Microprocessing and Microprogramming 36 (1993) 167–178
North-Holland

Performance modeling of multiprocessor systems for different data loading schemes

M. Atiquzzaman

Department of Computer Science and Computer Engineering, La Trobe University, Melbourne 3083, Australia

Received 1 October 1991
Revised 10 December 1991, 26 January 1993
Accepted 12 May 1993

Abstract

Atiquzzaman, M., Performance modeling of multiprocessor systems for different data loading schemes, Microprocessing and Microprogramming 36 (1993) 167–178.

When evaluating the performance of an algorithm in a multiprocessor system, most authors assume that the data are already loaded in the system. This assumption gives rise to a large error between the predicted and actual performances, specially for fine-grained tasks like image processing. Different data loading and interprocessor communication techniques can result in a significant difference in the performance of a machine. To quantify the amount of time spent for data loading and unloading as compared to the total processing time, different data loading and unloading schemes have been developed in this paper. A comparison of performance of a machine under the different schemes has also been presented. The comparison will give an insight into the considerable amount of time data loading, unloading, and interprocessor communications may require depending on the task, the architecture of the machine, and the method used to load and unload data to and from the machine respectively. It has been shown that concurrent data loading, unloading, communication, and processing using state-of-the-art processors helps in significantly reducing the overhead. An expression to determine the optimum number of processors in a machine has been developed.

Keywords. Performance modeling; data loading; multiprocessor systems; linear array multiprocessor.

1. Introduction

Multiprocessor systems are used to speed up the execution of computation intensive tasks. A multiprocessor system consists of a number of processors which are connected together using an interconnection network like the multiple bus, ring, mesh, multistage network, etc. A multiprocessor system, in general, does not exhibit a linear speedup with an increase in the number of processors. Factors limiting a linear speedup are communication among processors, and loading and unloading of data to and from the machine respectively. The time required for loading and unloading the data becomes significant in applications where a large amount of data have to be processed.

Digital image processing is characterized by compute-bound algorithms which have to operate on a huge amount of data. To speed up the execution of algorithms, multiprocessor systems employing upto several thousands of processors are used for real-time image processing [1]. Images produced by visual sensors are loaded into a multiprocessor system for processing. The processed images are unloaded from the machine at the end of processing. This is typical of iconic image processing having a one-to-one mapping between the input and the output image sizes.

The total time to process an image consists of three components – loading the image into the machine, operating on the image, and unloading the processed image from the machine. Since the amount of data contained in an image is very large, the data loading and unloading time contributes a significant fraction to the total time required to process an image. *When evaluating the performance of an image processing algorithm on a multiprocessor system, most researchers have a tendency to assume that the images are already loaded in the machine* [2–4]. *Results relating to the performance of an algorithm on a particular architecture, based only on the computation time, are in most cases misleading and do not represent the performance under actual working conditions.* It would be far more realistic to take into consideration the data loading and unloading times when determining the performance of a machine for executing fine-grained low-level image processing algorithms. An example of a machine where data loading and unloading times should be considered in performance evaluation is the Massively Parallel Processor (MPP) [5] which does not have a special I/O structure. Therefore, loading programs and data to the processor array of the MPP via the VAX 11/780 host consumes far more time than any preprocessing [6]. The objectives of this paper are as follows:

- To determine the performance of a linear-array multiprocessor [7,8] system under different loading, unloading and processing schemes.
- To develop an expression for the optimum number of processors required for an algorithm in a particular architecture.
- To show that the performance figures of multiprocessor systems, obtained by ignoring (as is done by most researchers) the data loading and unloading times, do not reflect the true performance of the machine. The performance figures, to be presented in this paper, will be obtained by using a realistic workload and measured values of communication parameters from a commercial multiprocessor.

The rest of the paper is organized as follows. A brief description of previous work in performance modeling of linear array multiprocessors is given in Section 2. To evaluate the expected performance of a linear-array multiprocessor system executing an image processing task, we develop a computational model of the system in Section 3. Section 4 describes the different data loading and unloading schemes which have been used in this paper for performance evaluation. Performance metrics and results are presented in Section 5, followed by a conclusion in Section 6.

2. Previous work

Several researchers have developed analytical models for the performance evaluation of linear array multiprocessor systems. Some of the authors have presented results for their model by using a general workload, while others have assumed a realistic workload for their model. To show the difference between the emphasis of our proposed model and previous models, a brief summary of the previous models is given below.

The effects of data loading, unloading, and interprocessor communication on the performance of multiprocessor systems of different topologies have been reported in [9]. Analytical expressions for the maximum power that can be achieved from a linear array system and the optimum number of processors in the system have been developed. Experimental results (without analytical modeling) of speedup for a number of frequently used algorithms (matrix inversion, quicksort, Hartley transform, Knapsack problem, Fibonacci-series, Queen's problem) on a Transputer-based linear array system have been presented. The performance of matrix inversion on a crossbar system, obtained from experimental measurements, has also been presented. The model does not consider any particular workload, nor does it show the possible results for any commercial machine. There is no indication of the values of the communication parameters that could be used in the model. Performance results from the analytical models were, therefore, not presented. Consequently, the validity of the model could not be assessed, nor could the accuracy of the results be determined.

The degradation in performance of multiprocessor systems (shared bus, multistage interconnection network, and mesh) due to interprocessor communications have been reported in [10]. Implementations of several FFT algorithms with different interprocessor communication requirements were modeled. The data loading and unloading times have not been considered in the model. The results indicate that a significant gain in speedup can be achieved from algorithms requiring little interprocessor communications.

Performance modeling of multiprocessor systems, taking into consideration the effects of communication speed and data routing, have been described in [11]. Hypercube and grid topologies have been considered in the study. It has been shown that wormhole routing greatly reduces the communication latency, and results in an improved performance of the system.

