5-1997

The Circulating Processor Model of Parallel Systems

Amy Apon
Clemson University, aapon@clemson.edu

Lawrence Dowdy
Vanderbilt University

The Circulating Processor Model of Parallel Systems*

Amy W. Apon and Lawrence W. Dowdy
Department of Computer Science
Vanderbilt University
Nashville, TN 37253
email: [apon,lwd]@vuse.vanderbilt.edu

Abstract

This paper introduces the circulating processor model for parallel computer systems. The circulating processor model is a product form queueing network model where the processors are allowed to circulate between the parallel applications instead of the more traditional circulating task model. Certain behaviors of parallel systems are better captured using this new approach. The circulating processor model may be load dependent or load independent. The load dependent circulating processor model is exact for systems which contain a single parallel application. An exact error is calculated for the load independent circulating processor model for systems which contain a single parallel application. The load dependent circulating processor model is a good approximation to the actual system in the case of multiple parallel applications. The load dependent circulating processor model compares favorably to the traditional circulating task model.

Index Terms: circulating processor model, load dependent model, parallel systems, performance evaluation, product form queueing network

*This work was partially supported by sub-contract 19X-SL131V from the Oak Ridge National Laboratory managed by Martin Marietta Energy Systems, Inc. for the U.S. Department of Energy under contract no. DE-AC05-84OR21400.

Contents

1. Introduction

The design and development of parallel computer systems is an important research topic for the scientific community. Analytic models of these systems can be an important tool for evaluating various design alternatives. Analytic models of these systems can be divided into state space based models and non-state space based models. State space based models, such as Markov models and Petri net models [1], enumerate all possible states that the systems may enter. While more accurate, the computational complexity becomes intractable as the size of the system and workload increases. Non-state space based models, such as product form queueing network models, do not enumerate or evaluate the steady state probability of individual states in the underlying Markov diagram of the system. Rather, computationally efficient techniques are used to determine mean performance metrics of the system [2]. These traditional product form queueing network models, however, do not capture behaviors such as task forking and joining, parallel gang scheduling, and barrier synchronization which are common in parallel computer systems.

The focus of this paper is on the development of an alternate computationally efficient queueing network model. The goal is to retain the computational efficiency of product form models, while accurately modeling issues such as forking and joining of parallel tasks, gang scheduling, and barrier synchronization. The circulating processor (CP) model is introduced. The CP model is a product form queueing network model which captures certain behaviors exhibited by complex parallel computer systems, while maintaining computational efficiency. Related work includes other models of parallel systems with computationally efficient solution techniques [3–8].

Section 2 introduces the circulating processor model and the types of systems to which this model can be applied. Section 3 gives the analysis of the CP model when one parallel application is in the system. Two versions of the CP model are presented: a load dependent CP model and a load independent CP model. Section 4 gives the analysis of the CP model when multiple parallel applications are in the system. In this case, the processor scheduling policy must be defined. Two scheduling policies are analyzed: a fully parallel scheduling policy and a fully sequential scheduling policy. Section 5 compares the CP model with a more traditional circulating task model. Section 6 presents an application of the model to an actual parallel system. Section 7 gives conclusions and future work on the circulating processor model of parallel systems.

2. The Circulating Processor Model

2.1 Overview

In a traditional queueing network model of a computer system, the processors are represented by service centers, and the parallel applications (i.e., jobs, tasks, workload) are represented by customers which circulate among the service centers. In a parallel application, the application splits and thus "visits" (i.e., is assigned to) more than one processor at a time. This simultaneous resource possession behavior is not allowed in product form models. In the fork-join application, however, each processor only "visits" (i.e., is assigned to) one parallel application at a time. An alternate method of modeling the system is to represent the parallel applications by the service centers in a queueing network model and the processors as the customers which circulate.

A number of observations make this model intuitively interesting and feasible. First, the number of processors in the system is a known quantity. This corresponds to a closed queueing network model with a fixed multiprogramming level. Second, most parallel systems are homogeneous, where all of the processors are identical. When viewed as customers in a queueing network model, these "customers" are all statistically identical, or single-class. Third, when the parallel applications are different, the traditional model which represents the parallel applications as the circulating customers is a multiclass model with heterogeneous customers circulating among homogeneous servers. When this is combined with the modeling of the non-product form fork-join behavior of the parallel applications, the model is often too complex to solve exactly in all but the most simple cases. With a circulating processor model, each application is represented by a different service center, which is parameterized independently. This allows homogeneous single class customers to circulate among heterogeneous servers. Even when the applications are different, as long as the processors are the same, the model is single class and and has an efficient product form solution.

The circulating processor model gives an alternate method of gathering performance metrics of parallel systems. The mean queue length of a service center in the circulating processor model represents the mean number of processors assigned to the parallel application, which is equivalent to the average parallelism of the corresponding application. The sum of the queue lengths of the service centers in the circulating processor model is equivalent to the mean number of utilized processors in the parallel system. The optimal branching probabilities, which can be calculated for the circulating processor model, are associated with the optimal partition sizes for allocation of processors in a parallel computing system.

Traditional performance metrics such as throughput of parallel applications and processor utilizations can also be calculated from the circulating processor model. However, they are somewhat more complex to visualize intuitively. For example, traditional throughput using the circulating processor model may be calculated as the number of processors which complete at the application per unit time, weighted by the expected number of processors used by the application (i.e., the average parallelism of the application). Processor utilization can be calculated by first finding the processor idle time, which is the mean wait time in the idle server of the CP model (i.e., where processors visit when they are not assigned to any application).

The circulating processor model is parameterized using the parallelism shapes of the parallel applications [9]. The parallelism shape gives the proportion of time that an application spends using each number of processors. By using the parallelism shape of a parallel application, the service time of the corresponding queueing network model server is load dependent. In this case, the mean service time of an application depends on the number of processors present at its server (i.e., currently assigned to the application). This load dependent service time is equal to the amount of time that the parallel application actually uses each number of processors, given that all processors are available to the application. The CP model may also be parameterized using load independent servers, calculated from the average amount of time that a processor spends at the application. In either case, the circulating processor model will have the same number of underlying states, with the same connections between states, but the flow rates between the states will be different.

The circulating processor model is still an approximation of certain parallel behaviors. In a simple fork-join system, a parallel application arrives at a fork point and forks into some number of parallel tasks. Each of these tasks may be assigned to different processors at exactly the same time. If many processors are idle at the time of a fork instruction, then many processors will begin executing a parallel task at the same time. Viewed from the circulating processor perspective, several processors leave the idle server and “arrive” in bulk at the parallel application server. This state-dependent service and routing behavior in the actual system in not captured in either the traditional models or in the CP model, and is a potential source of error.

2.2 Target System Assumptions

The target system to be modeled is a parallel system in which several parallel applications execute. As soon as one parallel application completes, another begins, so that the system can be viewed as having a constant number of executing applications. The target system is a multiprocessor with N identical processors. Various other assumptions may be made about the target system.

1. An additional delay server may be added to the model to represent terminal users submitting their parallel applications to the multiprocessor.

2. The queueing discipline at the multiprocessor may be specified (e.g., FCFS, round-robin).

3. The service distribution at the multiprocessor may be specified (e.g., exponential, Coxian).

4. When the number of parallel applications executing in the system is larger than one, then the scheduling policy at the multiprocessor may be specified to be either fully parallel or fully sequential [10]. For fully parallel scheduling, a parallel application arrives at the multiprocessor and forks into N tasks, each of which is scheduled on a unique processor. In this case, all applications have the same number of tasks and the same fork-join task graph, but the service demands for various tasks may be different. For fully sequential scheduling, all tasks of the application are scheduled onto a single processor. In this case, the number of tasks and the task graphs of the applications may be different. In either case, as each task completes, it arrives at a barrier synchronization point where it waits for all of its siblings to complete. When all sibling tasks have completed execution at the multiprocessor, the synchronization operation is completed, and the parallel application continues to circulate in the closed queueing network model.

5. An application may release each assigned processor as soon as the tasks which have been assigned to it completes. Because siblings finish at different times, this allows tasks of other applications to begin execution immediately on the released processor.

Due to the parallel synchronization behavior, an exact model would not have a product form solution.

