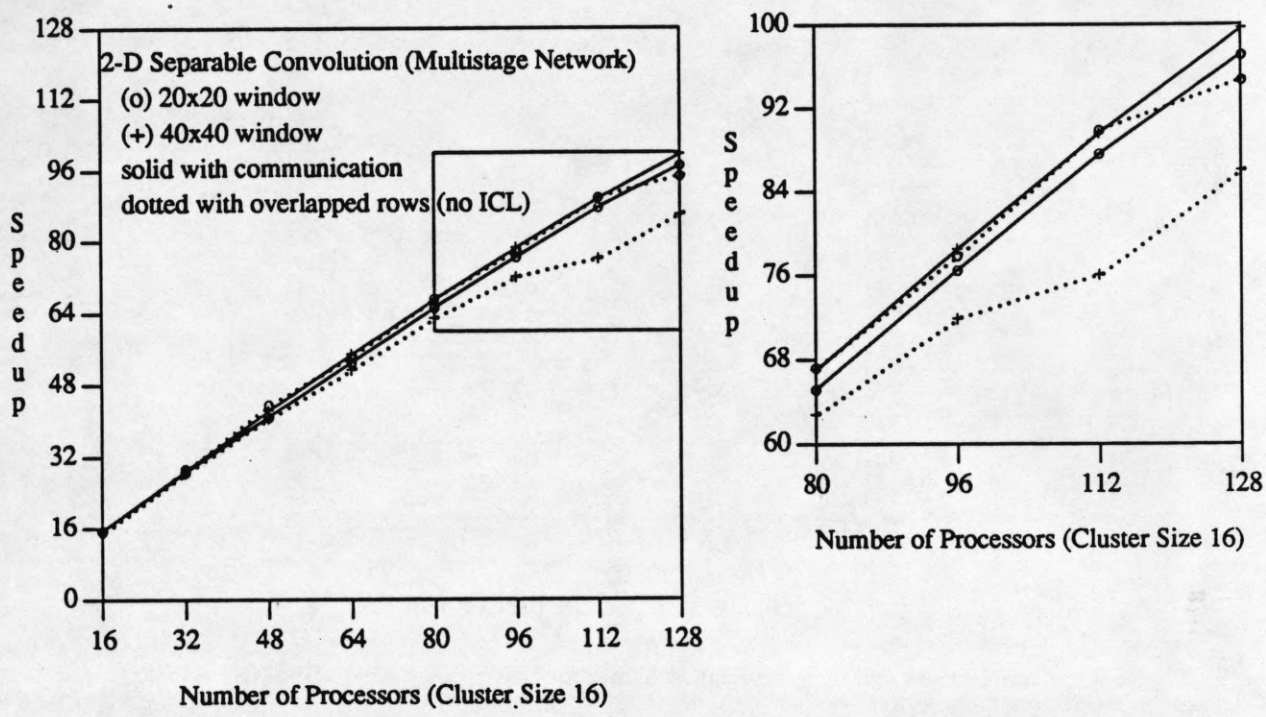


Figure 31 : Comparison of Performance between Overlapped Computation and Communication for Separable Convolution
(The box on the left graph has been blown up in the right graph)