The performability modeling of the Intel iPSC/2, taking communication latency into consideration has been described in [2,12]. A general workload has been assumed in [2]. Matrix multiplication has been used as an example of a realistic workload in [12].

The proposed model, to be presented in the next sections, differs from the previous models in the fact that a realistic workload and actual communication parameter values have been used to demonstrate the performance of a linear array multiprocessor system. Moreover, different data loading and unloading schemes have been presented, and their effects on the overall performance have been considered.

3. Computational model

The computational model we consider in this paper is a linear array of $N = 2^n$ processing elements (PEs) (*Fig. 1*). The PEs are numbered as PE_0, PE_1, $\cdots$, PE_{N-1} according to their address. PE_i, $i = 1, 2, \ldots, N-2$, is connected by dedicated inter-PE links to PE_{i-1} and PE_{i+1}.

An image is divided into several sub-images, each sub-image being processed in a different PE. The results are passed to a host computer which is responsible for the extraction of global features. Digitized images are loaded into the array of PEs via PE_0 (*Fig. 1(a)*), or through a video bus (VB) to which all the PEs are connected (*Fig. 1(b)*). Should it be required, the common bus (CB) can be used for communication among non-neighboring PEs. Such non-neighbor communications are usually not required in low-level image processing tasks.

We assume a median filtering operation on an image $I = \{I(x, y) : 0 \leq x, y \leq M - 1\}$, $M = 2^m = jN, j \geq 1$. The median filtering over a subimage of size $(2k + 1) \times (2k + 1)$ is defined as the problem of finding the $(2k + 1)^2/2$th largest pixel among the pixels in the window.

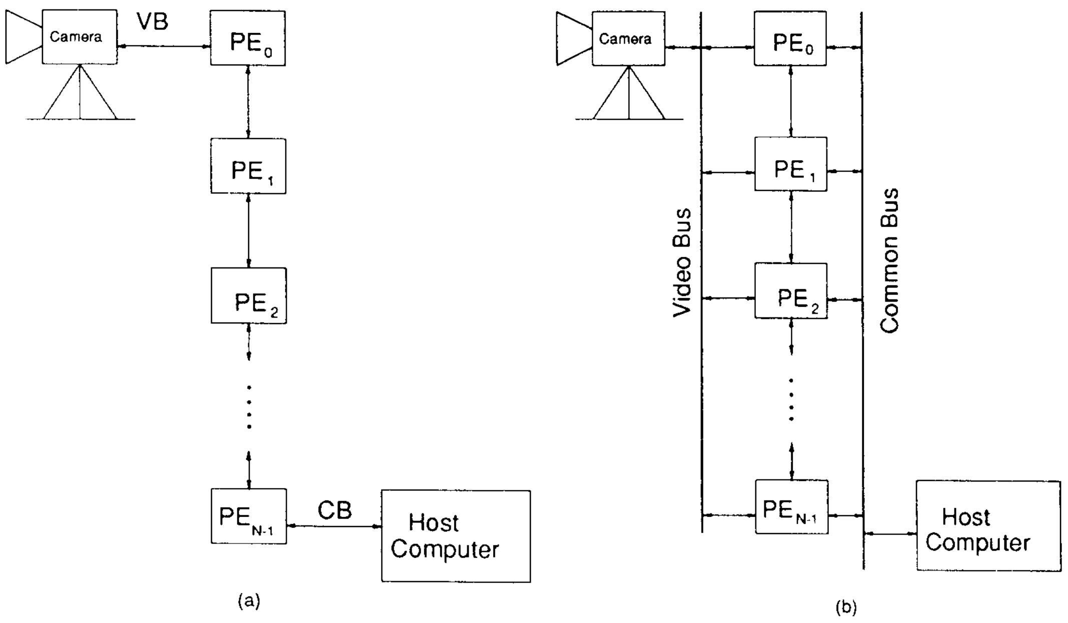


Fig. 1. Linear array of PEs, (a) Scheme I, (b) Scheme II.

The following assumptions are made for the analysis:

- All PEs are identical, have the same computing power, and are small computers in themselves having their own local RAMs, EPROMs, and I/O ports.
- Depending on the data loading and unloading methods, the array is either a SIMD machine or a Single-Code Multiple-Data (SCMD) stream machine [13]. In a SCMD machine, all PEs execute the same program code, but on different data sets, and the PEs are not synchronized in time. They may start execution at different instants of time depending on the availability of data. The PEs in a SCMD machine have their own control units, and can be synchronized to an external event if SIMD operation is desired.

The computational model described above can be abstracted by a set of parameters. The total time to handle one frame of an image is given by

$$t_h = f_h(t_l, t_p, t_c, t_u),$$

where f_h is a function, t_l = loading time of the picture, t_p = processing time of the picture, t_c = inter-PE communication time, and t_u = time to unload the picture. We characterize t_l, t_p, t_c, t_u by the following equations:

$$t_p = f_p(N, M, P) = f_p(I_s, P)$$
$$t_l = f_l(N, M, L)$$
$$t_c = f_c(I_c)$$
$$t_u = f_u(N, M, U)$$

where f_p, f_l, f_c, and f_u are functions, P is an image processing task, I_s is the size of the sub-image assigned to a PE, L and U are picture loading and unloading methods respectively, and I_c is the amount of data which needs to be interchanged between PEs. Throughout this paper, the unit of data is a line of pixels.