As a simple example of a target system, consider the system illustrated in Figure 1. The multiprocessor consists of three processors, and terminal activity is modeled as a delay server. Assume that parallel application A executes on this system and has a task graph as shown in Figure 2. Upon arriving at the multiprocessor, application A forks into three parallel tasks, A_1, A_2, and A_3, which execute concurrently on the three processors. Each parallel task of application A is assigned to a distinct processor of the system. The mean service time, or loading, of task A_i is designated a_i. As the tasks complete at the multiprocessor, a barrier synchronization forces all tasks to wait until the completion of the final task before application A returns to visit the delay server. The application executes at the delay server for an exponentially distributed amount of service time, designated a_0. Several applications, A, B, ..., may be in the system at the same time, so that the multiprogramming level may be any positive integer.

3. Analysis with a Single Parallel Application

Consider the case when there is: 1) a single parallel application executing in the system, 2) a single terminal delay server in addition to the multiprocessor, and 3) N tasks of the application, each of which is assigned to a unique processor. Figure 3 shows the Markov state space diagram for the

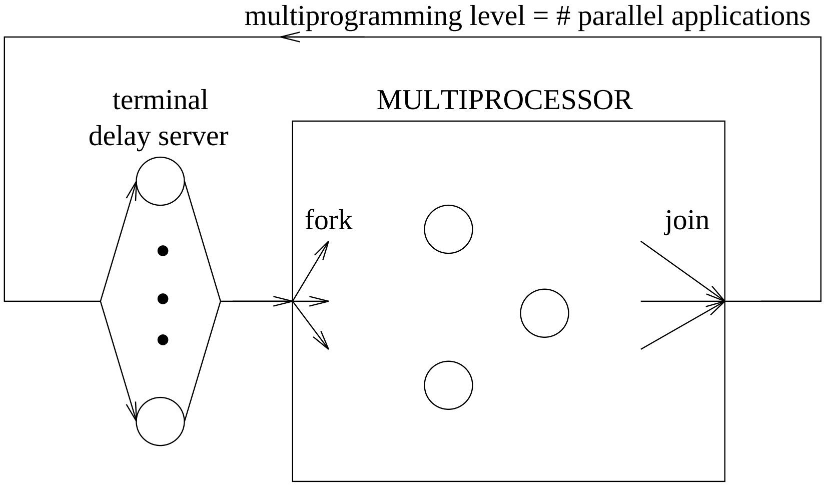


Figure 1: Example Target System

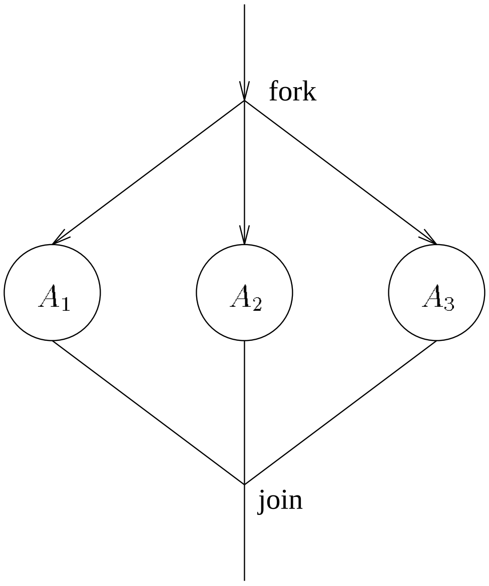


Figure 2: Task Graph for Parallel Application A

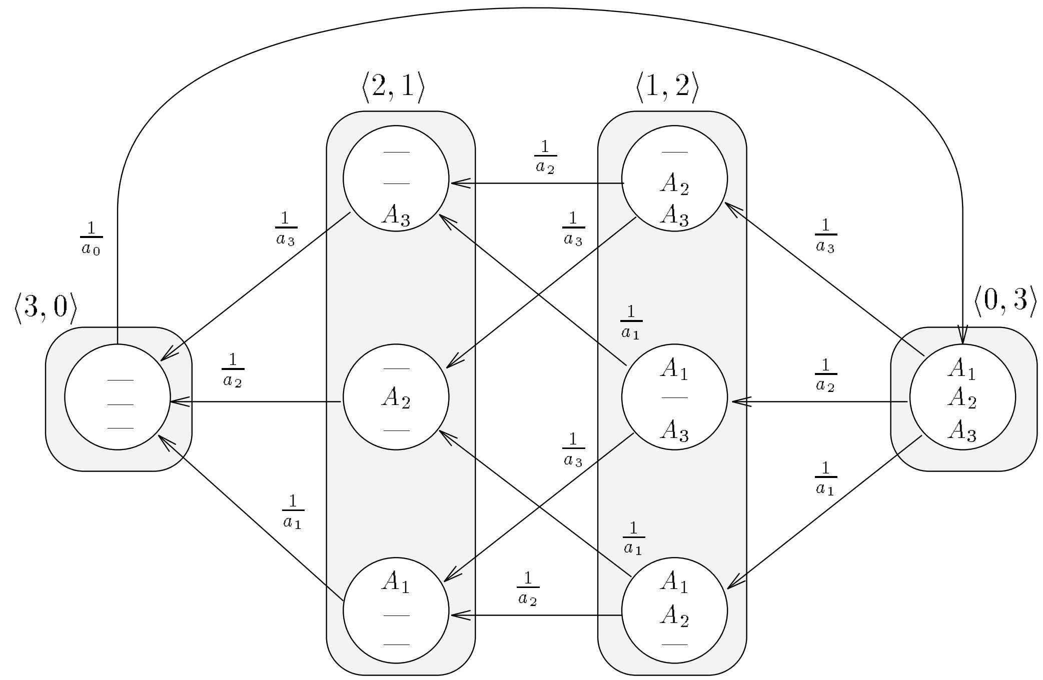


Figure 3: Markov Diagram for Actual System: 3 Processors, 1 Parallel Application

single parallel application running on the target system in which the multiprocessor contains three processors and the service times at the multiprocessor are exponentially distributed. The states are labeled with the tasks that are active during that state. No task is active at the multiprocessor when parallel application A is at the terminal delay server. Tasks A_1, A_2, and A_3 are all active when the program first arrives at the multiprocessor, just after the fork point. The tasks complete one at a time, in some order, until the application completes execution at the multiprocessor and returns to the delay server. The performance of the actual system is calculated based on the steady state probabilities of the eight possible system states shown in Figure 3. Let the states of the Markov diagram of the actual target system be denoted by $\langle\ \rangle$. For example, $\left\langle \begin{smallmatrix} - \\ - \\ A_3 \end{smallmatrix} \right\rangle$ denotes the state in which only task A_3 is active at the multiprocessor. This implies that tasks A_1 and A_2 have completed and are waiting at the synchronization join point.

States in which there are an equal number of tasks active at the multiprocessor are enclosed in a shaded box. Let $\langle n_I, n_A \rangle_{\text{actual}}$ denote the state(s) where n_I is the number of idle processors, and n_A is the number of processors actively executing a task. Where no confusion arises, the subscript will be dropped. This alternate view of the state space leads to a model in which the states are described by the number of processors currently executing tasks of the parallel application (i.e., a model in which the processors circulate between being active, working on an application's tasks, and being idle, awaiting the arrival of a new application).

3.1 Load Dependent Circulating Processor Model

The load dependent circulating processor model for the example system in Figure 1 with one parallel application is illustrated in Figure 4. The number of service centers is equal to the number of parallel applications plus an additional service center, denoted the *idle server*. This server models the idle time that each processor experiences between assignments to an application. Since there is

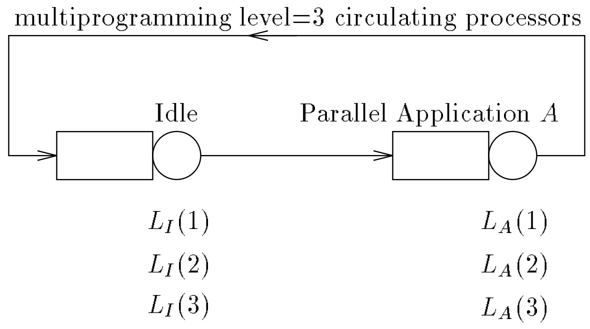
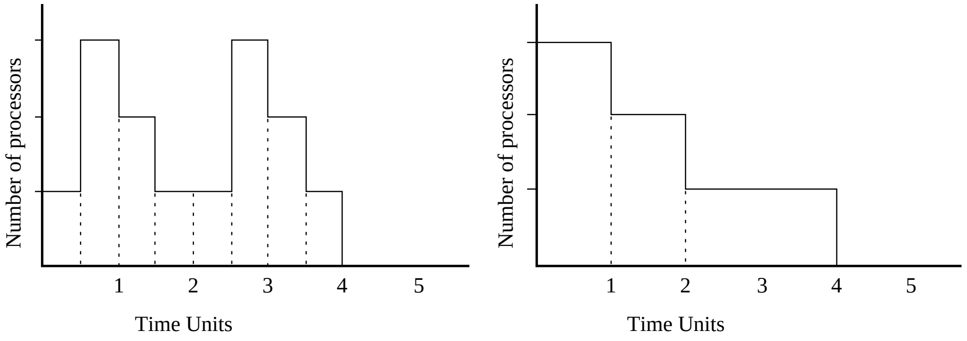


Figure 4: Circulating Processor Model: 3 Processors, 1 Parallel Application



(a) Parallelism Profile

(b) Parallelism Shape

Figure 5: Possible Parallelism Profile and Shape for an Application

a single application, the approximate model is a simple two device closed queueing network. The multiprogramming level is three, corresponding to the number of processors.

