Malaysian Journal of Computer Science, Vol. 10 No. 1, June 1997, pp. 1-9

COMPUTATION EFFICIENCY OF MASTER-SLAVE PROCESSORS IN MULTITASKING APPLICATIONS: A PERFORMANCE ANALYSIS

Malay R. Mukerjee
Dept. of Computer and Communication Systems
Universiti Putra Malaysia
43400 Serdang, Selangor

ABSTRACT

Examines the computational efficiency of the master slave Multiple processor architectures system by considering a system consisting of a master M and p slave processors. The system performance is found by modelling it as a Markov process and a new method presented for computing the steady-state performance by dividing the state space into an interior and boundary space. The throughput of the system is then compared with that of a cost equivalent single processor using different values for the well-known Grosch parameter. It is demonstrated that the system is computationally efficient only for a sufficiently large number of jobs.

Keywords: ***Multiprocessor systems, Closed queuing systems***

1.0 INTRODUCTION

Distributed architectures for processing with a number of different processors in the system have become a popular form of organization in recent years. There is however a question as to whether such architectures fully utilize the resources committed for computation. An example is the master-slave architecture which has been used in a number of real-time computer control applications like robotic motion control, aircraft control and computer-aided manufacturing systems. From a computer architectural point of view, the system can be thought of as a combination of a sequential and a parallel system. Jobs requiring both kinds of operation are sequentially processed in the master and then divided into tasks, which are processed in parallel by the slaves. On completion of processing of all the tasks, the job is reassembled and sent back to the master, possibly for further processing. This is an example of a parallel, cyclic two stage system [2]. As an alternative to this architecture we may also design a more powerful single processor to process all the jobs. As is obvious, this will process a single job more efficiently, as the processing of a single job does not use the computation resources efficiently in the distributed architecture, where some of the processors are idle for extended periods of time. To improve the utilization, a number of jobs can be processed to produce a multijob, multitasking environment. The question still remains whether this can indeed perform the processing in a more cost-effective manner as opposed to a single machine

The answer to this question clearly requires the evaluation of the performance of the master-slave processor system. The cyclic two stage system has been studied extensively in the literature [3-5] and it is noted that performance computations are difficult due to enormous increases in the size of the state space. In the present paper, we address two tasks: first we confine our attention to the master-slave architecture and propose a computationally efficient method for performance calculation by reduction of the state space size. We then use this method to compute the performance for different numbers of jobs and provide an analysis of the cost effectiveness of the parallel architecture, which shows that the increase in complexity can be justified only if the number of jobs in the system is sufficiently large.

The paper is organized as follows: in Section 2, a model for the system is presented to facilitate the analysis. The new method to compute the steady state probabilities is outlined in Section 3. In Section 4 we compare this method with others from the point of view of complexity of the algorithm. The method used to compare the master-slave configuration with a single processor system is then presented in Section 5 and the results given in Section 6. Some concluding remarks are presented in Section 7. The text is illustrated throughout by an example taken from robot dynamics calculations.

2.0 SYSTEMS MODEL

The model of the multitasking master-slave processor system considered in this paper is shown in Fig 1. It consists of two sections: the master section M and the slave section S. The master section consists of a single processor designated as M and the slave section consists of p identical processing centers S_i, i = 1,2...p. The master is connected to the slave section through a **fork** node F such that a job passing through this node is split into q sibling tasks (in this paper we confine ourselves to q = p). Similarly the slave section is connected to the master through a **join** node J. This join node is an AND join node i.e. all the sibling tasks have to be completed before the job can be reassembled and sent back to M.

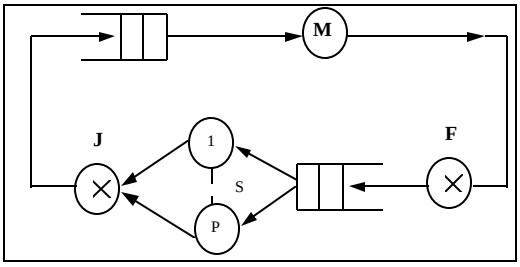

Fig. 1: Master Slave Architecture

The master-slave system can process work assigned to it by a combination of sequential and parallel processing. It is assumed that jobs assigned to it consist of parts that can be executed in parallel, with the parallel sections of the algorithm separated by sequential sections. All jobs arrive initially at the master M where an initial sequential section (if any) is processed. The job is then sent to F where it is split into tasks, which are assigned to available processors in the slave section. Any tasks for which processors are not available are queued in an FIFO queue at S. Upon completion of processing, a task arrives at J; if all its sibling tasks have been completed, it is routed back to the master; otherwise it waits at J for the remaining tasks to be completed. During the processing of the tasks in parallel, the master may begin the processing of the sequential section of another job and thus may not be available to the first job for processing, in which case the job waits at an FIFO queue at M for processing. On acquiring the processor, a further sequential section of the job is processed, possibly using the results of the previous parallel section, and the operation proceeds in this manner until the job is completed.

The processors are assumed to have an exponential service, the service rate being μ_M for the master and μ_S for each of the slaves. Let N be the total number of jobs being processed by the system and let N_M, N_S denote the number of jobs at the master and slave centers respectively (including the jobs/tasks in queue), with $N_M + N_S = N$.

A task completing service and arriving at J will have two possible situations:

1. Its remaining p-1 sibling tasks are already waiting at J after completion.
2. Some or all of them are still being processed in S or are in queue.