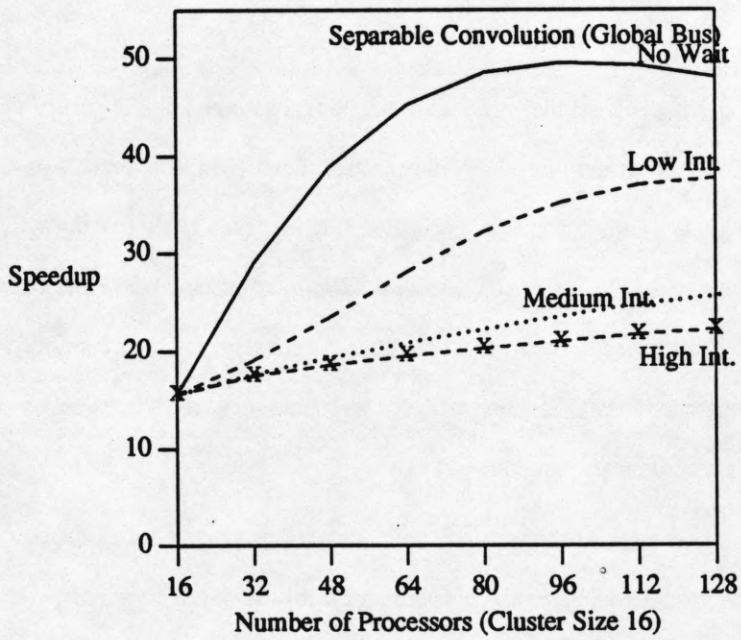


Figure 32 : Speedup for Separable Convolution (Global Bus)

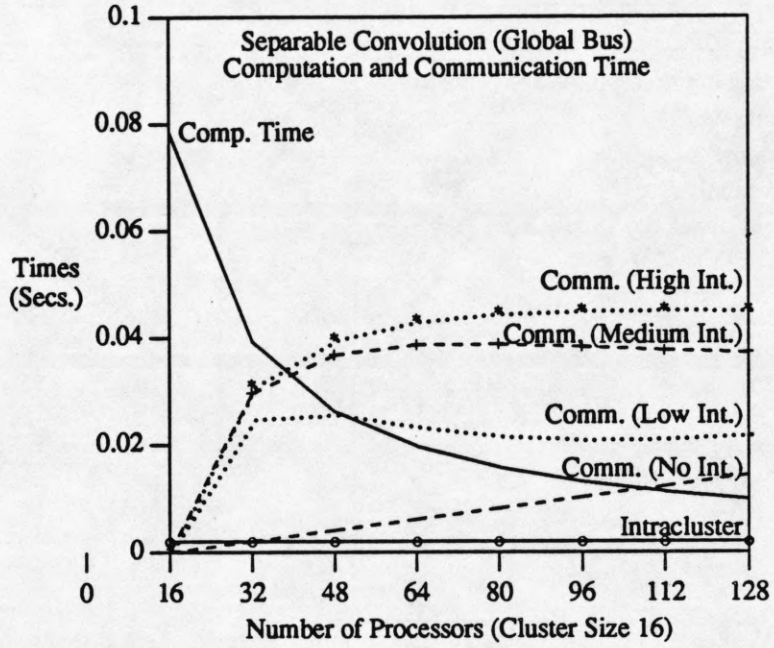


Figure 33 : Computation and Communication Times for Separable Convolution (Global Bus)

results are combined within a cluster, i.e., all the accumulator arrays are added together, and then a designated processor from each cluster writes the accumulator array to designated memory locations. Arrays from all the clusters participating in the algorithm execution are then collected by one cluster. In the third phase, the cluster having the entire accumulator array computes the local maxima.

Under PP, each cluster is assigned the entire input data but is assigned only a part of the parameter space. The parameter space is partitioned in proportion to the cluster size. Each cluster receives two more parameters (boundary values) so that inter-cluster communication is avoided. That is, each cluster performs a fixed amount of additional computation to avoid inter-cluster communication. Within a cluster, however, data is distributed equally among the processors, and all processors work on the entire allocated parameter space. Dividing the parameter space results in mutually exclusive accumulator arrays with processors, and therefore, to compute local maxima, there is no need for inter-cluster communication.

For DP, the computation and communication times for various phases are as follows: t_{cp1} is for computing accumulator count, t_{cp2} is for combining partial accumulator arrays within a cluster, t_{cp3} is for computing the final accumulator array, and t_{cp4} gives the time to compute the local maxima by one cluster.

$$t_{cp1} = \frac{3 \times t_{fl} \times N^2 \times \theta_c}{P} \tag{12}$$

$$t_{cp2} = \rho_c \times \theta_c \times \log_2 P_c \tag{13}$$

$$t_{cp3} = \frac{(n-1) \times \rho_c \times \theta_c}{P_c} \tag{14}$$

$$t_{cp4} = \frac{3 \times \rho_c \times \theta_c}{P_c} \tag{15}$$

Intra-cluster and inter-cluster communication times are give by

$$t_{cl} = (log_2 P_c + 1) \times \rho_c \times \theta_c \tag{16}$$

$$t_{icl} = \frac{n \times R \times P_p \times \theta_c \times \rho_c}{P_c} \tag{17}$$