A processor is idle whenever it is not servicing any task of a parallel application. Idle times for a given processor occur when the queue at that processor is empty. The various processors may be idle at various times, since sibling tasks complete execution at different times.

The service center loadings, or service times, are dependent on the number of processors currently at the center, and are labeled in Figure 4. The loadings at the idle server when there are 1, 2, and 3 processors present are designated by $L_I(1), L_I(2)$, and $L_I(3)$, respectively. The loadings at parallel application A when there are 1, 2, and 3 processors present (i.e., allocated or assigned to A) are designated by $L_A(1), L_A(2)$, and $L_A(3)$, respectively. The model parameterization issue is to determine these device loadings.

In the load dependent circulating processor model, the demands are calculated based on the parallelism profile (or parallelism shape) of the application. The parallelism profile is determined experimentally, and indicates the number of processors in use by the application as a function of time during the execution of the application [9]. Figure 5(a) illustrates a possible parallelism profile for an application which is allocated 3 processors. The parallelism shape is obtained directly from the parallelism profile, and gives the cumulative fraction of time that each number of processors is allocated. Figure 5(b) illustrates the corresponding parallelism shape for the same parallel application. In the load dependent circulating processor model, the load dependent demands (mean loadings) for parallel application A (i.e., the $L_A(i)$'s) are parameterized by the parallelism shape directly. That is, $L_A(i)$ is set to the time that the program executes on i processors. In Figure 5, $L_A(3) = 1$, $L_A(2) = 1$, and $L_A(1) = 2$. Also given by the parallelism shape is the amount of the time that the application uses 0 processors, due to delays caused by communication, or between invocations of the application. This time is indicated by $L_A(0)$, and can be calculated as the difference between the total time that measurements were taken on the system and the total time that the application was executing on one or more processors.

If the underlying Markov state space diagram of the actual target system is available, the load dependent demands can also be calculated analytically. The state probabilities can be found by solving the global balance equations. The total demand of a parallel application for a given number of processors is calculated as the mean time spent in the set of states in which that given number of processors is present at the parallel application. For example, the mean service time of application A when there is one processor present, $L_A(1)$, is calculated as:

$$
\begin{aligned}
L_A(1) &= \text{time spent in } \langle 2, 1 \rangle \\
&= \text{time spent in } \left\{ \left\langle \begin{matrix} - \\ - \\ A_3 \end{matrix} \right\rangle, \left\langle \begin{matrix} - \\ A_2 \\ - \end{matrix} \right\rangle, \left\langle \begin{matrix} A_1 \\ - \\ - \end{matrix} \right\rangle \right\} \\
&= \frac{\text{Prob}\left[\left\{ \left\langle \begin{matrix} - \\ - \\ A_3 \end{matrix} \right\rangle, \left\langle \begin{matrix} - \\ A_2 \\ - \end{matrix} \right\rangle, \left\langle \begin{matrix} A_1 \\ - \\ - \end{matrix} \right\rangle \right\} \right]}{\text{throughput out of } \left\{ \left\langle \begin{matrix} - \\ - \\ A_3 \end{matrix} \right\rangle, \left\langle \begin{matrix} - \\ A_2 \\ - \end{matrix} \right\rangle, \left\langle \begin{matrix} A_1 \\ - \\ - \end{matrix} \right\rangle \right\}} \\
&= \frac{\text{Prob}\left[\left\langle \begin{matrix} - \\ - \\ A_3 \end{matrix} \right\rangle \right] + \text{Prob}\left[\left\langle \begin{matrix} - \\ A_2 \\ - \end{matrix} \right\rangle \right] + \text{Prob}\left[\left\langle \begin{matrix} A_1 \\ - \\ - \end{matrix} \right\rangle \right]}{\frac{1}{a_3}\text{Prob}\left[\left\langle \begin{matrix} - \\ - \\ A_3 \end{matrix} \right\rangle \right] + \frac{1}{a_2}\text{Prob}\left[\left\langle \begin{matrix} - \\ A_2 \\ - \end{matrix} \right\rangle \right] + \frac{1}{a_1}\text{Prob}\left[\left\langle \begin{matrix} A_1 \\ - \\ - \end{matrix} \right\rangle \right]}
\end{aligned}
$$

In general, the total demand for a given number of processors is equal to the mean service time that an application requires the given number of processors. From Little's Result [11], the mean service time is equal to the total probability of being in the set of states (i.e., the states using the given number of processors), divided by the throughput from all individual states in the set. In this case, the visit ratios for all sets of states in the actual system are the same and equal to 1, since flow goes through sets $\langle 0, 3 \rangle$, $\langle 1, 2 \rangle$, $\langle 2, 1 \rangle$, $\langle 3, 0 \rangle$, and back to $\langle 0, 3 \rangle$ (see Figure 3). Therefore, the total demand (mean loading) of application A when there is one processor present, $L_A(1)$, is equal to the mean service time while in $\langle 2, 1 \rangle$.

Note that $L_A(1) = L_I(2)$ and $L_A(2) = L_I(1)$. In general, for a single parallel application,

$$L_A(i) = L_I(N - i), \quad 0 \leq i \leq N \tag{1}$$

where $L_A(0)$ indicates the mean time that application A uses 0 processors.

Figure 6 shows the state space diagram for the load dependent circulating processor model for one application and three processors. The state space description for the circulating processor model in Figure 6 is the vector $\langle n_I, n_A \rangle$, where n_I is the number of idle processors, and n_A is the number of processors executing at parallel application A. Each state in the diagram for the load dependent circulating processor model is denoted by $\langle N - i, i \rangle_{\text{ld cp}}$, where i is the number of busy processors.

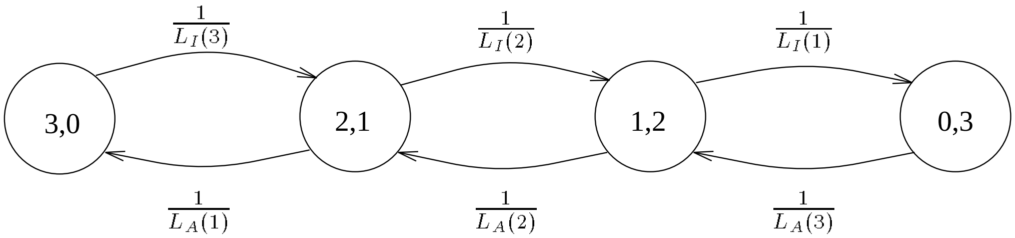


Figure 6: Markov Diagram for Load Dependent CP Model: 3 Processors, 1 Parallel Application

The example shows that there is a one to one correspondence between each state, $\langle N-i,i\rangle_{\text{ld cp}}$, in the circulating processor diagram (i.e., Figure 6), and a set of states, $\langle N-i,i\rangle_{\text{actual}}$, in the actual system diagram (i.e., Figure 3). There is not a one to one correspondence between the arcs in the two diagrams. For example, there is positive flow from state $\langle 3,0\rangle_{\text{ld cp}}$ to state $\langle 2,1\rangle_{\text{ld cp}}$ in the circulating processor diagram, but there is no flow from state $\langle 3,0\rangle_{\text{actual}}$ to state $\langle 2,1\rangle_{\text{actual}}$ in the actual system. However, as shown below, the steady state probability distributions for the two Markov diagrams are identical.

