International Journal of Information, Control and Computer Sciences
ISSN: 2517-9942
Vol:6, No:1, 2012

Processor Scheduling on Parallel Computers

Mohammad S. Laghari and Gulzar A. Khuwaja

***Abstract*—**Many problems in computer vision and image processing present potential for parallel implementations through one of the three major paradigms of geometric parallelism, algorithmic parallelism and processor farming. Static process scheduling techniques are used successfully to exploit geometric and algorithmic parallelism, while dynamic process scheduling is better suited to dealing with the independent processes inherent in the process farming paradigm. This paper considers the application of parallel or multi-computers to a class of problems exhibiting spatial data characteristic of the geometric paradigm. However, by using processor farming paradigm, a dynamic scheduling technique is developed to suit the MIMD structure of the multi-computers. A hybrid scheme of scheduling is also developed and compared with the other schemes. The specific problem chosen for the investigation is the Hough transform for line detection.

***Keywords*—**Hough transforms, parallel computer, parallel paradigms, scheduling.

I. Introduction

APPLICATIONS are decomposed into processes to exploit the parallelism inherent in an application. There are many ways to exhibit this parallelism, but the most commonly used parallel paradigms in scientific applications are: geometric parallelism, algorithmic parallelism and processor farming. Processor scheduling determines which and when processes are assigned to specific processors. There are different techniques of processor scheduling that can be used to optimize performance in parallel computer systems. Static process scheduling techniques are used successfully to exploit geometric and algorithmic parallelism, while dynamic process scheduling is better suited to dealing with the independent processes inherent in the process farming paradigm.

The work presented in this paper, investigates the performance of scheduling techniques for the parallel implementation of grid-type applications on a MIMD machine. The specific problem chosen for the investigation is the Hough transform for line detection. Several algorithms are developed for this application and are executed on a Networked Processor Linear Array *(NPLA)* consisting of 10 processors. Experiments are performed and compared in terms of total processing times, speedup and efficiency using varying number of processors.

M. S. Laghari is with the Electrical Engineering Department, Faculty of Engineering, United Arab Emirates University, P.O. Box: 17555, Al Ain, U.A.E. (phone: 00971-50-6625492; fax: 00971-3-7623156; e-mail: mslaghari@uaeu.ac.ae).

G. A. Khuwaja is with Department of Computer Engineering, College of Computer Sciences & Information Technology, King Faisal University, Al Ahsa 31982, Kingdom of Saudi Arabia. (e-mail: khuwaja@kfu.edu.sa).

Processor scheduling determines which processes are assigned to specific processors. There are many different techniques of processor scheduling that can be used to optimize performance in parallel computer systems [1].

Processors operating in such an environment will in general perform two functions; computations and communications. A suitable balance between these functions is required to ensure efficient use of the processing resources. When the time taken to perform the computation of a given sub-problem is less than the time associated with the communications of the data and results, then the communication bandwidth is likely to limit the performance. An appropriate scheduling technique can keep the processors as busy as possible and help achieve optimum performance.

The problem chosen for the investigation relates to the scheduling techniques for the parallel implementation of the Hough transform [2], [3].

The detection of straight lines in digital images is a recurring problem in computer vision and image processing. The Hough transform is an efficient method of detecting lines and curves. It is used to extract global features from shapes. The technique is robust in the presence of noise, missing points, and occlusions. Due to its computational complexity the Hough transform is not easily implemented for real-time applications. However, by using parallel paradigms, near real-time implementation of Hough transforms can be achieved on a network of multi-computers [4]-[7]. The Hough transform exhibits a regular structure of computation and may be considered best suited to static scheduling. However, by using dynamic and a hybrid (*statistic)* scheduling technique, the MIMD structure of NPLA is effectively exploited. A comparison of the investigated scheduling techniques is given in terms of total processing times, speedup and efficiency.

II. A Brief Review

Since their invention in the 1940's, computers based on the John von Neumann architecture have been built around one basic plan; a single processor, connected to a single store of memory, executing one instruction at a time. The processor fetches instructions from a program stored in the memory, then fetches operands for those instructions from the same memory, performs a calculation, and writes the results back to the memory.

The von Neumann architecture was popular for several reasons. It was conceptually simple, von Neumann machines were simpler to build and machines were economical. The idea of parallelism was originated at the same time by von Neumann to use many processors to solve a single problem.