Similarly, the corresponding computation and communication times for PP are given by

$$t_{cp1} = \frac{3 \times t_{fl} \times N^2 \times (\frac{\theta_c}{n} + 2)}{P_c} \tag{18}$$

$$t_{cp2} = \log_2 P_c \times \rho_c \times (\frac{\theta_c}{n} + 2) \tag{19}$$

$$t_{cp3} = \frac{3 \times \rho_c \times \theta_c}{n \times P_c} \tag{20}$$

$$t_{cl} = (log_2 P_c + 1) \times (\frac{\theta_c}{n} + 2) \times \rho_c \tag{21}$$

Figure 34 depicts the speedups for hough transform using the two partitioning methods. Due to the communication overhead through global memory, which increases linearly with the number of clusters, the speedup for DP levels off. Figure 35 shows the computation and communication times for hough transform, whereas Figure 36 shows the communication overhead for hough transform in detail. Data partitioning does not perform as well as parameter partitioning. However, degradation with respect to best case speedup in DP is small. As we can observe, good speedup can be obtained for a global data dependent algorithm like hough transform. Figure 35 and 36 illustrate the computation and communication times for the DP case.

Figure 37 shows the speedup for hough transform (DP), and Figure 38 depicts the communication and computation times, respectively when the bus is used as a global interconnection network. Note that performance of the hough transform under PP will be the same in both cases because there is no global communication.

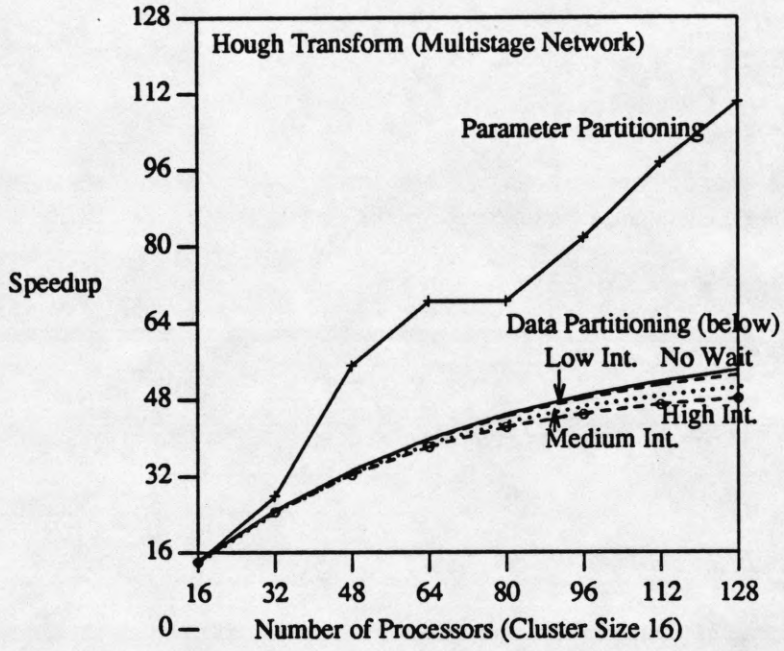


Figure 34 : Speedup for Hough Transform (Multistage Network)