The equivalence of the steady state probability distribution for the states in the load circulating processor model, and the sets of states in the actual system is true in general for one parallel application, for any number of circulating processors. Specifically:

Lemma: The probability of being in state $\langle N-i,i\rangle_{\text{ld cp}}$ in the load dependent circulating processor model is equal to the probability of being in the corresponding partition $\langle N-i,i\rangle_{\text{actual}}$ in the actual target system.

Proof: This can be shown by induction on i.

Let T be the total time spent in all states.

$$T = \sum_{i=0}^{N} L_A(i) = \sum_{i=0}^{N} L_I(i).$$

Prob$[\langle N,0\rangle_{\text{ld cp}}]$ can be calculated from the birth-death Markov diagram of the circulating processor model.

$$\begin{aligned}
\text{Prob}[\langle N,0\rangle_{\text{ld cp}}] &= \frac{1}{1+\sum\limits_{i=1}^{N}\prod\limits_{j=1}^{i}\frac{\frac{1}{L_I(N-j+1)}}{\frac{1}{L_A(j)}}} \\
&= \frac{1}{1+\sum\limits_{i=1}^{N}\frac{\frac{1}{L_I(N)}}{\frac{1}{L_A(i)}}} \\
&= \frac{L_I(N)}{\sum\limits_{i=0}^{N} L_A(i)} \\
&= \frac{L_I(N)}{T} \\
&= \text{Prob}[\langle N,0\rangle_{\text{actual}}].
\end{aligned}$$

Now, assume that $\mathrm{Prob}[\langle N-i, i\rangle_{\mathrm{ld\ cp}}] = \mathrm{Prob}[\langle N-i, i\rangle_{\mathrm{actual}}]$ for some i. It remains to show that $\mathrm{Prob}[\langle N-i-1, i+1\rangle_{\mathrm{ld\ cp}}] = \mathrm{Prob}[\langle N-i-1, i+1\rangle_{\mathrm{actual}}]$.

Since

$$\begin{aligned} \mathrm{Prob}[\langle N-i, i\rangle_{\mathrm{actual}}] &= \frac{L_A(i)}{\sum_{j=0}^{N} L_A(j)}, \quad \text{and} \\ \mathrm{Prob}[\langle N-i-1, i+1\rangle_{\mathrm{actual}}] &= \frac{L_A(i+1)}{\sum_{j=0}^{N} L_A(j)}, \end{aligned}$$

then

$$\begin{aligned} \mathrm{Prob}[\langle N-i-1, i+1\rangle_{\mathrm{actual}}] &= \mathrm{Prob}[\langle N-i, i\rangle_{\mathrm{actual}}]\frac{L_A(i+1)}{L_A(i)} \\ &= \mathrm{Prob}[\langle N-i, i\rangle_{\mathrm{ld\ cp}}]\frac{L_A(i+1)}{L_A(i)} \\ &= \mathrm{Prob}[\langle N-i-1, i+1\rangle_{\mathrm{ld\ cp}}]. \end{aligned}$$

□

From a general perspective, it is noted that the mapping from $\langle N-i, i\rangle_{\mathrm{actual}}$ to $(N-i, i)_{\mathrm{ld\ cp}}$ is exact due to calculating the total mean time in each partition as a function of the exact probabilities and throughput of the individual states in the partition. The steady state probability distributions are equivalent even though the Markov chain does not satisfy lumpability conditions [12]. Since the total flow out of each partition goes to a single other partition, the rate of flow is equal to the inverse of the total mean time in the source partition. This results in a cyclic Markov chain with $N+1$ states, as seen in Figure 3. The cyclic chain is equivalent to a birth-death Markov chain for the circulating processor model as shown in Figure 6.

Due to the equivalence of the steady state probability distributions of the actual system and the circulating processor model, any Markov reward function which rewards these corresponding states equally will yield equivalent performance measures. Therefore, the mean utilization of the application (i.e., the average parallelism) will be the same for both models. However, throughput must be viewed differently in the actual system than in the circulating processor model. The throughput in the circulating processor model (i.e., Figure 6) is *processor throughput*, and gives the number of processors which complete per unit time, and is calculated as

$$\sum_{i=0}^{N-1} \mathrm{Prob}[\langle N-i, i\rangle]\frac{1}{L_I(N-i)}.$$

Throughput viewed in the actual system (i.e., Figure 3) is *application throughput*, and gives the number of applications which complete per unit time. Application throughput calculated from the load dependent circulating processor model, $\mathrm{TPUT}_{\mathrm{ld\ cp}}$, is equal to processor throughput divided by the number of processors required to complete the application. For a single parallel application, when the application forks to all available processors, the number of processors required to complete

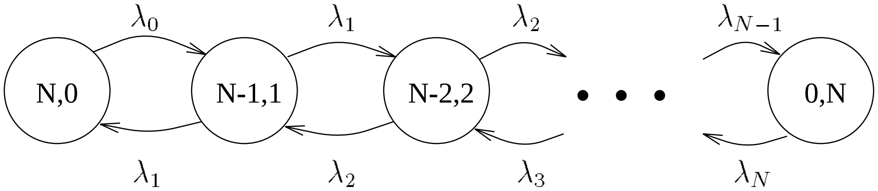


Figure 7: Markov Diagram for the Load Dependent CP Model: 1 Parallel Application

the application is the total number of processors, N. Thus, the application throughput of the load dependent circulating processor model is given as

$$\begin{aligned} \mathrm{TPUT}_{\mathrm{ld\ cp}} &= \frac{1}{N}\sum_{i=0}^{N-1} \mathrm{Prob}[\langle N-i, i\rangle_{\mathrm{ld\ cp}}]\frac{1}{L_I(N-i)} \qquad (2)\\ &= \frac{1}{N}\sum_{i=0}^{N-1} \mathrm{Prob}[\langle N-i, i\rangle_{\mathrm{actual}}]\frac{1}{L_I(N-i)}\\ &= \frac{1}{N}\sum_{i=0}^{N-1} \frac{1}{\sum_{j=0}^{N} L_A(j)}\\ &= \frac{1}{\sum_{j=0}^{N} L_A(j)}\\ &= \mathrm{TPUT}_{\mathrm{actual}} \end{aligned}$$

Therefore, the load dependent circulating processor model is exact for a system in which there is a single parallel application. Specifically, the probability of being in the set $\langle N-i, i\rangle_{\mathrm{actual}}$ in the actual system, in which there are i active tasks at the multiprocessor, is equal to the probability of being in the single state $\langle N-i, i\rangle_{\mathrm{ld\ cp}}$ in the circulating processor model, in which there are are i active processors. Further, application throughput can be calculated from the circulating processor model, and is equal to the throughput of the actual system.

3.2 Load Independent Circulating Processor Model

Although the load dependent circulating processor model is exact in the single application case, it requires that the parallelism shape be known in order to parameterize the model. If the parallelism shape is unknown, it is still possible to construct a load independent circulating processor model. In this case, loadings at each server are the same regardless of the number of processors which are currently at the parallel application. For the load independent model, $L_A(i) = L_A(j)$ and $L_I(i) = L_I(j)$ for all i and j. Since the load dependent circulating processor model maps exactly to the actual system, the error between the actual system and the load independent circulating processor model can be determined by comparing the load dependent and the load independent circulating processor models.