The idea was to build a cellular automata machine in which a very large number of simple calculators work simultaneously on small parts of a large problem. However, it did not become reality because of the hardware and software capabilities of that time.

Things began to change in 1960's when the vacuum tubes were switched to solid state components. Instead of using one arithmetic logic unit, multiple functional units were incorporated in a machine resulting in the CDC 6600 computer, which is a state of art machine of that time operating at a clock speed of 100 nanoseconds.

In the early 1970's, the first vector computer called Cray 1 was developed. The multiple functional units of CDC 6600 were based on the idea of replication. This vector computer was based on overlapping of operations. In vector computers, the arithmetic-logic unit is divided into stages. If two long vectors of numbers are being added together, successive additions are overlapped to increase the overall throughput.

Development in the field of parallel processing continued for all these years. Then, in the late 1970's, four things made the parallel processing possible. The first was the development of the VLSI *(Very Large Scale Integration)* technology, which allowed hundreds of thousands of transistors to be integrated on a single integrated circuit. The second was in the development of better concurrent programming methods. The third was the actual construction of parallel computers. Example is the C.mmp (computer with multiple mini-processors) from Carnegie-Mellon University. Lastly, the continued development in the field of high speed vector computers.

By the early 1980s, parallel computers were being manufactured commercially. The main advantage was the cost. Most of the parallel computers at that time were cheaper than their serial counterparts. The speed of the parallel computers is limited by the speed of light, therefore, the way of performing a computation more quickly is to move more bits of information at once, which is parallelism. These computers contain several processors together in order to solve a single problem. The questions remaining are how many processors should be used, how big they should be, and how should they be organized.

Multiple processor systems have a number of potential disadvantages, probably the most important being the very real problem of being able to use the processing power efficiently. For example, if a problem is solved by a processor in some particular time, then it is very difficult for the same problem to be solved in exactly half the time when two processors are used. This involves a number a factors; the ability to decompose a problem into an optimum number and size of modules, to define these modules in such a way that communications between processors may be carried out with the absolute minimum of waiting time, and with a minimal delay, across the communication network.

Present day real time problems tend to involve large amounts of data received at varying times and rates and yet requires responses to be generated instantly. To create a system which is totally general and yet provides maximum efficiency poses great problems.

III. Classification of Computers

Computer systems can be classified into a number of collective groups determined by the type of processing which is required, together with the method by which the processing elements communicate, use memory and operate with efficiency.

All computer systems, sequential and parallel can be divided according to the following schemes:

Feng's Scheme: based on *serial* versus *parallel* processing.

Handler's Classification: determined by the degree of *parallelism* and *pipelining* at various subsystem levels.

Shore's Taxonomy: based on how the computer is organized from its constituent parts (six machines).

Skillicorn's Taxonomy: based on the functional structure of the architecture and the data flow between its component parts.

Flynn's Classification: based on the multiplicity of *instruction streams* and *data streams* in a computer system. Most of the serial and parallel computers are classified according to Flynn's taxonomy. Therefore, it is discussed in detail. Four schemes of Flynn's classification are:

A. SISD (single instruction stream / single data stream)

This is the conventional serial John von Neumann computer as shown in Fig. 1 (a). Most serial computers fall into this category. Although instruction execution may be pipelined, computers in this category can decode only a single instruction in unit time. A SISD computer may have multiple functional units, but are under the direction of a single control unit.

B. SIMD (single instruction stream / multiple data streams)

These computers involve multiple processors simultaneously executing the same instruction on different data. These are the systems with multiple arithmetic-logic processors or units and a control processor as shown in Fig. 1 (b). Each arithmetic-logic unit processes a data stream of its own, as directed by the single control unit. SIMD machines are also called array processors.

C. MISD (multiple instruction streams / single data stream)

These computers involve multiple processors applying different instructions to a single data stream. There is no realization of this kind of architecture.

D. MIMD (multiple instruction streams / multiple data streams)

This consists of processing elements linked by an interconnection network or by accessing data in shared memory units. Each processing elements stores and executes independent instruction streams, using local data as shown in Fig. 1 (c). MIMD computers support parallel solutions that require processors to operate in a largely autonomous manner. Thus MIMD architectures are asynchronous computers, characterized by decentralized hardware control [8].