4. Modes of operation

The image I is partitioned into N subimages such that each PE processes $I_s = M/N$ lines of a picture. We assume

$$t_l = c_{l_s} + c_{l_1} I_s + c_{l_2} I_s^2,$$

where c_{l_s} is the communication channel *setup time* between the image source and a PE, and c_{l_1} and c_{l_2} are *proportionality constants* for the first and second order terms respectively. Similarly,

$$t_u = c_{u_s} + c_{u_1} I_s + c_{u_2} I_s^2,$$

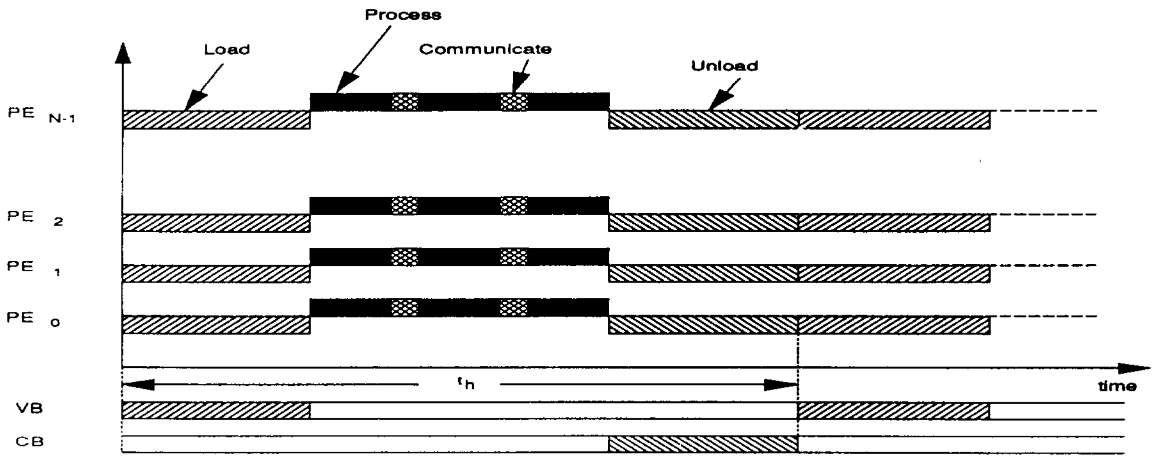


Fig. 2. State diagram for Scheme IA.

We assume that the processing time (t_p) of a PE is a linear function of the sub-image size:

$$t_p = c_{p_v} + c_{p_1} I_s,$$

where c_{p_v} accounts for the time to initialize counters, variables etc. The above assumption of linear processing time is valid in the case of tasks (P) like filtering, convolution, etc. t_p may have higher order terms for operations like correlation. t_c will be presented by

$$t_c = c_{c_s} + c_{c_1} I_c + c_{c_2} I_c^2$$

where I_c is the amount of data that has to be interchanged with neighboring PEs, and c_{c_s} accounts for the inter-PE communication channel setup time. We consider two schemes of data loading in the machine.

4.1. Scheme I: Serial data loading and processing

In this scheme, the image is first loaded into the machine followed by processing. This is typical of many systems [14] where pixels are loaded through one end of the machine and then shifted through the PEs until the pixels are properly registered with the appropriate PEs. The scheme can be further divided into two subschemes as follows.

4.1.1. Scheme IA: Separate loading, processing, and unloading

The full picture is first loaded into the machine by shifting the data through the processors followed by processing of the data. The processed picture is unloaded before a new picture is loaded. *Fig. 2* is a Gantt chart showing the activities of the PEs at different epochs of time for this scheme. Time to handle a single frame in scheme IA (t_h) is given by

$$\begin{aligned} t_h &= t_l + t_p + t_c + t_u \\ &= c_{l_s} + c_{l_1} I_s + c_{l_2} I_s^2 + c_{p_v} + c_{p_1} I_s + c_{c_s} + \\ &\quad c_{c_1} I_c + c_{c_2} I_c^2 + c_{u_s} + c_{u_1} I_s + c_{u_2} I_s^2 \\ &= c_{l_s} + c_{l_1} M + c_{l_2} M^2 + c_{p_v} + c_{p_1}(M/N) + \\ &\quad 4n_p(c_{c_s} + c_{c_1} k + c_{c_2} k^2) + c_{u_s} + c_{u_1} M + \\ &\quad c_{u_2} M^2. \end{aligned} \tag{1}$$

For the case of median filtering, each PE has to communicate with two neighboring PEs, and twice with each neighbor – once for receiving and once for sending the pixels. Therefore, for a window of size $(2k + 1) \times (2k + 1)$, PE$_i$ will send k lines of pixels to each of the neighbors PE$_{i-1}$ and PE$_{i+1}$, and also receive the same number of lines from

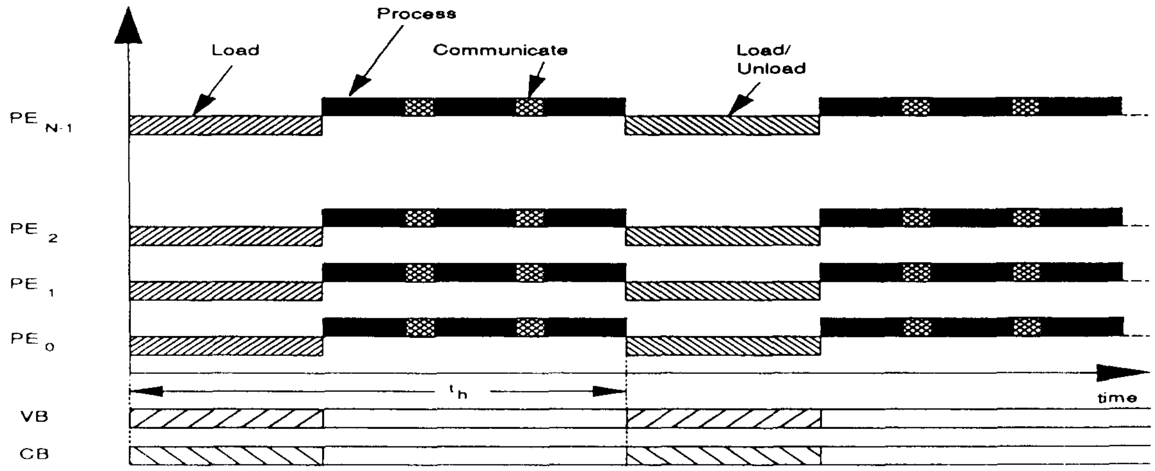


Fig. 3. State diagram for Scheme IB.

them resulting in an interprocessor communication of $4k$ lines of pixels for PE_i. In some instances, n_p passes over a picture may be required.