Consider the slightly modified Markov diagram for the load dependent circulating processor model shown in Figure 7. This is a generalization of Figure 6. In Figure 7 the rates are labeled

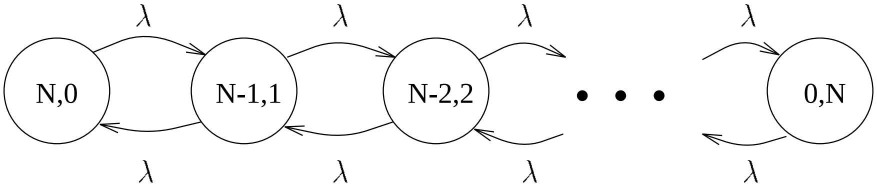


Figure 8: Markov Diagram for the Load Independent Counterpart Model: 1 Parallel Application

λ_i, and represent the rate of processing at the parallel application when i processors are allocated to the application (i.e., $\lambda_i = \frac{1}{L_{A(i)}}$). The load independent counterpart model is constructed by calculating the "average" rates of processing of both the parallel application and the idle server. The average rate of service of the idle processor, λ_I, is calculated as the average of all rates flowing from left to right in Figure 7, weighted by the state probabilities. Similarly, the average rate of service of application A, λ_A, is calculated as the average of all rates flowing from right to left in Figure 7, likewise weighted by the state probabilities.

$$\lambda_I = \text{Prob}[\langle N,0\rangle_{\text{ld cp}}]\lambda_0 + \text{Prob}[\langle N-1,1\rangle_{\text{ld cp}}]\lambda_1 + \cdots + \text{Prob}[\langle 1,N-1\rangle_{\text{ld cp}}]\lambda_{N-1}$$

$$\lambda_A = \text{Prob}[\langle 0,N\rangle_{\text{ld cp}}]\lambda_N + \text{Prob}[\langle 1,N-1\rangle_{\text{ld cp}}]\lambda_{N-1} + \cdots + \text{Prob}[\langle N-1,1\rangle_{\text{ld cp}}]\lambda_1$$

The solution of the Markov diagram gives the probability of being in state $\langle N,0\rangle_{\text{ld cp}}$ as

$$\begin{aligned}\text{Prob}[\langle N,0\rangle_{\text{ld cp}}] &= \frac{1}{1+\frac{\lambda_0}{\lambda_1}+\frac{\lambda_0}{\lambda_2}+\cdots+\frac{\lambda_0}{\lambda_N}} \\ &= \frac{\lambda_1\lambda_2\cdots\lambda_N}{\sum_{i=0}^{N}\prod_{j=0,j\neq i}^{N}\lambda_j}. \\ \text{Prob}[\langle N-k,k\rangle_{\text{ld cp}}] &= \frac{\prod_{j=0,j\neq k}^{N}\lambda_j}{\sum_{i=0}^{N}\prod_{j=0,j\neq i}^{N}\lambda_j}. \qquad (3)\end{aligned}$$

Thus, by substituting to find λ_I and λ_A, the load independent service rates for the idle server and the application are

$$\lambda = \lambda_I = \lambda_A = \frac{N\prod_{i=0}^{N}\lambda_i}{\sum_{i=0}^{N}\prod_{j=0,j\neq i}^{N}\lambda_j}.$$

The Markov diagram for the load independent circulating processor model of Figure 7 is illustrated in Figure 8. Since all rates on all arcs are equal in the Markov diagram of the load independent model, the probability of all states in the load independent model are equal (i.e., in Figure 8, $\text{Prob}[\langle N-i, i\rangle_{\text{li cp}}] = \frac{1}{N+1} \ \ \forall\ i$). Therefore, performance metrics can be calculated directly. For example, the rate at which processors leave the idle server in the load independent circulating processor model is

$$\lambda(\text{Prob}[\langle N,0\rangle_{\text{li cp}}] + \cdots + \text{Prob}[\langle 1, N-1\rangle_{\text{li cp}}]) = \lambda(\frac{1}{N+1} + \cdots + \frac{1}{N+1}) = \lambda\frac{N}{N+1}.$$

That is, given a single parallel application in the system, with N processors at the multiprocessor, the processor throughput of the load independent circulating processor model is equal to $\frac{N}{N+1}\lambda$. Therefore, since each application requires service from each of the N processors (in parallel), the application throughput of the load independent circulating processor model is

$$\text{TPUT}_{\text{li cp}} = \frac{1}{N}\frac{N}{N+1}\lambda = \frac{\lambda}{N+1}$$

By substituting Equation 3 into Equation 2 and simplifying, the application throughput of the load dependent circulating processor model, $\text{TPUT}_{\text{ld cp}}$, is equal to $\frac{\lambda}{N}$. (Recall that it was shown that $\text{TPUT}_{\text{ld cp}}$ was exact and equal to $\text{TPUT}_{\text{actual}}$.) Therefore, the relative error of application throughput of the load independent circulating processor model is dependent only on the number of processors at the multiprocessor, and is equal to $\frac{1}{N+1} \times 100\%$.

4. Analysis with Multiple Parallel Applications

When the number of applications executing in the system is larger than one, the scheduling policy at the multiprocessor must be specified. The state space diagram for the actual system will vary, depending on these assumptions. The applicability of the various circulating processor models will also vary, depending on the scheduling assumptions made for the actual system. The two extreme scheduling policies are analyzed here, the fully parallel scheduling case and the fully sequential scheduling case.

4.1 Fully Parallel Scheduling

In fully parallel scheduling, a parallel application arrives at the multiprocessor and forks into N tasks, each of which is scheduled on a unique processor. The state space diagram for fully parallel scheduling with 2 parallel applications and with 2 processors is shown in Figure 9. The two applications are A and B. The tasks of application A are labeled A_i, and the tasks of application B are labeled B_i. The tasks execute at the multiprocessor using a FCFS queueing discipline. When a task completes, the processor on which it was executing is released, and it may begin executing a task of another parallel application. There is one additional server in the network, which is a delay server. The delay server is used to represent the time that a processor spends not servicing a parallel application. The delay server may represent waiting time while communication occurs between tasks, waiting time while sibling tasks complete, or idle time between the submission of parallel applications. The labels A_0 and B_0 indicate the presence of the application at the delay server. The service times at the servers are assumed to be exponentially distributed, but the mean service times for different applications may be different.

In Figure 9, the states are labeled with the tasks that are active or queued at each device in that state. In the state labeled $\langle -, \overset{B_1 A_1}{-} \rangle$, task A_1 arrived first and is executing at the "top" processor. Task B_1 arrived second and is waiting to execute in a FCFS queueing discipline at the