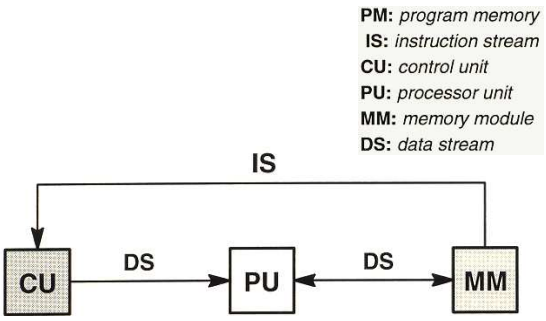

Fig. 1 (a) SISD computer

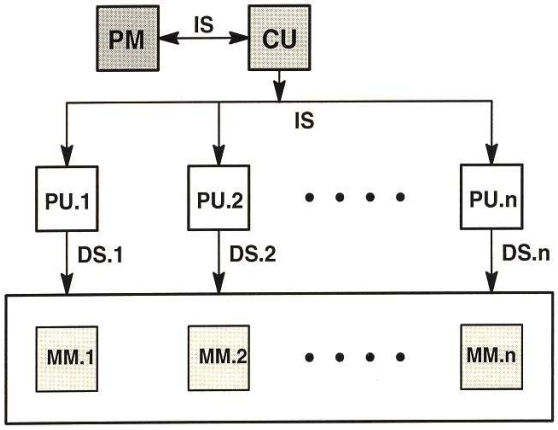

Fig. 1 (b) SIMD computer

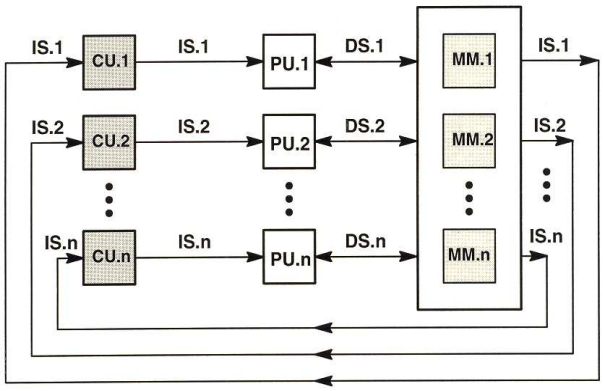

Fig. 1 (c) MIMD computer

With the aid of Flynn's classification, a structural taxonomy of both serial and parallel computers is formulated. The basic division is made according to the instruction streams. The single instruction computers consist of SISD and SIMD machines. The SISD computers include single-unit serials, multiple-unit scalars, and pipelined computers. The SIMD computers include processor arrays and associative processors.

In the multiple instruction stream, the MIMD architectures are divided into subgroups of multiprocessors and multi-computers. The multiprocessors are classified in terms of *loosely coupled*, which is sharing the local memory of the processors, and *tightly coupled*, which is that all the processors share a global memory through a central switching mechanism. The switching mechanism determines the processor organization and can be a common bus, a crossbar switch or a multistage switched network.

Multi-computers are characterized by distributed memory. Every CPU *(Central Processing Unit)* has its own memory, and all communications (point-to-point) and synchronization between processors are via message passing. The CPU's are inter-connected to form a processor organization, referred to as `topology'.

Dedicated computers were further divided into three basic types based on the ideas from MIMD computers: Array Processors, Pipeline Computers, and Very Large Scale Integration Computers. DSP Parallel architectures are another concept of hierarchical classification initiated in the early 1990s [9].

Parallel systems based on geometrical decomposition of applications are divided into three categories of a) computer based dedicated systems, b) computer based general systems, and c) digital signal processing systems. The first category includes array processor, pipeline computers, multiprocessor systems, very large scale integration, whereas the second one includes Distributed Shared Memory (DSM), Massive Parallel Processing (MPP), and Clusters, and the third one shares the combined capabilities of the first two categories [10]-[12].

IV. Hough Transforms

Hough transform technique allows discovering shapes from edges. It attempts to combine edges into lines where a sequence of edge pixels in a line indicates that a real edge exists. It is a popular procedure to detect lines and circles.

The simplest characterization of the Hough transform is to convert a difficult global detection problem in one space (image space) into a more easily solved local peak detection problem in another space (parameter space). A popular parameterization of a straight line is via the equation of the normal vector from the origin of the straight line,

$$\rho = x\cos(\theta) + y\sin(\theta)$$

where, ρ is the length of the normal to the line from the origin, and θ is the angle that the normal makes with the x-axis.