To denote such situations, we define the *job state* for a job A being processed in S as $\varphi_A = (l , m)$, where l is the number of tasks in A which have been completed and m is the number being processed. Then $p - l - m$ tasks are in queue at S. Clearly $l + m \le p$ and $m = 0$ implies $l = 0$. Also l can only assume values 0,1...p-1, since, as soon as $l = p$, the job is reassembled and sent back to M. Let $\varphi_1 , \varphi_2, ... \varphi_N$ be the job states of the N jobs and let $\Phi = (\varphi_1 , \varphi_2, ... \varphi_y)$ with $0 \le y \le p$ denoting the jobs with at least one task being served in S. The *system state* $\sigma = [\Phi, N_S, N_M]$. Clearly the number F of all the different possible Φ depends only on p and not on N as illustrated below.

Illustrative Example

To illustrate and clarify the procedure and its advantages, we apply the method to a case calculation of robot dynamics. Robot arm dynamics computations involve a lot of operations on small matrices related to co-ordinate transformations. A well-known method uses the Newton-Euler equations of motion to calculate the force or torque to be applied to each robot joint given a desired motion trajectory. Master-slave architectures have often been used for these computations with a number of processors, each processor being used to perform the computations for one of the joints or degrees of freedom. The details of the model are omitted here and can be found in **[6]** and accompanying references. Here we assume that the computations are to be carried out on a 3 processor system consisting of a master and two slave processors. Five separate robot joints are considered and the calculation of each is regarded as a separate job. The usual method is to model the job as a task graph [6], which divides each job into a number of sequential and parallel operations. For convenience we regard the taskgraphs of all the jobs as identical. In the terminology of our model, N=5 and p=q=2.

The values of Φ are $\Phi_1 = (0,2)$, $\Phi_2 = \{(1,1),(0,1)\}$, $\Phi_3 = \{(1,1),(1,1)\}$ and F = 3. As may be noted, the number F is not changed if N is changed say from 5 to 6, but it changes if p increases. The states are :

$\sigma_1 = [\Phi_1,0,4]$ $\sigma_2 = [\Phi_1,2,3]$ $\sigma_3 = [\Phi_1,4,2]$ $\sigma_4 = [\Phi_1,6,1]$ $\sigma_5 = [\Phi_1,8,0]$ $\sigma_6 = [\{(1,1)\}, 0,4]$ $\sigma_7 = [\Phi_2,1,3]$ $\sigma_8 = [\Phi_2,3,2]$ $\sigma_9 = [\Phi_2,5,1]$ $\sigma_{10} = [\Phi_2,7,0]$ $\sigma_{11} = [\Phi_3,0,3]$ $\sigma_{12} = [\Phi_3, 2,2]$ $\sigma_{13} = [\Phi_3,4,1]$ $\sigma_{14} = [\Phi_3,6,0]$

All transitions occurring in the system can be classified as one of the following types:

A. A change of state due to a task moving from S to J: if z_A is the number of jobs with a task in S and less than q-1 tasks in J, then rate of this is $z_A \mu_S$.
B. A change of state due to a task moving from S to M: if z_B is the number of jobs with q-1 tasks in J, then the rate of this is $z_B \mu_S$. Also $z_A + z_B = p = q$.
C. A change of state due to a job moving from M to the S queue (and splitting into q tasks): the rate of this is μ_M.

It may be noted that a B type transition increases the number of jobs in M by 1, a type C transition decreases it by 1 and a type A transition leaves it unchanged. For example, the example system moves from σ_8 to σ_9 by a type

C transition, splitting the job into 2 tasks. It also moves from σ_{13} to σ_8 by a type B transition; since there are two ways to do this, the rate is $2\mu_S$.

The system is then a homogeneous Markov process. The transition probabilities from one state to another are determined by which jobs are currently in M or S. It is assumed that only single event transitions occur, the probability of multiple events completing simultaneously being considered negligible. Let W denote the state space of dimension k, Q the one step Markov transition matrix of dimension kxk and let $\pi = [\pi_{\sigma 1}, \pi_{\sigma 2}, \ldots \pi_{\sigma k}]$ denote the steady state probabilities of the states σ_1, σ_2 etc. Since the transition matrix is easily verified to be irreducible, there exists a unique steady state probability given by the solution of the linear equations $\pi Q = \pi$, subject to the normalising condition $\sum \pi_i = 1$. The first step in determining performance is to calculate these steady state or equilibrium probabilities.

3.0 THE SOLUTION METHOD

After the transition matrix has been obtained, a solution can be obtained by different methods. One of these is the solution of the equations $\pi Q = \pi$ by standard methods of solving linear algebraic equations like the Gaussian elimination. Another method is the well-known Matrix Geometric method [7] which takes advantage of the sparsity of the matrix. This relies on the fact that the equation $\pi Q = \pi$ can be rewritten as $\pi P = 0$ where $P = Q - I$ and I is the identity matrix. It is readily shown that P has a block tridiagonal structure. The probability vector π can then be written as the sum of two matrix geometric terms plus a linear term; the matrix geometric terms can be found by solving for new matrices R' and R", which are solutions of two matrix quadratic equations. The time complexity of the MG method depends on the time to find these new matrices, which are usually calculated by iterative methods. The number of iterations increases with the spectral radius of R' and R" and the calculations become increasingly computation intensive.