4.1.2. *Scheme IB: Concurrent loading and unloading*

The processed picture is unloaded concurrently with the loading of a new picture (*Fig. 3*), as if the new pixels being shifted through the PE array pushes the processed picture out of the array. Since, no separate time is needed for unloading, t_h is given by

$$\begin{aligned} t_h &= t_l + t_p + t_c \\ &= c_{l_s} + c_{l_1}M + c_{l_2}M^2 + c_{p_v} + c_{p_1}(M/N) + \\ &\quad 4n_p(c_{c_s} + c_{c_1}k + c_{c_2}k^2). \end{aligned} \tag{2}$$

4.2. *Scheme II: Concurrent data loading and processing*

In the second scheme, a processor starts processing as soon as data are ready. This requires processing being overlapped with data loading and unloading. Since a PE starts processing as soon as data is ready, it cannot perform the data shifting operations as was done in scheme I. Therefore, the architecture shown in *Fig. 1(b)* is used where the PEs receive pixels from a video bus. This architecture can also be used in scheme I. However, the expressions derived earlier would remain unchanged since PE_0 would be the only processor connected to VB. We can subdivide this scheme into two further subschemes depending on whether the concurrency is at the system level or at the processor level as follows.

4.2.1. *Scheme IIA: Concurrency at the system level*

The system as a whole supports concurrent loading, processing and unloading of images. However, a processor has to perform the above three operations in sequence. We distinguish between a compute-bound and an I/O-bound system as follows.

Case 1. Compute-bound system

$$t_p + t_c + t_u \geq (N-1)\,t_l.$$

The processors are busy throughout the operation of the system, whereas the busses remain idle for some periods (*Fig. 4*). The handling time is given by

$$\begin{aligned} t_h &= t_l + t_p + t_c + t_u \\ &= c_{l_s} + c_{l_1}(M/N) + c_{l_2}(M/N)^2 + c_{p_v} + \\ &\quad c_{p_1}(M/N) + 4n_p(c_{c_s} + c_{c_1}k + c_{c_2}k^2) + \\ &\quad c_{u_s} + c_{u_1}(M/N) + c_{u_2}(M/N)^2. \end{aligned} \tag{3}$$

Case 2. I/O-bound system

$$t_p + t_c + t_u < (N-1)\,t_l.$$

The processing, communication, and unloading take less time than the loading of the picture. The busses are always busy, while the processors are idling at certain times (*Fig. 5*). For such a system

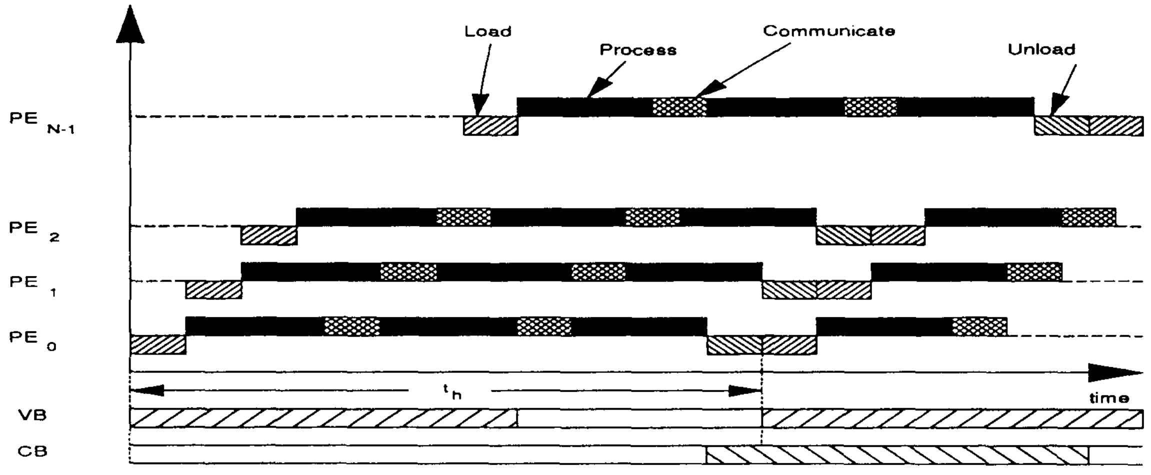


Fig. 4. State diagram for Scheme IIA (Case 1).

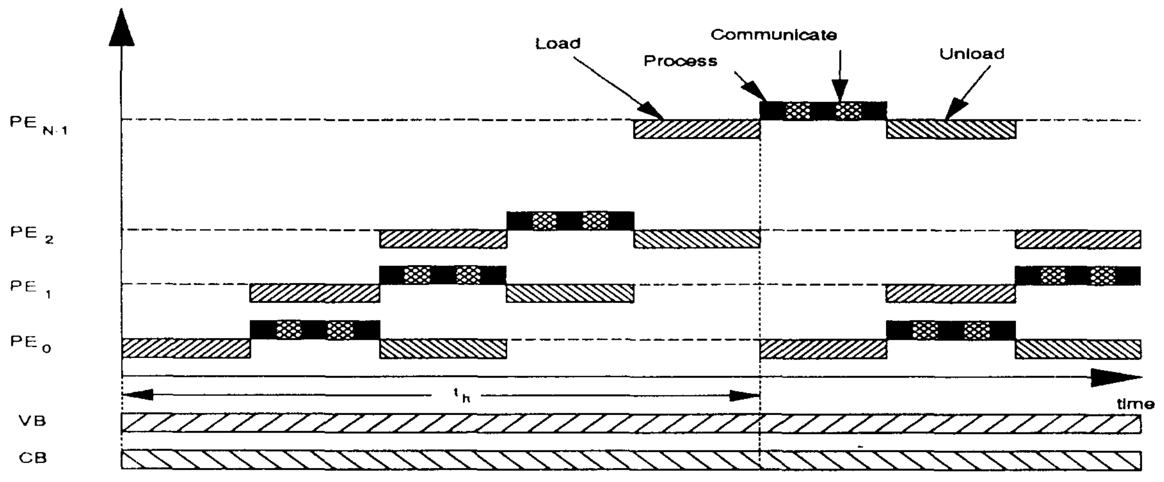


Fig. 5. State diagram for Scheme IIA (Case 2).