The ρ and θ parameters of a line are unique if $0^o \leq \theta \leq 180^o$. Points which are collinear in an image space all intersect at a common point in a parameter space and the coordinates of this parameter point characterizes the straight line connecting the image points. Using this parameterization each image point (x, y) generates a sinusoidal locus in (ρ, θ) space and thus lines are identified by the intersection of many of these sinusoids.

The Hough transform has been used in many applications in the field of medicine, character recognition, industrial inspection, military, shape recognition, geology, etc. [13].

V. Parallel Paradigms

In order to efficiently utilize the computational potential of a large number of processors in a parallel processing environment it is necessary to identify the important parallel features of the application. There are several simple paradigms for exploiting parallelism in scientific and engineering applications, but the most commonly occurring types fall into three classes. These three paradigms are described in more detail in [14], [15].

A. Algorithmic Parallelism

Is present where the algorithm can be broken down into a pipeline of processors in such a way that each processor executes a small part of the total algorithm. The parallelism inherent in the algorithm is exploited. In this decomposition, the data flows through the processing elements and is sometimes referred as *Data Flow* parallelism. The communication loads on each processor is increased and the communication bandwidth problems can degrade the performance.Therefore, for algorithmic parallelism to be successful the work load must be balanced uniformly across the processors. The advantage of this decomposition is that little data space is required per processor. The computer systems based on the algorithmic parallelism gives acceptable efficiency. Fig. 2 shows an example of algorithmic parallelism.

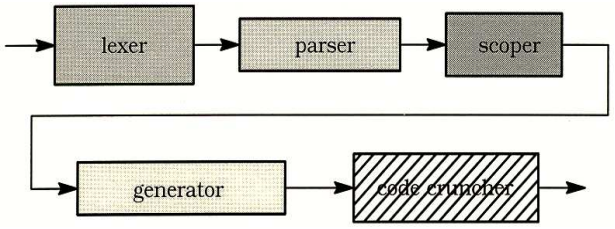

Fig. 2 Example algorithmic parallelism – a language compiler

B. Geometric Parallelism

Is present where the problem can be broken down into a number of similar processes in such a way as to preserve processor data locality and each processor operate on different subset of the total data to be processed. All the data required by a processor are arranged to be on that processor or one of its immediate neighbors.

In this decomposition, parallelism inherent in the data is exploited. This type of parallelism requires only a fraction of the total data on each processor, and is sometimes referred as *Data Structure* parallelism.

Processors communicate with the neighboring processors. The communication loads are proportional to the size of the boundary of the element, while the computational loads are proportional to the volume of the element. Each processor has an almost complete copy of the whole program, therefore moderate memory requirements. The computer systems based on the geometric parallelism gives very good efficiency.

C. Processor Farm

Is present where a program must be run large number of times with different parameters. It is often most efficient to run these independent tasks concurrently on different processors. The typical architecture for this type of applications is a farm of processors where each processor is executing the same program with different initial data in isolation from all the other processors. Large amounts of storage are therefore required on each processing element. Because of the very limited communication requirements, this method is very efficient, but the memory costs may be significant. Fig. 3 shows an example of data structure parallelism. Fig. 4 shows an example of processor farming [16], [17].

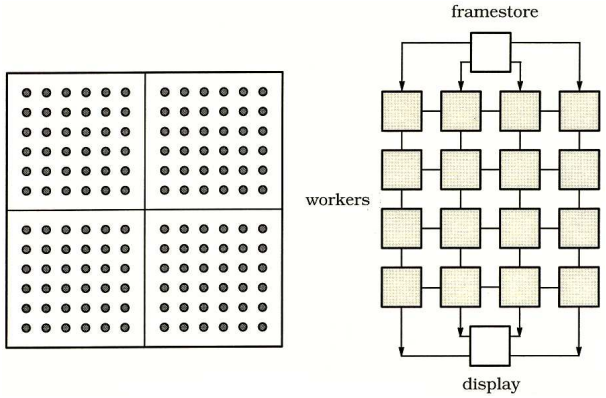

Fig. 3 Example geometric structure

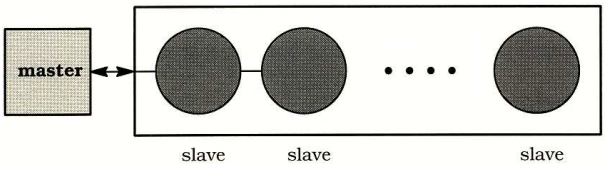

Fig. 4 Example processor farm

VI. Performance Measure

Two important measures of the quality of parallel algorithms implemented on multiprocessors and multi-computers are *speedup* and *efficiency*. The speedup achieved by a parallel algorithm running on *n* processors is the ratio between the time taken by that parallel computer executing the fastest serial algorithm and the time taken by the same parallel computer executing the parallel algorithm using *n* processors. The speedup 'S' is given by:

$$S(n) = T_1 + T_n$$

Efficiency is defined as the average utilization of the *n* allocated processors. The efficiency of a single processor system is 1. The relationship between efficiency 'E' and speedup is given by:

$$E(n) = S(n) / n$$

VII. The Hardware

In order to meet the high speed and performance, a scalable and reconfigurable multi-computer system (NPLA) is used. This networked multi-computer system is a bit similar to the NePA system used to implement Network-on-Chip [18].

The system used is a linear array of processors. It includes RISC processors and memory blocks. Each processor in the array has a compactOR, internal instruction memory, internal data memory, data control unit, and registers. One of the processors is used as a master or main processor and the remaining as slaves. The system has a network interface with the main processor having four and others equipped with two port routers. Routers can transfer both control as well as application data among processors.

VIII. Topology

The algorithms are executed on a total of 10 processor 8 of which constitute the processing elements. The remaining two are used for connection to the local host and for graphics display. The processing elements are connected in a linear array. The first processor in the chain of processors is known as the "master" processor. This interacts with the user through the local host, directs the operation of the graphics processor and the remainder of the processing elements, known as "slaves". The chain is connected by a bidirectional communication system, allowing data and results to be transferred from master to other processors and vice versa, see Fig. 5.

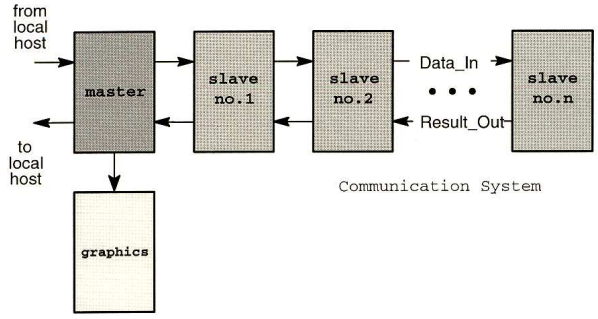

Fig. 5 Computer network for the topology

The implementation has two phases: computing ρ and θ for all the points in image space, then finding the peaks that identify the lines. The image space used is of a regular grid of 192 x 192 pixels. The range of values for θ is restricted to (0, π) to speed computation.

A. Static Scheduling:

In static scheduling, processes are allocated to processors at compile time. The master processor inputs the image from the host memory and stores it in a 2-dimensional image array. Then it divides the image space into regions according to the number (#) of slave processors and communicates the data from each region to the slave processors by doing a raster scan in such a way that slave # 1 receives data of the image points from region 1 and so on as shown in Fig. 6.

In all calculations a lookup table is used to replace calls to 'sin' and 'cos' library functions. Each slave processor holds an array of the table to avoid excessive communication overheads. For each (x, y) in the appropriate region, it then computes the values of ρ for each θ ranging from $0º to 180º and stores a vote in an accumulator array. The accumulator array of (ρ, θ) grid has a dimensionality of 464 x 180. The maximum value of ρ that can be computed for the chosen image space is from the range of 272 and -192. Therefore, the ρ index of the accumulator array is the addition of the above two values. When all the points for a region are computed, a lower threshold is applied to remove noise values in the form of lonely votes and the array is communicated back to the master processor.