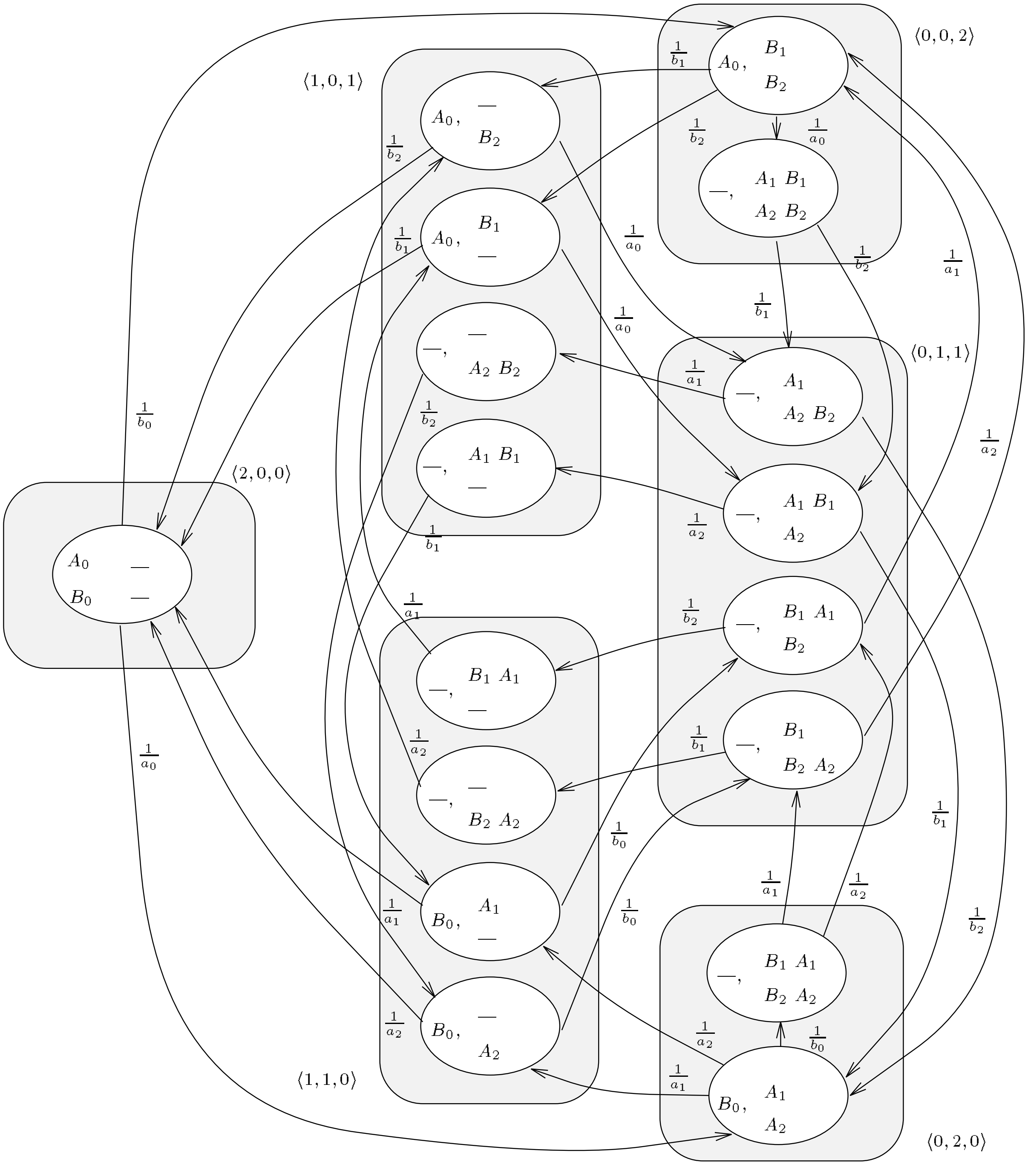


Figure 9: Markov Diagram for the Actual System: 2 Processors, 2 Applications

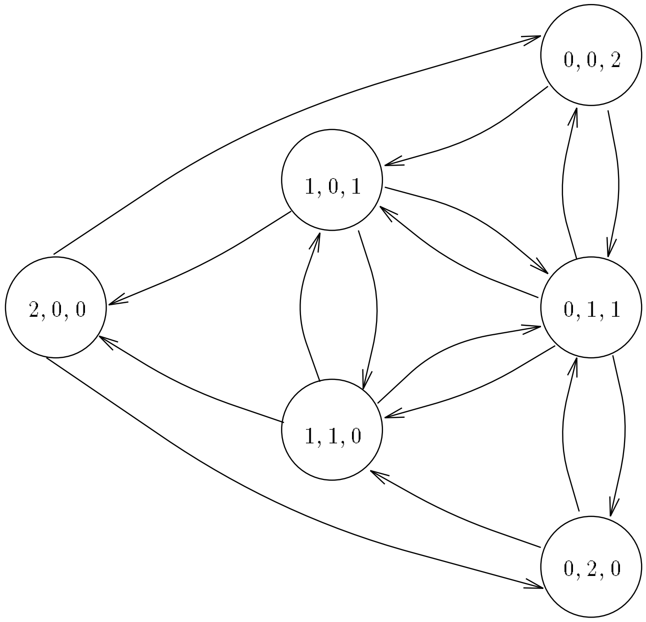


Figure 10: Simplified Markov Diagram for the Actual System: 2 Processors, 2 Applications

same processor. In state $\langle -, \overset{B_1 A_1}{-} \rangle$ no tasks are either waiting or executing at the other processor of the multiprocessor or at the delay server. In the state labeled $\langle \overset{A_0}{B_0}, \overset{-}{-} \rangle$, both application A and B are "executing" at the delay server. The "tasks" A_0 and B_0 execute at rates $\frac{1}{a_0}$ and $\frac{1}{b_0}$, respectively. There are two possible arcs leaving state $\langle \overset{A_0}{B_0}, \overset{-}{-} \rangle$, representing the completion of the two tasks at the delay server (i.e., an arrival to the multiprocessor). This system is not product form, so global balance equations must be solved in order to determine the state probabilities.

In the system containing a single parallel application (Section 3), the circulating processor model is compared against the actual system by grouping the states of the Markov diagram for the actual system (Figure 3) into sets of states with a more abstract state description. The same procedure is applied to the system with 2 parallel applications. The aggregated state description for the system with 2 parallel applications is a vector $\langle n_I, n_A, n_B \rangle$, where n_I is the number of idle processors (i.e., processors which are not actively executing a task), n_A is the number of processors actively executing a task of parallel application A, and n_B is the number of processors actively executing a task of parallel application B. For 2 parallel applications, with 2 processors, the aggregated state space is $\{\langle 2,0,0 \rangle, \langle 1,1,0 \rangle, \langle 1,0,1 \rangle, \langle 0,2,0 \rangle, \langle 0,1,1 \rangle, \langle 0,0,2 \rangle\}$. In Figure 9 the sets of states in the aggregate state space are shaded. Using the aggregate state description, a simplified picture of the Markov diagram is shown in Figure 10.

The load dependent circulating processor model for two parallel applications is illustrated in Figure 11. There are three service centers in this model, one each for parallel application, A and B, and one for the idle server, I. The multiprogramming level of the model is equal to the number of processors in the multiprocessor, 2. The load dependent demands are labeled for applications A and B and for the idle server, I. The branching probabilities are labeled as P_{ij}, which represents the probability of a processor which completes at server i next being assigned to server j, for $i, j \in \{A, B, I\}$.

In the circulating processor model, it is possible for a processor to leave an application and branch either to the other application or to the idle server. When a processor leaves the idle server,

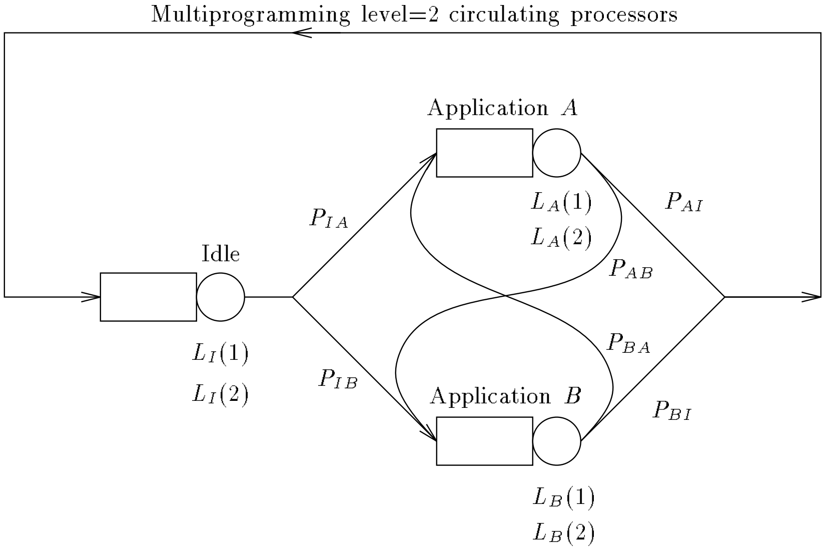


Figure 11: Load Dependent Circulating Processor Model: 2 Processors, 2 Applications

due to being allocated to an application, it branches to a parallel application with a specified probability. These branching probabilities, P_{IA} and P_{IB}, are dependent on the relative throughputs of the applications. If a processor leaves an application and branches directly to another application (i.e., P_{AB} or P_{AB}), this represents the event that a parallel task of a second application has been queued at this processor while it is still busy executing a task of the first application. As soon as the processor completes the current task, it begins executing the queued task immediately. If no task is in the wait queue for the processor in the actual system, then this is modeled in the circulating processor model by the event that the processor branches to the idle server (i.e., P_{AI} or P_{BI}).

The Markov diagram for the circulating processor model is shown in Figure 12. The states are labeled $\langle 2,0,0\rangle, \langle 1,0,1\rangle, \langle 1,1,0\rangle, \langle 0,0,2\rangle, \langle 0,1,1\rangle$, and $\langle 0,2,0\rangle$. The rate of flow along each arc is labeled. As in the case of a single parallel application, the Markov diagram of Figures 9 and 12 cannot be mapped exactly from one to another. Even though there is a one-to-one correspondence of states between the two diagrams, there is not a one-to-one correspondence of arcs between the states. For example, there is an arc with positive flow from state $(2,0,0)_{\text{ld cp}}$ to state $(1,0,1)_{\text{ld cp}}$ in the load dependent circulating processor model, but there is no flow from state $(2,0,0)_{\text{actual}}$ to state $(1,0,1)_{\text{actual}}$ in the actual system. Unlike the system with a single parallel application, the steady state probability distribution of the actual system is not equivalent to the steady state probability distribution of the load dependent circulating processor model. This leads to error when using the circulating processor model for calculating the performance of a system which contains multiple parallel applications.