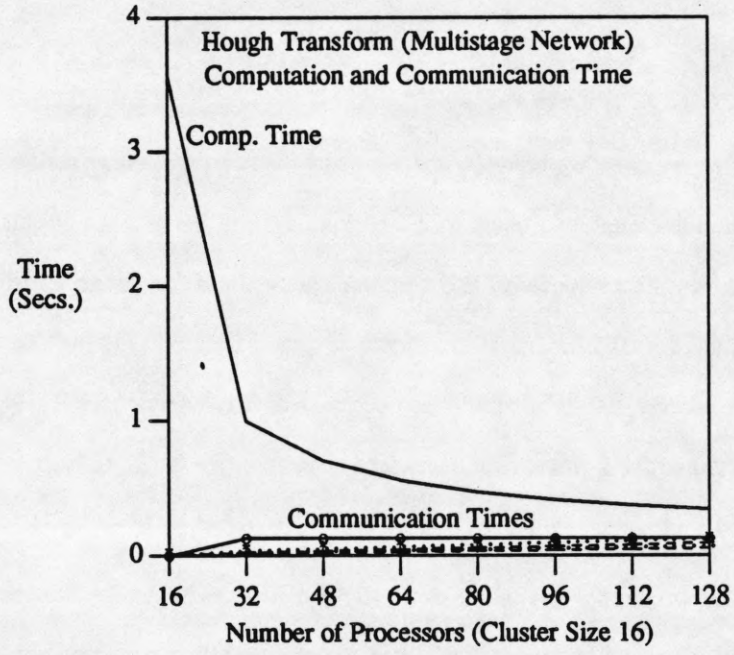


Figure 35 : Computation and Communication Times for Hough Transform (Multistage Network)

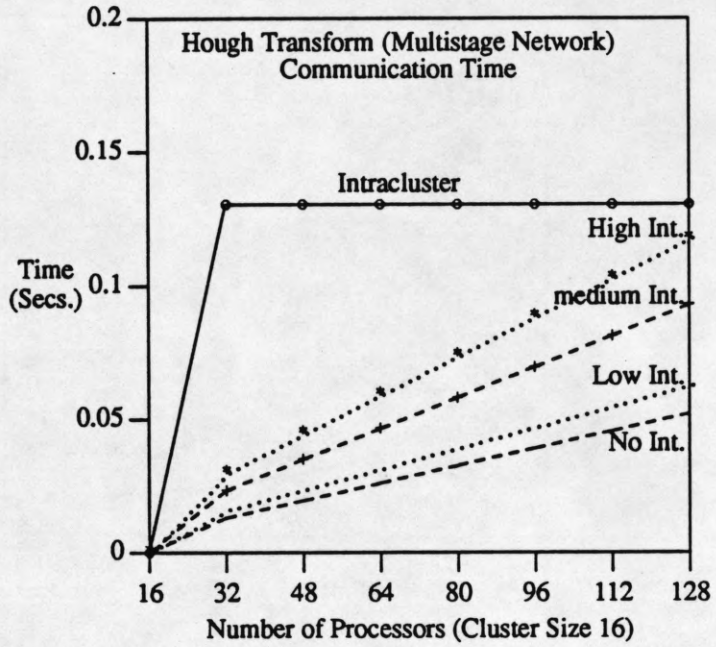


Figure 36 : Communication Times for Hough Transform (Multistage Network)

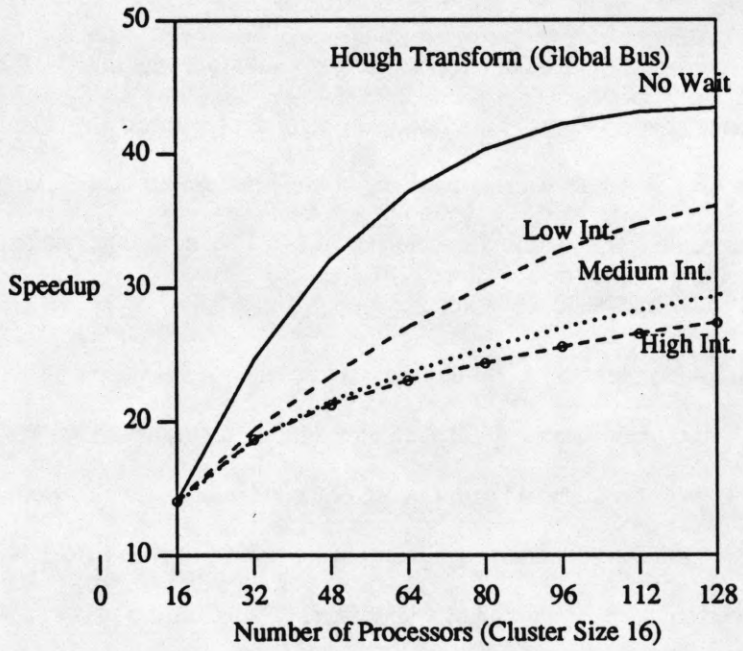


Figure 37 : Speedup for Hough Transform (Global Bus)