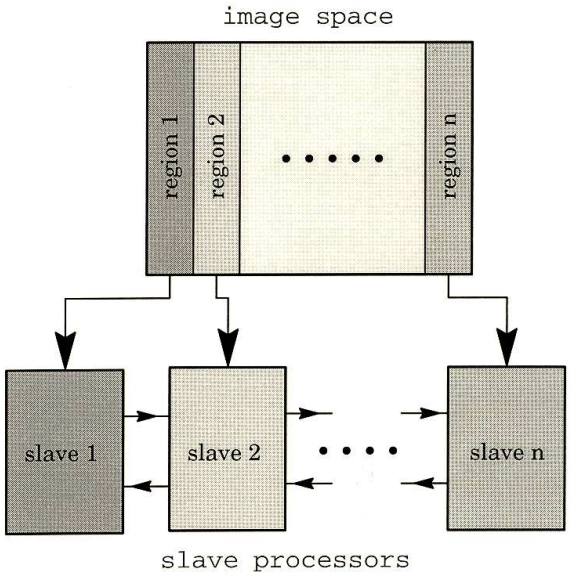

Fig. 6 Data distribution in static scheduling

The master accumulates all values returned by the slaves in another local accumulator array. After all the results are received and votes incremented, peaks are detected by scanning the array using an upper threshold. Peaks with maximum votes identify lines.

Experiments are performed on the proposed scheme with varying network sizes. Timings for 1 through 7 slave processors are obtained.

Table 1: shows the time taken in seconds, speedup and the efficiency to recognize few lines appearing in an image space.

B. Dynamic Scheduling:

In this implementation of the Hough transform, processes are allocated to processors at run time. The topology used is the same as in the static scheduling, which is a master processor and from 1 to 7 slaves. The slaves operate as a processor farm with the code replicated on each of them. The master processor distributes image points from the image space to the farm of processors.

TABLE I
RESULTS FROM STATIC SCHEDULING

# of Slave Processors	Timing in secs.	Speed-up	Efficiency %
1	3.14	1	100
2	2.26	1.39	69
3	1.74	1.8	60
4	1.53	2.05	51
5	1.43	2.19	44
6	1.29	2.43	40
7	1.26	2.49	36

Each slave processor executes two main processes in parallel. One is a *work* process where actual computation takes place and is run in low priority with the other which is a *task_schedular* as shown in Fig. 7.

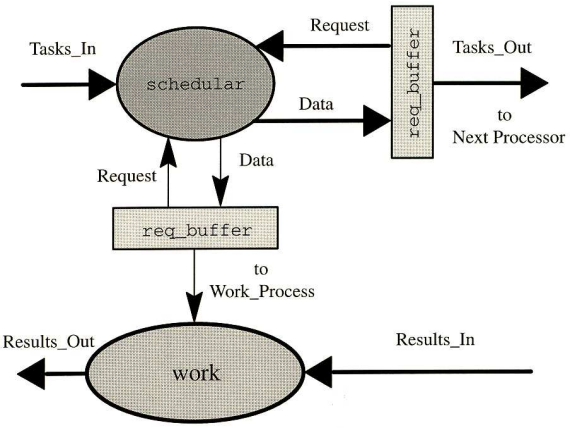

Fig. 7 Task_schedular in dynamic scheduling

In order to keep the slave processors busy in computing the image points, the task_schedular buffers an extra item of work so that when the work process completes the current computation for an image point it can immediately start computation on the next point rather than having to wait for the master processor to send another item of work.

The work process computes the Hough transform (ρ, θ) in the same way as for the static allocation explained earlier. After computing all the points associated for the particular processor, noise is removed and the resultant arrays are communicated back to the master processor where the peak values detect the lines.

Experiments are performed on the dynamic scheme using varying number of processors. Timings for 1 through 7 slave processors are obtained. Table 2: shows the time taken in seconds, speedup and the efficiency for the same image space as used for the static scheduling.

C. Statistic Scheduling:

In this implementation of the Hough transform, the master processor raster scans the image space and counts the number of the foreground points, and then divides the points equally among the slave processors. Distribution is done statically in the form of data arrays corresponding to the number of processors. Therefore, this scheduling scheme is statistically balanced and avoids communicating the entire image or regions to the slave processors.

TABLE II
RESULTS FROM DYNAMIC SCHEDULING

# of Slave Processors	Timing in secs.	Speed-up	Efficiency %
1	3.03	1	100
2	1.67	1.81	91
3	1.23	2.46	82
4	0.98	3.07	77
5	0.87	3.48	70
6	0.79	3.81	63
7	0.75	4.04	58

The slave processors receive an array of equal number of image points. Compute the results and store votes in the accumulator array, remove noise and send the results back to the master processor. Communication is buffered and prioritized. The master processor operates the same way as for the other two techniques to detect lines.