4.1.1 Load Dependent Circulating Processor Model

For the simple case of two parallel applications and two processors, the performance metrics of the actual system can be calculated directly from the Markov diagram (Figure 9) by solving the global balance equations. The performance metrics of the load dependent circulating processor model can be calculated using a product form solution technique, such as mean value analysis [2]. The error

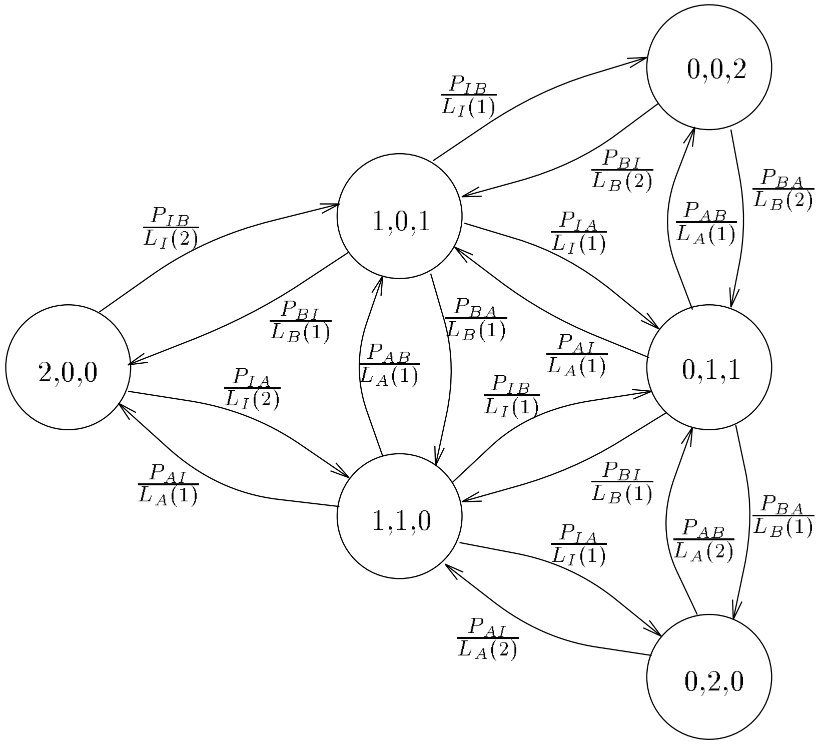


Figure 12: Markov Diagram for the CP Model: 2 Processors, 2 Programs

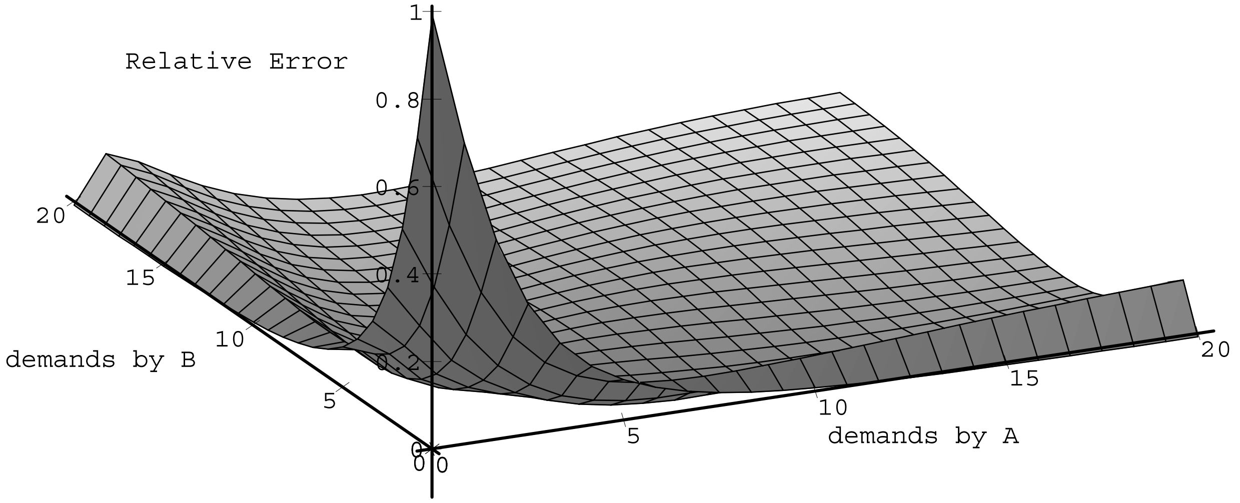
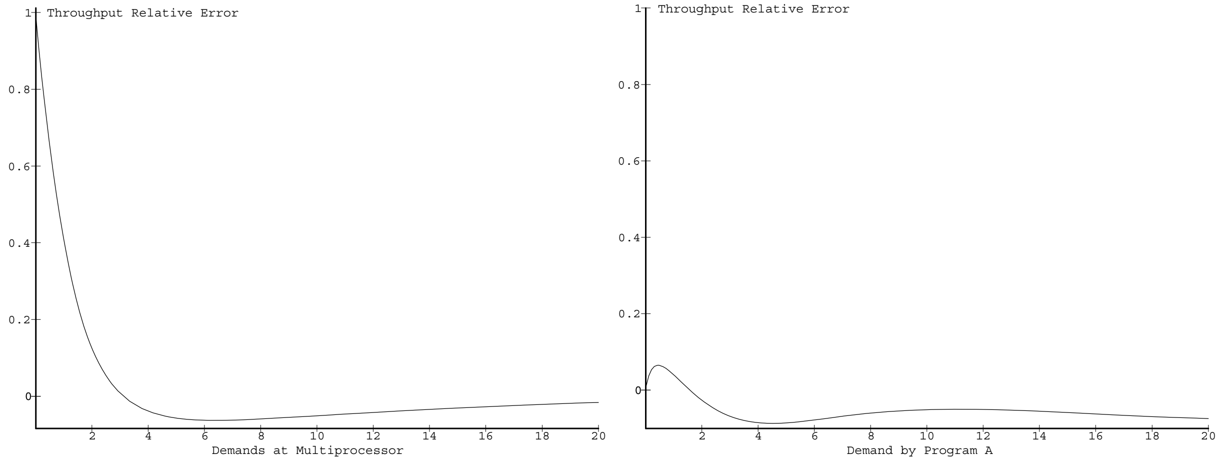


Figure 13: Throughput Relative Error of Load Dependent CP Model, 2 Applications, 2 Processors



(a) Balanced Multiprocessor

(b) Application B Demand=10

Figure 14: Throughput Relative Error

between the two models can be calculated. Figure 13 illustrates the relative error between the actual throughput and the load dependent circulating processor model throughput as the demands at the applications A and B are varied. In this figure the demands at the delay server for both applications are set equal to 1.0. The demands of the two multiprocessor tasks of application A are equal to each other, and the demands of the two multiprocessor tasks of application B are equal to each other. The greatest relative error occurs when the demands by tasks of applications A and B are both small (i.e, when the multiprocessor is lightly loaded and the delay server is more heavily loaded). As the multiprocessor becomes more heavily utilized, the relative error decreases.

Along the diagonal slice of Figure 13 where demands by tasks of application A equal the demands by tasks of application B, all tasks place the same demands on the multiprocessor, and the processors at the multiprocessor are balanced. Figure 14(a) illustrates the relative error when all multiprocessor task demands are equal. The relative error along this slice is shown in Figure 14(a). It can be analytically shown that the relative error approaches $\frac{1}{21}$ as the demands of all tasks approach infinity. The relative error approaches 1 (i.e., 100%) as the demands of all tasks approach zero.

Figure 14(b) illustrates the relative error along the slice of Figure 13 where the demand of each task of Application B is equal to 10. The demand the tasks of Application A are varied. The relative error is close to 0.0, when the demand by Program A is close to 0. The relative error is positive for small demands by Program A. As the demand by Program A increases, the relative error become negative and remains negative. This indicates that the load dependent circulating processor model tends to overestimate throughput when the multiprocessor is heavily loaded. In this case the error is always less than 10%.

The method of calculating the performance metrics of the actual system directly from the Markov diagram does not extend to higher multiprogramming levels or to more processors, because the number of states in the state space becomes large. A discrete event simulation for the solution of the actual fork-join system was constructed using the queueing network solution package QNAP [13]. All simulations were run to a 95% confidence level. From the simulation, measurements were collected in order to calculate the load dependent loadings and the branching probabilities.