$$t_h = \max\{N(c_{l_s} + c_{l_1}I_s + c_{l_2}I_s^2),\; N(c_{u_s} + c_{u_1}I_s + c_{u_2}I_s^2)\}. \quad (4)$$

4.2.2. Scheme IIB: Concurrency at the processor level

In this scheme, a PE can perform the loading, processing and unloading operations in parallel. Such state-of-the-art processors are already available commercially, like the Inmos Transputer [15]. The Transputer is a 32-bit microprocessor with four dedicated links. All the links and the CPU can operate concurently. Simulation and design of a linear-array multiprocessor using Transputers can be found in [16]. If such a processor is not used, then separate I/O processors may be employed to carry out the loading and unloading operations. This is typical in many computer systems.

Case 1. Compute-bound system

$$t_p + t_c \geq \max\{Nt_l, Nt_u\}.$$

For a compute-bound system, the buses are idling at certain times while the processors are al-

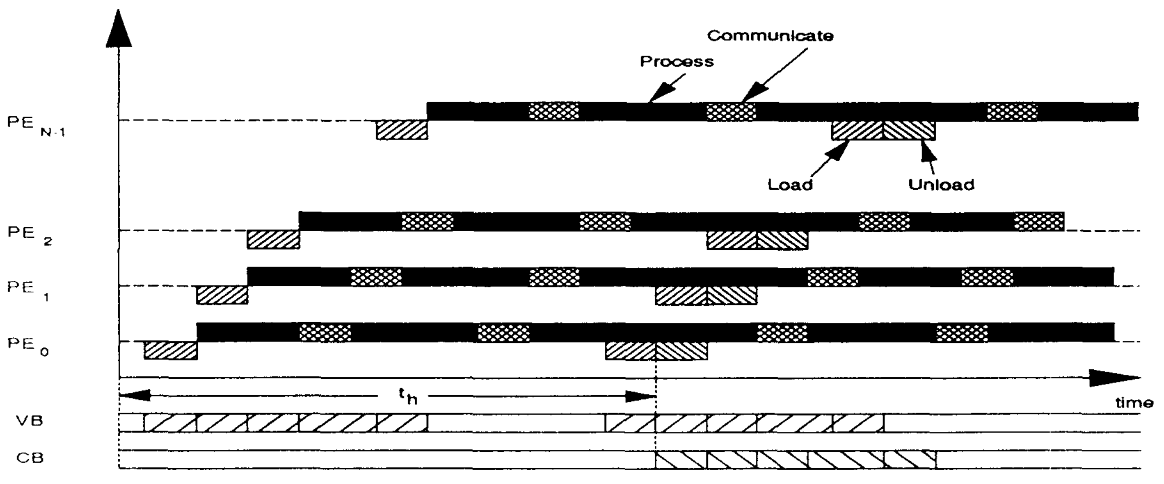


Fig. 6. State diagram for Scheme IIB (Case 1).

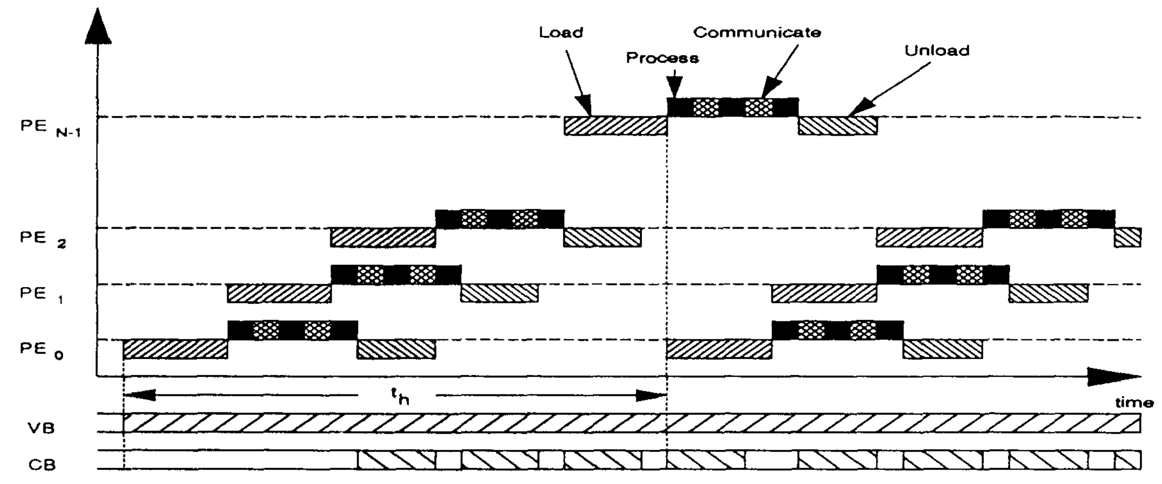


Fig. 7. State diagram for Scheme IIB (Case 2).

ways busy (*Fig. 6*). The handling time is given by

$$\begin{aligned} t_h &= t_p + t_c \\ &= c_{p_v} + c_{p_1}(M/N) + \\ &\quad 4n_p(c_{c_s} + c_{c_1}k + c_{c_2}k^2). \end{aligned} \tag{5}$$