We will hence introduce a different method for calculating π which is computationally more efficient. In order to do this, the state space W is partitioned into inner space W_i and boundary space W_b, defined as follows: $W = W_i + W_b$ where

> W_i is the set of all states such that for any state $\sigma \in W_i$ the number of jobs j_s in S satisfy $p < j_s < N$ and hence the number of jobs j_m in M satisfy $1 \le j_m \le N\text{-}p\text{-}1$.
> W_b is the set of all states such that $0 \le j_s \le p$ or $j_s = N$ and $j_m = 0$ or $N\text{-}p \le j_m \le N$,

The equilibrium probability vector can also be written as

$$\mathbf{p} = [\, \mathbf{p}_b(0), \mathbf{p}_i(1) \ldots\ldots \mathbf{p}_i(N\text{-}p\text{-}1), \mathbf{p}_b(N\text{-}p) \ldots\ldots \mathbf{p}_b(N)\,] \qquad (1)$$

where $\mathbf{p}_a(k)$ includes the equilibrium probabilities for all states with k jobs in M for a = i or b. Since any state σ can be identified by (Φ_i, k), one can write $\mathbf{p}_i(k) = [\pi_{\phi 1}, \pi_{\phi 2}, \ldots\ldots \pi_{\phi |F|}]$ as all |F| values of Φ are possible for any inner state. The same is true for $\mathbf{p}_b(0)$ and $\mathbf{p}_b(N\text{-}q)$. As can easily be verified, $\mathbf{p}_b(N)$ has order 1 and $\mathbf{p}_b(N\text{-}1)$ order q. The remaining subvectors $\mathbf{p}_b(N\text{-}2)$ to $\mathbf{p}_b(N\text{-}q+1)$ have orders increasing from q to |F| as enough tasks are not available to produce all values of Φ.

As illustration, we return to our example. Here the inner states are k=1,2 and all three values of Φ are seen to be valid, as they are for the boundary states k=0,3. However the type B transition from Φ_3 to Φ_2 is not possible for σ_{11} as the S queue is empty, thus making this value of k a boundary state. Though k=0 is a boundary state as a type C transition is not possible, it also has all the 3 values of Φ. For k=4, only Φ_1 and a form of Φ_2 (in σ_6) is possible and k = 5 has of course only one possible state with all jobs in M. We now note some properties of the system:

Property 1. The number F does not depend on N.

Property 2. The inner state space has cardinality $|W_i| = |F|(N\text{-}p\text{-}1)$ and the boundary space has $|W_b| \le (|F|+1)p + 1$, which is independent of N.

This is obvious as there are N-p-1 states and each has all the |F| values of Φ. Let σ_A be the set of states from which a given state σ may be entered by a type A transition and let x_A be the value of z_A for this. Similarly define σ_B, σ_C and x_B. Then

Property 3. All inner states σ with a common Φ_i must have the same global balance equation

$$\pi_\sigma(z_A\mu_S + z_B\mu_S + \mu_M) - \sum_{\sigma_A} \pi_{\sigma_A} x_A \mu_S - \pi_{\sigma_B} x_B \mu_S - \pi_{\sigma_C} \mu_M = 0 \qquad (2)$$

Proof. The first term in the above expression gives the probability of transiting *from* σ. and the second term that of transiting *into* σ., which must be equal in the steady state. All states have the same Φ and only differ in the number of N_S and N_M. Now since the N_S jobs are waiting in a queue, they cannot contribute to any transitions. Thus transitions from σ will be due only to the jobs in Φ and those in N_M. Since this is an inner state, N_M is at least 1 and thus one way of transiting from σ is at a rate μ_M by a C type transition. Further, since Φ is the same for all the states, ways of making A or B type transitions from the state must be identical with weights z_A and z_B respectively. Similarly for any inner state, the transitions of all types into the state are

possible, with weight 1 for a C type transition and x_A, x_B for A and B type transitions. The equation hence follows:
The equation (2) can be rewritten as a *common* equation for all inner states

$$\mu_M \pi_{\sigma_i}(k+1) = (z_{A_i}\mu_S + z_{B_i}\mu_S + \mu_M)\pi_{\sigma_i}(k) - \sum_{\sigma_A} x_{\sigma_A}\mu_S \pi_A(k) - x_{\sigma_B}\mu_S \pi_B(k-1) \quad (3)$$

with $A_i, B_i \in \{1,2....|F|\}$ and k = 1,2 ... N-p-1. The equilibrium probabilities $\pi_t(2)$ ... $\pi_t(N\text{-}p\text{-}1)$, $\pi_b(N\text{-}p)$ can now be obtained by solving the series of equations (3) as recurrence relations. This still leaves a set of W_b unknown probabilities $\pi_b(0)$, $\pi_t(1)$, $\pi_b(N\text{-}p+1)$ .....$\pi_b(N)$. These can be obtained by writing the global balance equations for all the states in W_b. This can be done by inspection and requires a limited amount of work as by definition W_b is independent of N.