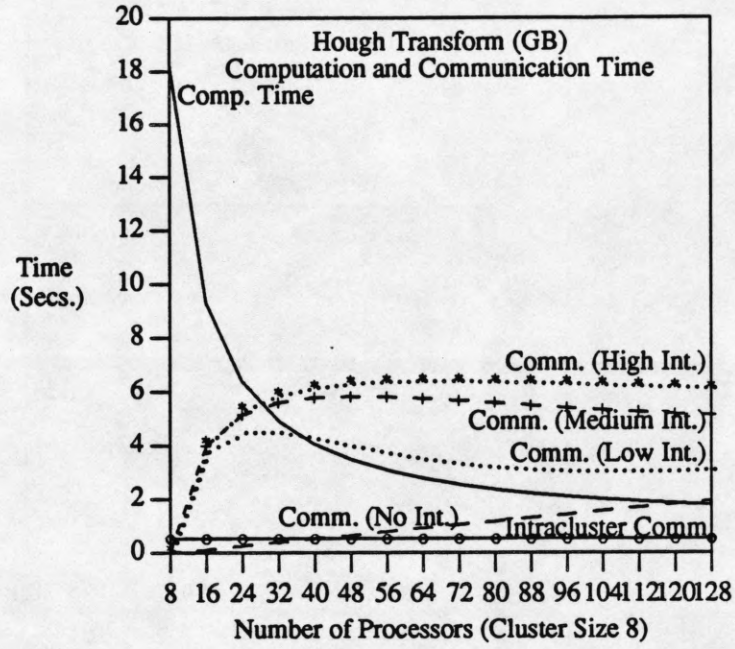


Figure 38 : Computation and Communication Times Hough Transform (Global Bus)

5. Summary

We presented performance evaluation of NETRA using several vision algorithms. The goal of this paper was to illustrate the performance of components of NETRA using algorithms with varying characteristics and communication requirements. For each algorithm we presented one or more mapping strategies, its performance evaluation, and a discussion of the results. The algorithms included 2-D FFT, convolution, separable convolution, hough transform, sobel edge detection and median filtering.

To evaluate parallel algorithms on a cluster, we explore alternative mapping strategies and computation modes. Some of the algorithms have been implemented on a simulated cluster and we show that the analysis provides very accurate results. We also discussed performance of the algorithms when they are mapped across multiple clusters. The results are used to compare alternative inter-cluster communication strategies and they show that it is possible to obtain good performance for algorithms with different characteristics under varying degrees of conflicts in global interconnection network.

In general, a multistage interconnection network as the global interconnection performs much better than a global bus, as expected. The parameters chosen for processor speed and communication speed were very conservative. We think that much faster processors and communication links are possible and available with current technology, and therefore, the performance results presented in this chapter are also conservative. However, we obtained

insight into the sensitivity of the performance measures as a function of various architecture parameters.

REFERENCES

[1] Alok Choudhary, Janak Patel, and Narendra Ahuja, "NETRA - A parallel architecture for integrated vision systems I: architecture and organization," *IEEE Transactions on Parallel and Distributed Processing (submitted)*, August 1989.

[2] C. Weems, A. Hanson, E. Riseman, and A. Rosenfeld, "An integrated image understanding benchmark: recognition of a 2 1/2 D mobile," in *International Conference on Computer Vision and Pattern Recognition*, Ann Arbor, MI, June 1988.

[3] Alok N. Choudhary, "Parallel architectures and parallel algorithms for integrated vision systems," in *Ph.D. Thesis*, University of Illinois, Urbana-Champaign, Agust 1989.

[4] E. Horowitz and S. Sahni, *Fundamentals of computer algorithms*. Computer Science Press, 1984.

[5] D. H. Ballard and C. M. Brown, *Computer vision*. Prentice-Hall, 1982.