Experiments are performed on the statistically balanced scheme by varying the network size. Timings for 1 through 7 slave processors are obtained. Table 3: shows the time taken in seconds, speedup and the efficiency for the proposed scheme.

TABLE III
RESULTS FROM STATISTIC SCHEDULING

# of Slave Processors	Timing in secs.	Speed-up	Efficiency %
1	3.04	1	100
2	1.85	1.64	82
3	1.46	2.08	69
4	1.25	2.43	61
5	1.14	2.67	53
6	1.07	2.84	47
7	0.98	3.1	44

IX. COMPARISON

The results for the static, dynamic and statistic schemes are compared for the same image space and over the range of varying network sizes.

Fig. 8 shows time taken in seconds for the 3 schemes when from 1 to 7 slave processors are used. Time taken is nearly the same when only one slave processor is used for all the schemes. The timings improve for dynamic and statistic when 7 slave processors are used.

A 4 fold speedup is achieved for the dynamic scheme when the number of slaves is increased from 1 to 7, see Fig. 9. However, this level of speedup is not maintained for the other 2 schemes.

For static allocation, the speedup does not rise monotonically with the increase in the network size. This is due to the fact that additional processors may be allocated sparse areas of the image.

Fig. 10 shows the efficiency of the system when from 1 to 7 slave processors is used for the proposed schemes. The dynamic scheme shows an efficiency of nearly 60 percent.

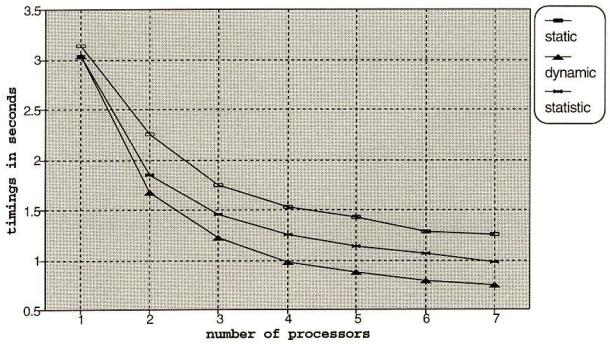

Fig. 8 Timing diagram for scheduling schemes

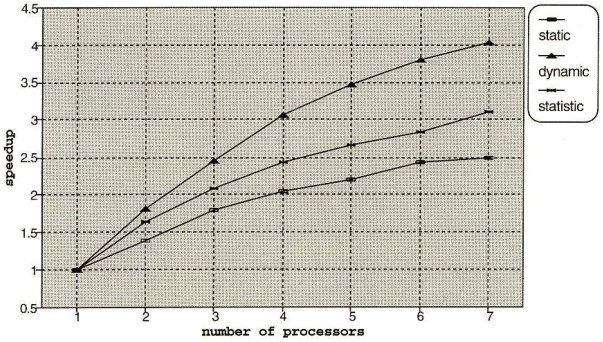

Fig. 9 Speedup graph for the scheduling schemes

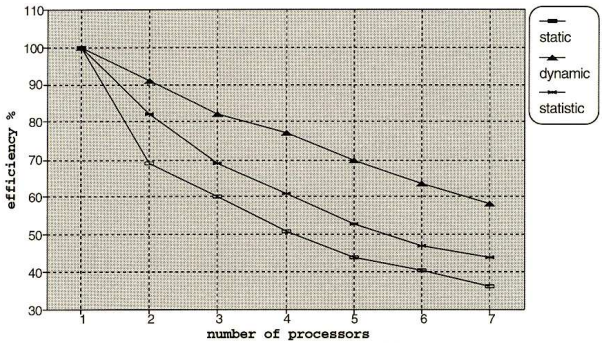

Fig. 10 Efficiency graph for the scheduling schemes

X. Conclusion

In this paper we have considered scheduling techniques for straight lines detection in digital images using the Hough transform method. The spatial and independent data characteristics, but a regular structure of computation for each image point of this algorithm is a representative of an important class of algorithms in computer vision and image processing. With the help of paradigms of parallel processing, the paper investigated the performance of static, dynamic, and statistic scheduling techniques for the parallel implementation of this type of algorithms on computer networks. Performed experiments suggest that dynamic scheduling can outperform its rivals in terms of speedup and efficiency, and is well suited to the MIMD structure of computer networks.

References

[1] T. L. Casavant and J. G. Kuhl, "A Taxonomy of Scheduling in General-Purpose Distributed Computing Systems," *IEEE Trans. on Software Engineering*, vol. 14, no. 2, Feb. 1988.