Using our example for illustration, for Φ_2, the inner states are σ_8 and σ_9. For both $z_A = z_B = 1$. σ_8 can be entered by a type A transition from σ_3, a type B transition from σ_{13} and a type C transition from σ_7 and $x_A = x_B = 2$. Exactly the same is true for σ_9 with the indices of states increased by 1. Hence (2) becomes

$$\pi_{\sigma 8}[\mu_M + \mu_S.1 + \mu_S.1] - \pi_{\sigma 7}\mu_M - \pi_{\sigma 3}\mu_S.2 - \pi_{\sigma 13}\mu_S.2 = 0$$

and is a generic equation for all σ with Φ_2. The states can be entered by a C transition from another state with Φ_2, an A transition from a Φ_1 state or a B transition from a Φ_3 state. Thus (3) is

$$\mu_M \pi_{\phi 2}(k+1) = [2\mu_S + \mu_M]\pi_{\phi 2}(k) - 2\mu_S \pi_{\phi 1}(k) - 2\mu_S \pi_{\phi 3}(k-1) \qquad k=1,2$$

Similar equations can be written for Φ_1 and Φ_3 for the inner states. We can now combine these by defining a vector $X(k) = [\pi_{\phi 1}(k), \pi_{\phi 2}(k), \pi_{\phi 3}(k), \pi_{\phi 3}(k-1), \pi_{\phi 2}(k-1)]$. Then we have the recurrence relation

$$X(k+1) = X(k)\,C \qquad k=1,2$$

and the matrix C is

$$C = \begin{bmatrix} 2\mu_S+\mu_M & -2\mu_S & 0 & 0 & 0 \\ 0 & 2\mu_S+\mu_M & -\mu_S & 0 & 0 \\ 0 & 0 & 2\mu_S+\mu_M & 0 & 0 \\ -\mu_S & 0 & 0 & \mu_S & 0 \\ 0 & -2\mu_S & 0 & 0 & \mu_S \end{bmatrix}$$

Thus the steady state probabilities can be expressed as $X(k) = X(1)\,C^{k-1}$ for k = 2,3. The remaining steady state probabilities are found by writing the equations for the boundary states, e.g. for k=4

$$\mu_M \pi(5) = \mu_S \pi_{\sigma 6} - 2\mu_S \pi_{\sigma 1} - 2\mu_S \pi_{\sigma 11}$$

These equations are limited in number and can be written by inspection. Thus all the steady state probabilities are easily available in terms of X(1) and the final values can be obtained by using the normalizing equation.

The complete details for the example are given in Appendix 1. A similar procedure can be followed for any other example with a different set of states and a different matrix C. We restate the solution procedure in the form of an algorithm.

ALGORITHM 1

1. Generate the set F of all the Φ for the specified value of p.
2. Obtain the equation set (3) for each value of Φ for all inner states.
3. Determine the matrix C.
4. Complete the set of equations by obtaining the equations for the boundary states by inspection.
5. Obtain all the inner state X(k) in terms of X(1) and find X(1) by using the boundary state equations and the normalizing equation.
6. Substitute values of μ_M and μ_S to obtain numerical values of all probabilities.

4.0 COMPLEXITY ANALYSIS

The dimension of the matrix Q is O(N)xO(N) asymptotically and solving the equation $\pi Q = \pi$ by the Gauss elimination method would cost $O(N^3)$. By taking advantage of the banded nature of the matrix, the cost can be reduced to O(N). However this is a numerical solution method and hence unsuitable for parametric studies.

The Matrix Geometric method requires the computation of the matrices R and R', which may be obtained by an iterative procedure. Each step of this procedure is $O(|F|^3)$ and the convergence rate depends on the spectral radius of R and R' and requires from $O(|F|^{1/3})$ to O(|F|) steps. The computation of the probability vector requires O(N) steps, each requiring the product of the |F| order matrix R by π. Thus the total cost is $O(|F|^4) + O(N|F|^2)$.

The present method incurs heavy computation only in Step 5 of the algorithm and requires the product of C by the vector **p**(k). The matrix C is of order |F| and is independent of N and can be easily banded, so that the cost of a single step is O(|F|) and, since O(N) steps have to be performed, the cost is O(N|F|) and thus linear in N. However F increases rapidly with p, a problem also faced by the above two methods. An important advantage of this method is that up to Step 5 of the algorithm all values are expressed symbolically and numerical values are calculated only in Step 6. This factor increases its advantage in parametric studies, as opposed to the purely numerical methods.

Moreover, once a solution has been obtained for a given N, the solution for another N' can be obtained only by O (N'-N) operations.

5.0 COMPUTATION OF PERFORMANCE OF MASTER-SLAVE ARCHITECTURE

As explained in Section 1, the method outlined above is used extensively to calculate the steady state probabilities and thus the throughput of the system for different values of the total number of jobs N. The primary purpose of this is to evaluate the efficiency of the system as opposed to a single processor from the point of view of computation. The method adopted here will be to compare two systems, i.e. a single and a multiprocessor one, which are equivalent in terms of cost and evaluate the relative gain in processing power by using multiple processing units.

An empirical law proposed by Grosch in the 1940s stated that the cost c of computers increased as

$$c = t P^{a} \quad (4)$$

where t is a constant, P is the processing power of the computer and *a* is the Grosch parameter. The formula has been widely used with different measures like memory capacity, instruction cycles, MIPs etc. being used for P. A value of 0.5 for *a* was projected, indicating that computer systems are subject to economies of scale i.e. doubling the power would cost less than double. More recently [8] *a*=1 has been suggested as valid for modern distributed architectures. Thus it is suggested that a computer design based on a single powerful processor is no longer valid due to economies of scale. In essence, there exists a limit to the amount of computing power that can be compressed into a single chip due to limited speed of electronic signals and resolution of integration processes. The more the design moves towards these limits, the more the value of *a* increases. One way to counteract this effect would be to distribute the computing power between a number of processors as in the master-slave architecture, and a parametric analysis of such systems will demonstrate the extent to which such action would be successful.