Case 2. I/O-bound system

$$t_p + t_c < \max\{Nt_l, Nt_u\}.$$

For an I/O-bound system, the processors are idling at certain times while at least one of the busses (either VB or CB) is saturated (*Fig. 7*).

$$\begin{aligned} t_h = \max\{&N(c_{l_s} + c_{l_1}I_s + c_{l_2}I_s^2), N(c_{u_s} + \\ &c_{u_1}I_s + c_{u_2}I_s^2)\}. \end{aligned} \tag{6}$$

5. Results

Several well-established criteria exist in the literature [17] for the performance measurement and evaluation of multiprocessor systems. As a measure of performance, we will consider the execution time, speedup, efficiency, and utilization since these are the basic measures from which the rest can be determined. *Execution time* is defined as

$$t_N(M) = t_N^c(M) + t_N^o(M),$$

where $t_N(M)$ is the execution time to process (handle) M data points (M lines of a picture in our case) using an N-processor system. $t_N^c(M)$ and $t_N^o(M)$ are the computation time and overhead times respectively. *Speedup* is defined as the increase in the computing speed for an N-processor system as compared to a single-processor system:

$$s_N(M) = \frac{t_1(M)}{t_N(M)}.$$

Efficiency for an N-processor system is defined as

$$e_N(M) = \frac{s_N(M)}{N},$$

where $s_N(M)$ is the speedup as defined above. The efficiency represents the fraction of the maximum attainable speed with an N-processor system. *Utilization* can be expressed as

$$u_N(M) = \frac{\sum_{i=1}^{i=N} t_{N_i}^c(M)}{N t_N(M)},$$

where $t_{N_i}^c(M)$ represents the computation time of the ith processor in an N-processor system. The utilization of a system represents the fraction of the total time during which the PEs are performing computations, i.e. it represents the average computation time. To measure the amount of time spent for loading and unloading as a fraction of the total time, we define the *load/unload factor* as follows:

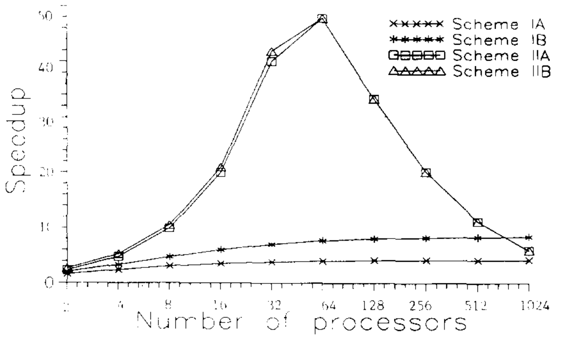


Fig. 8. Speedup vs. number of processors.

$$l_N(M) = \frac{\sum_{i=1}^{i=N} t^l_{N_i}(M)}{N t_N(M)},$$

where $t^l_{N_i}(M)$ represents the loading and unloading times for the ith processor in an N-processor system.

The speedup, efficiency, and utilization for the different schemes as a function of N, for a fixed M ($= 1024$), and a particular task P ($=$ median filtering) have been determined from the analytical expressions developed in the previous section. Instead of using hypothetical values for the different parameters mentioned in Section 4, and arriving at unrealistic performance figures, we have used the values of the parameters for the Intel iPSC/1 as measured by Lee [2].

Figures 8 and *9* are the logarithmic plots of the speedup and efficiency. As expected, schemes IIA and IIB are significantly better than schemes IA and IB upto a certain number of processors, after which the speedup of schemes IIA and IIB starts falling. For example, when the number of processors is 1024, the speedup with schemes IIA and IIB worsens to an extent where it is even lower than scheme IB. This is because the data loading and unloading times in schemes IIA and IIB, for a large number of processors, contribute a significant proportion to the total execution time.

The efficiency in scheme IIB for $N \leq 32$ in *Fig. 9* is greater than unity. This is because the state-of-the-art processors used in this scheme have the capability of processing being overlapped with I/O, while the processors of scheme I do not have this capability. In fact, if these state-of-the art processors are used in scheme I, there would be no change in the performance because concurrent loading and processing of the images at the system level is not supported in scheme I. The overhead for data loading and unloading have been counted separately in scheme I, while schemes IIA and IIB overlap the overhead with computation, thereby decreasing the overall time significantly. The efficiency would have been unity if scheme I supported simultaneous data loading and unloading *at the system level.*

The efficiency plot also shows that $e_N(M)$ decreases continuously for scheme I, while it remains almost steady for scheme IIB for small values of N, and starts falling as N increases. This is due to the fact that scheme IIB is relatively robust against increase in $t^o_N(M)$ with increasing N, because $t^o_N(M)$ is overlapped in time with $t^c_N(M)$ until the point where $t^o_N(M)$ exceeds $t^c_N(M)$ and the efficiency starts falling.

It is interesting to note that in scheme IIA, $e_N(M)$ increases until a point is reached where it starts falling off. This can be explained as follows.

$$e_N = \frac{s_N(M)}{N} = \frac{t_1(M)}{t_N(M) \times N}. \tag{7}$$

Depending on the scheme used, $t_N(M)$ can consist of t_l, t_p, t_c, and t_u. For the values of the parameters used in this analysis, t_c is negligible, t_p decreases linearly with increasing N, and t_l and t_u decrease exponentially with increasing N because of the second order terms. The overall effect is that $t_N(M)$ decreases at a faster rate than the increase of N, resulting in an increase in efficiency. As N increases, t_l and t_u become insignificant (since I_s decreases, t_l and t_u

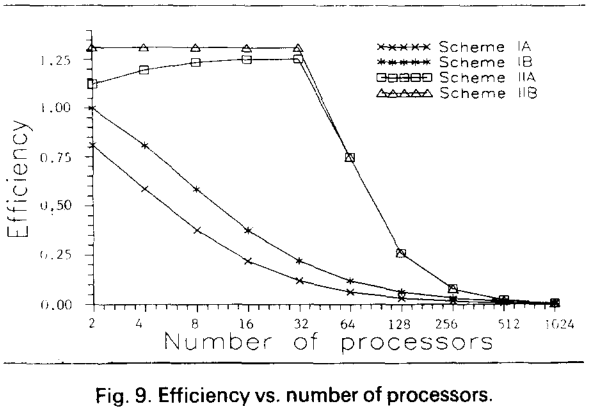


Fig. 9. Efficiency vs. number of processors.