[2] P. V. C. Hough, "Method and means for recognising complex patterns," U.S. Patent No.3069654, 1962.

[3] R. O. Duda and P. E. Hart, "Use of the Hough Transformation to Detect Lines and Curves in Pictures," CACM, vol. 15, no. 1, Jan. 1972.

[4] Z. Zivkovic, R. Kleihorst, A. Danilin, and H. Corporaal, "Real-time implementations of Hough Transform on SIMD architecture," *in Proc. 2nd ACMIEEE Int. Conf. on Distributed Smart Cameras*, Palo Alto, California, 2008, pp. 1-8.

[5] A. Epstein, G. U. Paul, B. Vettermann, C. Boulin, and F. Klefenz, "A Parallel Systolic Array ASIC for Real-Time Execution of the Hough Transform," *IEEE Trans. on Nuclear Science*, vol. 49, no. 2, pp. 339-346, Apr. 2002.

[6] R. Strzodka, I. Ihrke, and M. Magnor, "A Graphics Hardware Implementation of the Generalized Hough Transform for fast Object Recognition, Scale, and 3D Pose Detection," *in Proc. 12th Int. Conf. on Image Analysis and Processing*, Mantova, Italy, 2003.

[7] S. S. Sathyanarayana, R. K. Satzoda, and T. Srikanthan, "Exploiting Inherent Parallelisms for Accelerating Linear Hough Transform," *IEEE Trans. on Image Processing*, vol. 18, no. 10, pp. 2255-2264, Oct. 2009.

[8] M. J. Flynn, "Very high-speed computing systems," *in proc. of the IEEE*, vol. 54, no. 12, pp. 1901-1909, 1966.

[9] M. Dongdong, L. Jinzong, Z. Bing, and Z. Fuzhen, "Research on the Architectures of Parallel Image Processing Systems," *in proc.2nd Int. Symp. on Intelligent Information Technology Application*, Shanghai, China, Dec. 2008, pp. 146-150.

[10] N. Zhang and J. Wang, "Image parallel processing based on GPU," *in proc. 2nd Int. Conf. on Advanced Computer Control*, Shenyang, China, 2010, pp. 367-370.

[11] Y. Krishnakumar, T. D. Prasad, K. V. S. Kumar, P. Raju, and B. Kiranmai, "Realization of a parallel operating SIMD-MIMD architecture for image processing application," *in proc. Int. Conf. on Computer, Communication and Electrical Technology*, Tirunelveli, Tamilnadu, India, 2011.

[12] H. Liu, Y. Fan, X. Deng, and S. Ji, "Parallel Processing Architecture of Remotely Sensed Image Processing System Based on Cluster," *in proc. 2nd Int. Congress on Image and Signal Processing*, Tianjin, China, 2009, pp. 1-4.

[13] A. G. Vicente, I. B. Muñoz, P. J. Molina, and J. L. L. Galilea, "Embedded Vision Modules for Tracking and Counting People," *IEEE Trans. on Instrumentation and Measurement*, vol. 58, no. 9, pp. 3004-3011, Sep. 2009.

[14] D. J. Pritchard, "Transputer Applications on Supernode," *in proc. Int. Conf. on Application of Transputers*, Liverpool, U.K., Aug. 1989.

[15] M. S. Laghari and F. Deravi, "Static vs. Dynamic Scheduling in Cellular Automaton," *in proc. of Fall meeting # 4, North American Transputer User Group*, Ithaca, New York, October 1990.

[16] A. S. Wagner, H. V. Sreekantaswamy, and S. T. Chanson, "Performance Models for the Processor Farm Paradigm," *IEEE Trans. on Parallel and Distributed Systems*, vol. 8, no. 5, pp. 475-489, May 1997.

[17] A. Walsch, "Architecture and Prototype of a Real-Time Processor Farm Running at 1 MHz," Ph.D. Thesis, University of Mannheim, Mannheim, Germany 2002.

[18] Y. S. Yang, J. H. Bahn, S. E. Lee, and N. Bagherzadeh, "Parallel and Pipeline Processing for Block Cipher Algorithms on a Network-on-Chip," *in proc. 6th Int. Conf. on Information Technology: New Generations*, Las Vegas, Nevada, Apr. 2009, pp. 849-854.