Let Z be the average number of operations required at the processing center S; then average operations from each processor will be Z/q. Let P(1) and P(q) denote average processing power for a single and q processor unit and let μ and μ‘ denote average operating rates. and c and c' the costs of the two systems. One has

$$\mu = P(1)/Z \quad (5)$$

for the single processor, denoted System A and

$$\mu' = g(q)\ P(q)/\ (Z/q) \quad (6)$$

for the p processor system, denoted System B. Here $0 < g < 1$ is a factor to take into account the processing rate degradation due to parallel operation.

For the single processor system to be at an advantage, c < q c'. Substituting the Grosch equation, one has, after some algebra,

$$\mu' = q^{(a-1)/a}\, \mu\, g(q) \quad (7)$$

for the two system costs to be equal. Thus μ‘, the rate of processing for the parallel processor system, is seen to be a function of the Grosch parameter *a* nd the level of parallelism q in order to provide the same overall cost as the single processor system.

Table 1 shows the values of μ‘ computed for different values of a using (7) for q = 2, μ being taken as 10. The value of μ‘ is always equal to 10 for all q for a = 1 and the change is clearly much faster for higher values of q.

We will now take the two systems with arbitrary μ and μ‘ obtained as above so that they are cost equivalent and compare their performance. The performance measure is chosen here to be the system throughput or the number of jobs executed by the system in unit time. Throughput is null if the system runs no jobs (N=0) and increases as N is increased, eventually reaching to a bottleneck saturation.

Table 1: Values of a and u‘ for q = 2

a	μ‘
0.1	0.020
0.3	1.98
0.5	5.00
0.7	7.42
0.9	9.26
1.0	10.00
1.2	11.23
1.5	12.60
2.0	14.14

To examine the effect of the parameters *a,* q and N on the performance, g has been taken as 1 in the present study. The analysis can be modified easily if values of g<1 are to be used. To compare the throughputs of the two systems correctly, it must be remembered that the units being processed in a single processor system are jobs, whereas those in the multiprocessor one are tasks. In other words, since each job is split into q tasks for being processed in parallel, the completion of a job is equivalent to that of the completion of q tasks. This is taken into account by dividing the productivity of the processors in System B by q before comparing with that of System A. The analysis for the systems is now done as indicated in the following algorithm.

ALGORITHM 2

1. Set the value of N.
2. Set μ = 10 (an arbitrarily chosen value) and calculate the throughput T for System A by conventional methods **[3].** To ensure that the slave processor does not become a bottleneck node, μ_M is taken as k where k is the number of jobs at the master.
3. Set the value of *a* and q and calculate μ‘ to make the cost of the two systems equal.
4. For this value of *a*, q, and N, calculate all steady-state probabilities by Algorithm 1.
5. Compute the system throughput T’ from the steady state probabilities by conventional methods, e.g. that given in **[3]**.
6. Repeat steps **3-5** for increasing values of *a.*
7. Repeat steps **3-6** for different values of q.
8. Repeat steps **2-7** for different values of N.

In the above exercise, the value of μ_S is taken as μ‘ and μ_M taken as explained.

The master-slave processor forms a closed queuing network and various parameters like state probabilities, waiting time, throughput etc. can be calculated by the standard methods for such networks like the load balance method or mean value analysis. It is also possible to calculate the state probabilities and obtain the other parameters directly from these. Example computations are outlined in Appendix 2. As is well-known, the throughput is limited by a node with greatest service demand, which may be regarded as a ‘bottleneck’. The parameters are selected here to make sure that the parallel processor does not become the bottleneck.

6.0 RESULTS AND DISCUSSIONS

The system throughput of both Systems A and B are shown plotted against the value of the Grosch parameter *a* in Fig. 2 for different values of q and N. Only the values for q = 2, the lowest degree of parallelism, are shown for N =2,5,10 and 20, as well as the results for q=3 for N=2. The conclusions obtained from this are readily extended to cases of higher values of q.

The result confirms that as a is increased from values below 1 to those above 1, greater performance can be obtained by using a distributed architecture without any additional cost.