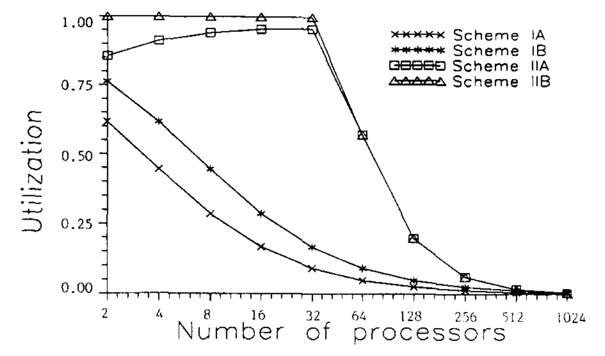


Fig. 10. Utilization vs. number of processors.

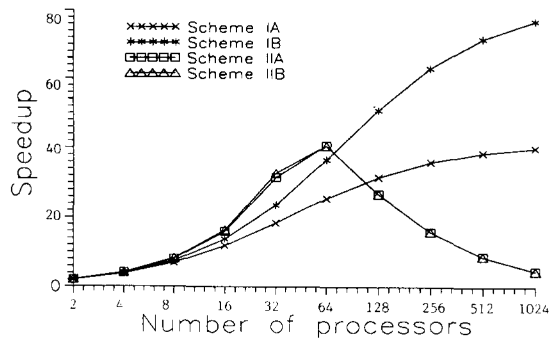


Fig. 12. Speedup vs. number of processors (second order terms ignored).

decrease exponentially), and therefore, the curve becomes almost constant. These decreases in t_l and t_u are evident from the increase in utilization (*Fig. 10*).

The increase in efficiency is not observed in scheme IIB because I/O is done in parallel to the computation, and hence does not contribute to $t_N(M)$ until the machine becomes bus-bound, when the efficiency starts falling for both schemes IIA and IIB.

As expected, the increase in efficiency stated above is not observed if the second order terms are neglected as seen in *Fig. 11*. But this neglect gives rise to a large error in performance calculation as is evident from *Figs 12* and *13* which are the plots of speedup and utilization obtained by neglecting the second order terms.

The main reason for this decrease in speedup is the change in the state of the machine from a compute-bound to an I/O-bound state. Once the machine becomes I/O-bound, the I/O channel is saturated and a further increase in the number of processors results in a drop in the speedup.

The crossover point, N_{co}, at which the speedup in scheme IIA starts falling is given by

$$t_p + t_c + t_u = (N_{co} - 1)\, t_l.$$

Substituting values for t_l, t_p, t_c, and t_u and then rearranging the terms gives

$$\begin{aligned} &c_{p_v} + c_{p_1}(M/N_{co}) + 4n_p(c_{c_s} + c_{c_1}k + c_{c_2}k^2) + \\ &\quad c_{u_s} + c_{u_1}(M/N_{co}) + c_{u_2}(M/N_{co})^2 - \\ &\quad (N_{co} - 1)(c_{l_s} + c_{l_1}(M/N_{co}) + \\ &\quad c_{l_2}(M/N_{co})^2) \approx 0. \end{aligned} \tag{8}$$

Equation 8 can be simplified to give a 3rd order equation in N_{co}. The solution of the equation gives

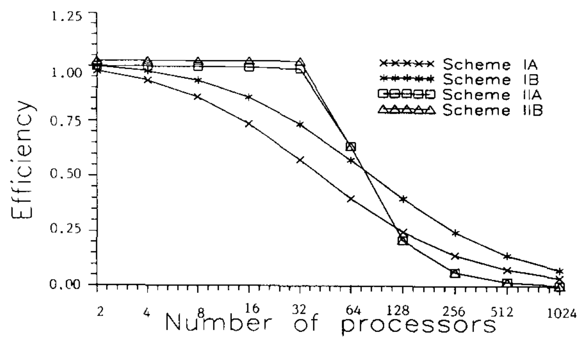


Fig. 11. Efficiency vs. number of processors (second order terms ignored).

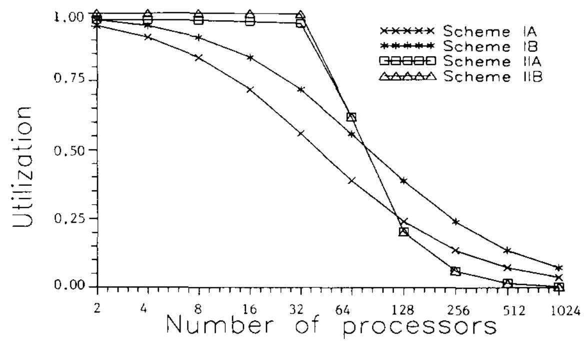


Fig. 13. Utilization vs. number of processors (second order terms ignored).

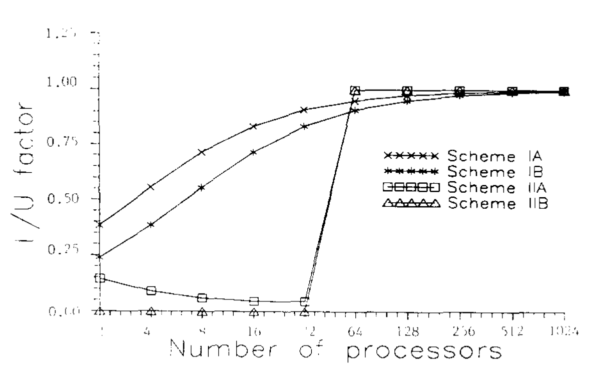


Fig. 14. Load/unload factor vs. number of processors.

the optimum number of processors in a linear array multiprocessor system using scheme II for data loading and unloading.