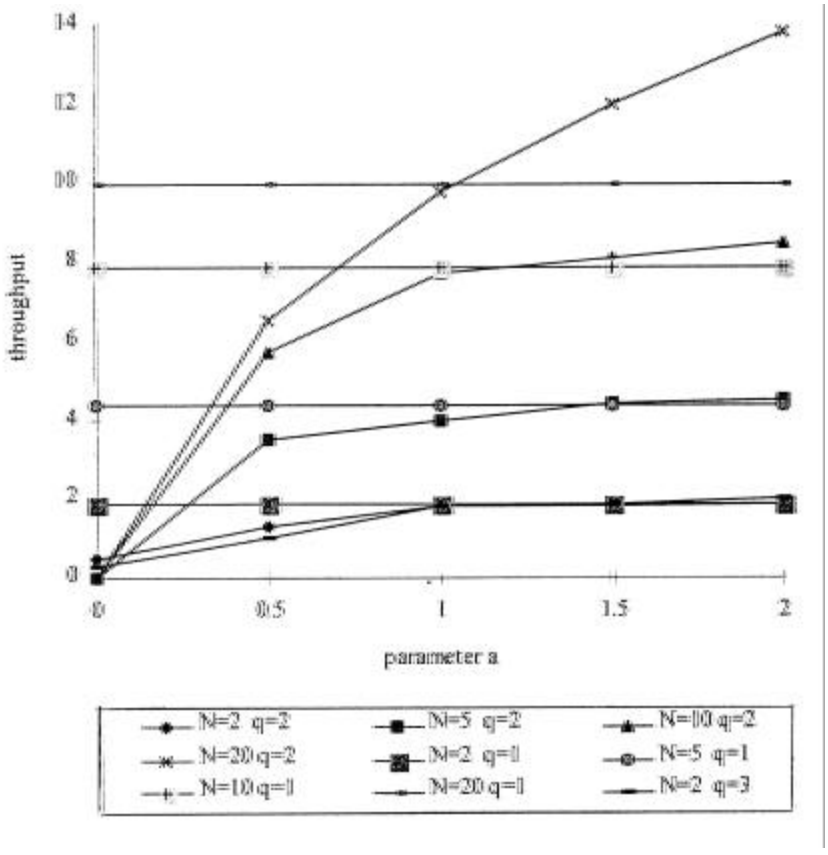

Fig. 2: Grosch Parameter vs Throughput

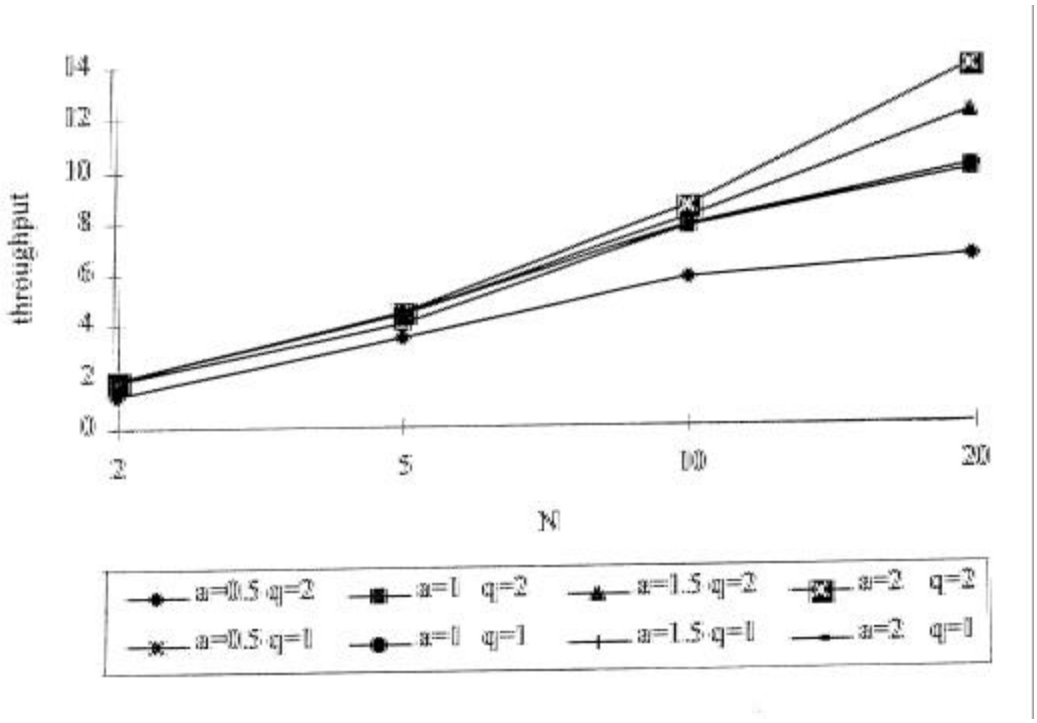

Fig. 3: Jobs vs Throughput

From Fig. 2, we have the values of throughput as 1.80, 4.46, 7.85 and 9.98 for N = 2, 5, 10 and 20 respectively for System A. This indicates of course that bottleneck effects are increasingly coming into play as N increases. At a = 0.5 parallelism is seen to be far from a useful alternative for any N. However it becomes increasingly more attractive as soon as a increases. Interestingly, the value of a^* seems to decrease with N. $a^*(2) = 2$, $a^*(5) = 1.3$, $a^*(10) = 1.03$ and $a^*(20) = 1.0$, implying that a^* tends to be around 1 for large N. On the other hand, this can also be interpreted to mean that, for any given a, provided it is large enough, the number of jobs has to be large to make parallelism computationally attractive. For the current postulated value of a = 1, the number of jobs is seen to be more than 10 to make the master-slave architecture competitive.

Fig. 3 shows a graph drawn between different values of N and the throughput; this essentially complements the information in Fig. 2. The data values for different N and a are given in Table 2 for q=2. System A calculations are given in Appendix 2.

The introduction of a more realistic value of g (<1) would make this requirement even stronger and still larger N would be required to make the parallel architecture efficient.