For the values of the parameters used in this paper, N_{co} is found to be 43.34. Since N is an integer power of two, we use $N_{co} = 32$. *Figure 14* shows the load/unload factor taking into account the second order terms. It represents the fraction of time the machine spends in loading and unloading data as compared to the total handing time. It is seen from the figure that a considerable amount of time is spent in scheme I for data loading and unloading, while in scheme II the machine becomes totally I/O-bound after N_{co}. *Due to this significant amount of time spent for loading and unloading, we conclude that performance figures of multiprocessor systems, obtained without taking into account the data loading and unloading times, may be in severe errors and could be misleading.* This is very important specially in the case of fine-grained tasks like low-level image processing, where hundreds or thousands of processors may be employed to process a huge amount of data.

6. Conclusions

Different data loading and unloading schemes for a linear-array multiprocessor system have been considered. It is evident that for $N \leq N_{co}$, scheme II is better than scheme I. Moreover, state-of-the-art processors like Transputers can be used in scheme II. A comparison of speedup, efficiency, utilization, and load/unload factor has been presented to compare and contrast the performance of the different schemes. It has also been shown that neglecting second order terms in the expressions for loading, unloading, and inter-processor communications, gives rise to serious errors.

Analytical expression for the optimum number of processors in a linear array multiprocessor system using scheme II has been developed. Similar expressions can also be developed for other schemes. Furthermore, it has been shown that performance measurements of fine-grained parallel algorithms, obtained by neglecting the data loading and unloading times, are misleading and may not reflect the true performance of the machine.

The models developed by previous authors have either assumed general workloads or tasks which involve processing a small amount of data. Since, the aim of our study was to show the effects of data loading and unloading on the performance of a linear array system, an image processing task involving the loading and unloading of a large amount of data was assumed as the workload. Hence, a direct comparison between the results from the proposed model and previous models is not meaningful in this context.

References

[1] T.J. Fountain, *Processor Arrays: Architectures and Applications* (Academic Press, London, 1987).

[2] S.Y. Lee and J.K. Aggarwal, Exploitation of image parallelism via the hypercube, in: M.T. Heath, ed., *Hypercube Multiprocessors,* (SIAM, Philadelphia: 1987) 426–437.

[3] P.J. Marrow and R.H. Perrot, The design and implementation of low-level image processing algorithms on a transputer network, in: I. Page, ed., *Parallel Architectures and Computer Vision* (Clarendon Press, Oxford, 1988) 243–259.

[4] A.N. Choudhary and R. Ponnusamy, Implementation and evaluation of Hough algorithms on a shared-memory multiprocessor, *J. Parallel Distributed Comput.* 12 (2) (June 1991) 178–188.

[5] K.E. Batcher, Design of a massively parallel processor, *IEEE Trans. Comput.* C-29 (9) (Sep. 1980) 836–840.

[6] T.J. Fountain, A review of SIMD architectures, in: J. Kittler, ed., *Image Processing System Architectures* (Research Studies Press, 1985) 3–22.

[7] M. Atiquzzaman and W.H. Shehadah, A microprocessor-based office image processing system, *IEEE Trans. Comput.* 41 (3) (March 1992) 379–381.

[8] Y. Okawa, A linear multiple microprocessor system for real-time picture processing, *IEEE Conf. on Pattern Recognition and Image Processing* (June 14–17, 1982) 393–395.

[9] W.H. Burkhardt, Performance penalty by communication in multiprocessor systems, *Internat. Conf. on Parallel Processing* (August 12–16, 1991) I 660–661.

[10] R.N. Mahapatra, V.A. Kumar, B.K. Das and B.N. Chatterji, Performance of parallel FFT algorithm on multiprocessors, *Internat. Conf. on Parallel Processing* (August 13–17, 1990) III 368–369.

[11] X. Zhang, System effects of interprocessor communication latency in multicomputers, *IEEE Micro* 11 (April 1991) 12–15.

[12] S.M. Koriem and L.M. Patnaik, Performability studies of hypercube architectures, *6th Internat. Parallel Processing Symp.* (March 23–26, 1992). 488–495.

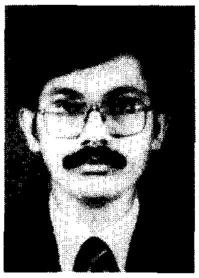
[13] Q.F. Stout, Pyramid algorithms optimal for the worst case, in: L. Uhr, ed., *Parallel Computer Vision* (Academic Press, New York, 1987) 147–168.

[14] M.J.B. Duff, Review of the CLIP image processing system, *Proc. Nat. Computer Conf.* (1978) 1055–1060.

[15] Transputer Reference Manual, Inmos Ltd., 1985.

[16] M. Atiquzzaman, Algorithms and architectures for automatic traffic analysis, PhD thesis, University of Manchester Institute of Science and Technology, England, 1987.

[17] L.H. Siegel, H.J. Siegel and P.H. Swain, Parallel algorithms performance measures, in: K. Preston, ed., *Multicomputers and Image Processing* (Academic Press, New York, 1982) 241–252.



M. Atiquzzaman received the M.Sc and Ph.D. degrees in electrical engineering and electronics from the University of Manchester Institute of Science and Technology, England in 1984 and 1987 respectively. From 1987 to 1992, he was an assistant professor in the department of electrical engineering at King Fahd University of Petroleum & Minerals, Saudi Arabia. Since 1992 he has been a lecturer in the Department of Computer Science and Computer Engineering at La Trobe University, Melbourne, Australia. His current research interests are in computer architecture, multiprocessor systems, interconnection networks, parallel processing, image processing, and pattern recognition.