An interesting insight into the matter may be obtained by a study of Table 1. It is seen there that the power of the parallel processor for the same cost as the uniprocessor, as given by its rate of processing, falls off rapidly as a becomes less than 1 whereas for a=1 the two have the same processing power. (It may be recalled that μ = 10 in the above calculations of Table 1). This clarifies the reason for the lack of advantage to the multiprocessor system for values of $a < 1$.

Table 2: Throughput values for different N and a for q=2

		N		
a	**2**	**5**	**10**	**20**
0.5	**1.30**	**3.5**	**5.7**	**6.5**
1.0	**1.75**	**4.0**	**7.7**	**9.8**
1.5	**1.84**	**4.6**	**8.1**	**12.0**
2.0	**1.86**	**4.7**	**8.5**	**13.8**

7.0 CONCLUSIONS

In the above paper we have presented a method for evaluating the performance of a distributed processor system, the master-slave architecture, from the point of view of computational efficiency. Incidentally a method for calculating the steady state probabilities has also been presented, which is seen to be particularly useful for calculations when the parameter values are to be varied, as opposed to purely numerical methods. The advantage lies in the ability of the proposed method to use symbolic

representations over most of the steps of the algorithm so that different numerical values may be introduced only at the last step. This method has been used to compute the performance of the master-slave system for a range of parameter values and compare its throughput with that of a cost-equivalent single processor system to evaluate any improvement. It has been seen that the advantage can lie with the multiprocessor system only if computer sizes do not respond to economies of scale, as is widely believed now, and further if the number of jobs being processed by the system is sufficiently large. It may be noted that these conclusions are primarily based on the utilization of the processing powers of the systems concerned. Other considerations, like that of speed of processing the job by making use of more than one processor, have not formed the basis of this study, though they may of course play an important part in deciding the form of the system architecture to be adopted.

ACKNOWLEDGMENT

The author would like to acknowledge the support of Universiti Pertanian Malaysia through their research grant no. 50208-95-03.

REFERENCES

[1] F. Baccelli, W. A. Massey and D. Towsley, "Acyclic fork-join queuing networks", JACM, Vol. 36(3), 1989, pp. 615-42.

[2] A. Duda, "Approximate performance analysis of parallel systems", in G. Iazeolla, P. J. Courtois and O. J. Boxma(ed) *Computer Performance and Reliability*, North-Holland (Amsterdam), 1988.

[3] E. D. Lazowska, J. Zahorjan, G. S. Graham and K. C. Sevcik, *Quantitative System Performance: Computer System Analysis Using Queuing Network Models*, Prentice Hall, New Jersey, 1984.

[4] G. Iazeolla, "The complexity of performance analysis of parallel algorithms and systems", in A. Monacoa nd R. Negrini (ed), *Proc. IEEE Compuero 91, 5th Annual European Conf, May 1991*.

[5] R. Nelson and A. N. Tantawi, "Approximate analysis of fork/join synchronization in parallel queues", *IEEE Trans Software Engg*, Vol. 37(6), 1988, pp. 739-43.

[6] H. Kasahara, " Parallel processing of robot control and simulation", in S. G. Tzafestas(ed) *Microprocessors in Robotic and Manufacturing Systems*, Kluwer Academic (Boston), 1991.

[7] L. Gun and A. M. Makowski, "Matrix-geometric solutions for finite capacity queues with phase type distribution," in P. J. Courtois and G. Larouche(ed), *Proc. 12th IFIP WG7.3 Int Symp on Comp Perf Modelling, Measurement and Evaluation, Brussels, Dec 1987*, North-Holland (Amsterdam), 1988.

[8] H. Mendelson, "Economies of scale in computing: Grosch's law revisited," *Comm ACM*, Vol. 30(12), 1987.

APPENDIX 1

Here we briefly describe the equations obtained for our illustrative example. For Φ_1 the inner states are σ_3 and σ_4 and the equations are

$\pi_{\sigma 3}[\mu_M + \mu_S.2] - \pi_{\sigma 2}\mu_M - \pi_{\sigma 9}\mu_S = 0$

$\pi_{\sigma 4}[\mu_M + \mu_S.2] - \pi_{\sigma 3}\mu_M - \pi_{\sigma 10}\mu_S = 0$

as no Type B transition is possible from this state and it cannot be entered by a Type A transition. For Φ_2 the inner states are σ_8 and σ_9 and the equations are

$\pi_{\sigma 8}[\mu_M + \mu_S.1 + \mu_S.1] - \pi_{\sigma 7}\mu_M - \pi_{\sigma 3}\mu_S.2 - \pi_{\sigma 13}\mu_S.2 = 0$

$\pi_{\sigma 9}[\mu_M + \mu_S.1 + \mu_S.1] - \pi_{\sigma 8}\mu_M - \pi_{\sigma 4}\mu_S.2 - \pi_{\sigma 14}\mu_S.2 = 0$

and finally for Φ_3, the inner states are σ_{12} and σ_{13} and the equations are

$\pi_{\sigma 12}[\mu_M + \mu_S.2] - \pi_{\sigma 11}\mu_M - \pi_{\sigma 8}\mu_S = 0$

$\pi_{\sigma 13}[\mu_M + \mu_S.2] - \pi_{\sigma 12}\mu_M - \pi_{\sigma 9}\mu_S = 0$

as a Type A transition is not possible from this state and it cannot be entered by a type B transition.

This is rewritten as

$\mu_M \pi_{\phi 1}(k+1) = [2\mu_S + \mu_M]\pi_{\phi 1}(k) - \mu_S \pi_{\phi 2}(k-1)$ $k=1,2$

$\mu_M \pi_{\phi 2}(k+1) = [2\mu_S + \mu_M]\pi_{\phi 2}(k) - 2\mu_S \pi_{\phi 1}(k) - 2\mu_S \pi_{\phi 3}(k-1)$ $k=1,2$

$\mu_M \pi_{\phi 3}(k+1) = [2\mu_S + \mu_M]\pi_{\phi 3}(k) - \mu_S \pi_{\phi 2}(k)$ $k=1,2$

leading to the matrix C given above after augmenting with two trivial equations to make it square.

The boundary equations are written as

$\mu_M \pi(5) = 2\mu_S \pi_{\phi 1}(4) - \mu_S \pi_{\phi 2}(3)$

$\mu_M \pi_{\phi 1}(4) = [2\mu_S + \mu_M]\pi_{\phi 1}(3) - \mu_S \pi_{\phi 2}(2)$

$\mu_M \pi_{\phi 1}(1) = 2\mu_S \pi_{\phi 1}(0)$

$\mu_M \pi_{\phi 2}(4) = [2\mu_S + \mu_M]\pi_{\phi 2}(3) - 2\mu_S \pi_{\phi 1}(3) - 2\mu_S \pi_{\phi 3}(2)$

$\mu_M \pi_{\phi 2}(1) = 2\mu_S \pi_{\phi 2}(0) - 2\mu_S \pi_{\phi 1}(0)$

$\mu_M \pi_{\phi 3}(1) = 2\mu_S \pi_{\phi 3}(0) - \mu_S \pi_{\phi 2}(0)$

The solution then proceeds as indicated in the algorithm

APPENDIX 2

The procedure for System A is first outlined for various values of N starting with 2.
For N=2, the states are (0,2), (1,1) and (2,0) where the numbers refer to jobs in the master and slave respectively.
The steady state equations are readily found to be
$10\,\pi(0) = \pi(1)$
$5\,\pi(1) = \pi(2)$
and the normalising equation
$\pi(0) + \pi(1) + \pi(2) = 1$

Here the number in parentheses refers to the number of jobs at the master. Solving $\pi(0) = 1/61$ and this gives the throughput 110/61 or nearly 1.8.

For N= 5, the state equations are
$10\,\pi(0) = \pi(1)$; $5\,\pi(1) = \pi(2)$;
$10\,\pi(2) = 3\,\pi(3)$; $2.5\,\pi(3) = \pi(4)$;
and $2\pi(4) = \pi(5)$
together with the normalising equation
$\pi(0) + \pi(1) + \pi(2) + \pi(3) + \pi(4) + \pi(5) = 1$
giving $\pi(0) = 3/4330$ and throughput = 4.46.

We similarly find throughput for N = 10 as 7.8 and for N = 20 as 9.98.

For System B, the solution has to be done by the procedure of Algorithm 2. Only the case for N = 2 is illustrated. The job states are ϕ_1 (0,2), ϕ_2 {(1,1),(0,1)}, ϕ_3{(1,1),(1,1)} and states are $\sigma_1 = \{\phi_1,0,1\}$, $\sigma_2 = \{\phi_1,2,0\}$, $\sigma_3 = \{(1,1),0,1\}$, $\sigma_4 = \{\phi_2,1,0\}$, $\sigma_5 = \{\phi_3,0,0\}$

We can write the state balance equations
$2\,\pi(2) + k\,\pi_2(0) - (2k+1)\,\pi_1(1) = 0$
$\pi_1(1) - 2k\,\pi_1(0) = 0$
$2k\,\pi_1(1) + 2k\,\pi_3(0) - (1+k)\,\pi_2(1) = 0$
$2k\,\pi_1(0) + \pi_2(1) - 2k\,\pi_2(0) = 0$
$k\,\pi_2(0) - 2k\,\pi_3(0) = 0$
together with the normalising equation
$\pi(2) + \pi_1(0) + \pi_1(1) + \pi_2(0) + \pi_2(1) + \pi_3(0) = 1$

Here the subscripts 1, 2, 3 refer to states with ϕ_1,ϕ_2,ϕ_3 respectively and $k = \mu_S/\mu_M$. As indicated above, the symbolic equations can be solved in terms of the parameter k. After simplifying we get
$(8k^3 + 24k^2 + 34k + 3)\,\pi_1(0) = 6k$

At this point numerical values have to be substituted to get various system parameters. As an example we see for a = 1, $\mu' = \mu_S = 10$. Substituting we get the throughput to be 18840/10743 = 1.753. Similarly for $\mu' = \mu_S = 14$, the throughput is found to be 49448/26706 = 1.85.

Calculation for other values of N are similar though more complex due to an increased number of balance equations.

BIOGRAPHY

Malay R. Mukerjee obtained his DIISc from Indian Institute of Science, Bangalore, MASc from University of British Columbia, Canada and Ph.D. from University of Roorkee, India, all in Electrical Engineering. He has a teaching experience of more than 30 years in India, Canada, Iraq and Malaysia. He is currently Professor in the Department of Computer & Communications Systems, Faculty of Engineering, University Putra Malaysia. He has published more than 60 papers in different national and international journals and conference proceedings. His current research interest includes computer & communication networks, communication protocols, distributed computer systems and digital switching.