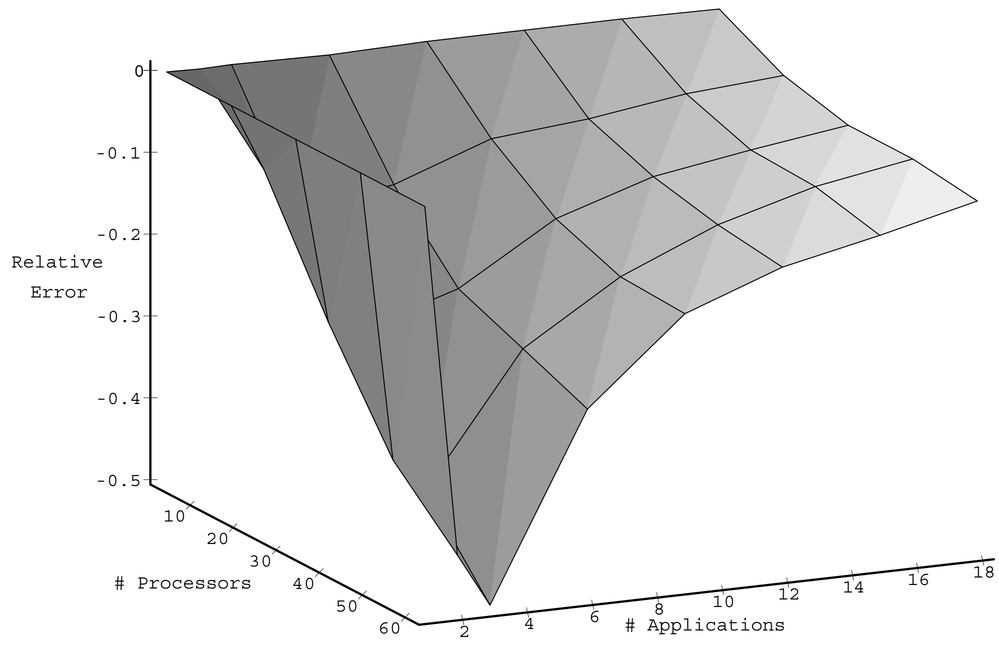


Figure 15: Throughput Relative Error of Load Dependent CP Model: Multiple Applications and Processors

These measurements were used to parameterize the circulating processor model. The load dependent circulating processor model was solved exactly by QNAP, using a load dependent mean value analysis algorithm.

Figure 15 shows the relative error between the actual system and its circulating processor model. In this figure the task demands are set to 1.0 and the number of processors and the multiprogramming level are varied over a wide range. As expected, no error occurs with 1 application. The figure shows that the greatest relative error in throughput occurs when the multiprogramming level is small (2 to 3), and when the number of processors is large. As the multiprogramming level (i.e., the number of parallel applications) increases, the relative error decreases.

4.1.2 Load Independent Circulating Processor Model

The performance metrics of the load independent circulating processor model can be calculated using a product form solution technique. As in the load dependent case, the error between the load independent circulating processor model and the actual system can be calculated for the simple case of two parallel applications and two processors, since the performance metrics of the actual system can be calculated directly from the Markov diagram (Figure 9). Figure 16 shows the relative error between the throughput of the actual system and the load independent circulating processor model for 2 parallel applications and 2 processors.

In Figure 16, the demands at the delay server for each application are set equal to 1.0. The demands of the two tasks of application A are equal, and the demands of the two tasks of application B are equal. The demands of the tasks of application A and B are allowed to vary independently. Figure 16 shows that the relative error is always positive, and tends to be high except when the multiprocessor is very lightly loaded (i.e., when the demands of the tasks of the two applications are both close to 0). The load independent circulating processor model is not a good approximation to the actual system in the case of multiple parallel applications.

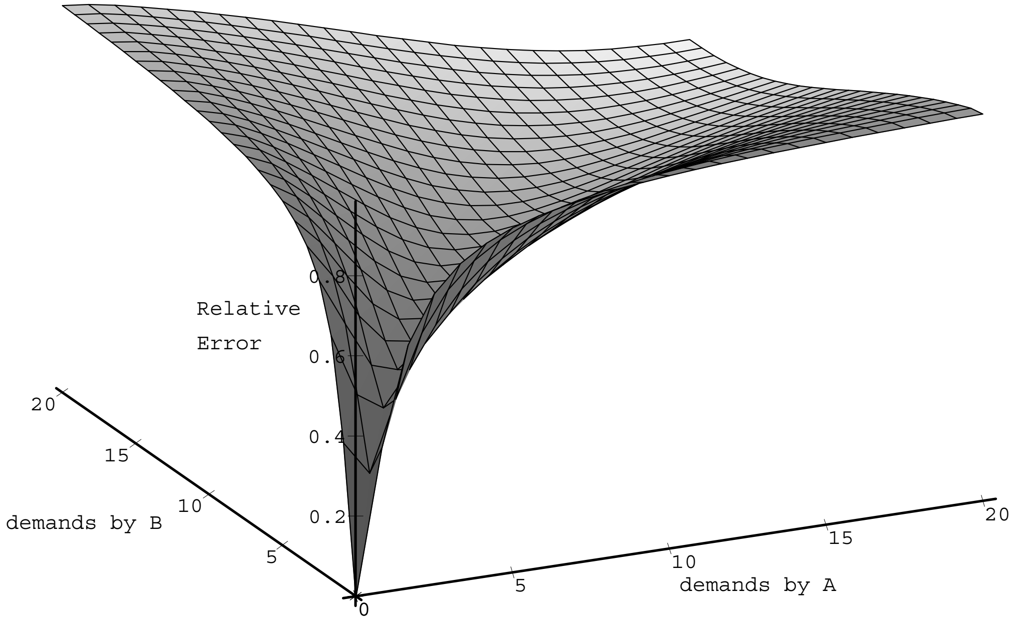


Figure 16: Throughput Relative Error for Load Independent CP Model: 2 Applications, 2 Processors

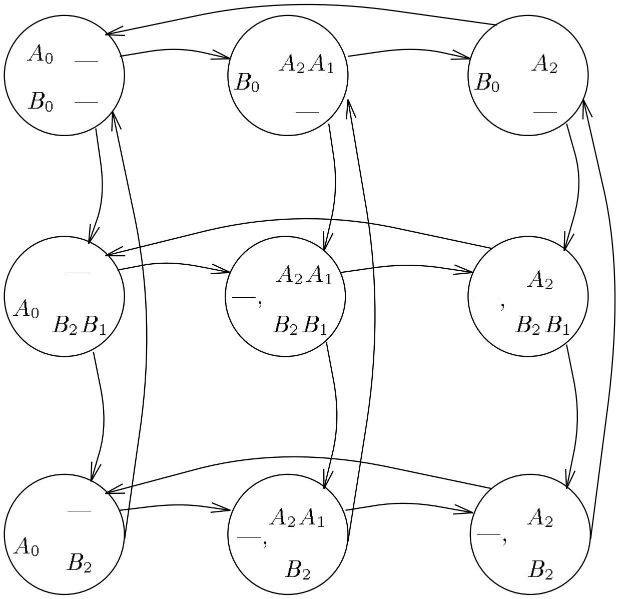


Figure 17: Markov Diagram for Fully Sequential Scheduling

4.2 Fully Sequential Scheduling

In the case of a single parallel application, when the number of tasks forked is equal to the number of processors, it is possible to construct a circulating processor model which gives the exact performance metrics as the actual system (see Section 3.1). It is also possible to construct a circulating processor model which is exact when fully sequential scheduling is used. In fully sequential scheduling, all tasks from the same applications are assigned to the same processor and are executed sequentially on that processor. The application is complete when all of its component tasks have completed execution. The applications are assigned to a free processor in the order that they arrive. Figure 17 shows the Markov diagram for a system in which there are two parallel applications and two processors. The processors are assumed to be identical.

Figure 17 can also be viewed as the Markov diagram for the circulating processor model. In this Markov diagram, processor one visits application A, followed by the idle server, followed by application A, and so on. Processor two visits application B, followed by the idle server, followed by application B, and so on. The processors circulate independently in the closed queueing network. As long as the number of processors is equal to the number of applications in the parallel system, this model will be exact for any task graph. The model is product form and mean performance metrics can be found using efficient solution techniques.

5. Comparison to the Circulating Task Model

A more traditional way of modeling parallel systems is to construct a single class load independent product form queueing network model in which the tasks of the parallel application are viewed as independent customers which circulate among the processors of the network [14]. In this model, N service centers represent the N processing nodes of the multiprocessor system. A delay server models the interarrival times of the applications to the multiprocessors. The service demands at each of the service centers are determined from measurement data. By measuring the utilization of each processor and the number of tasks served, their ratio gives the mean service demand. The

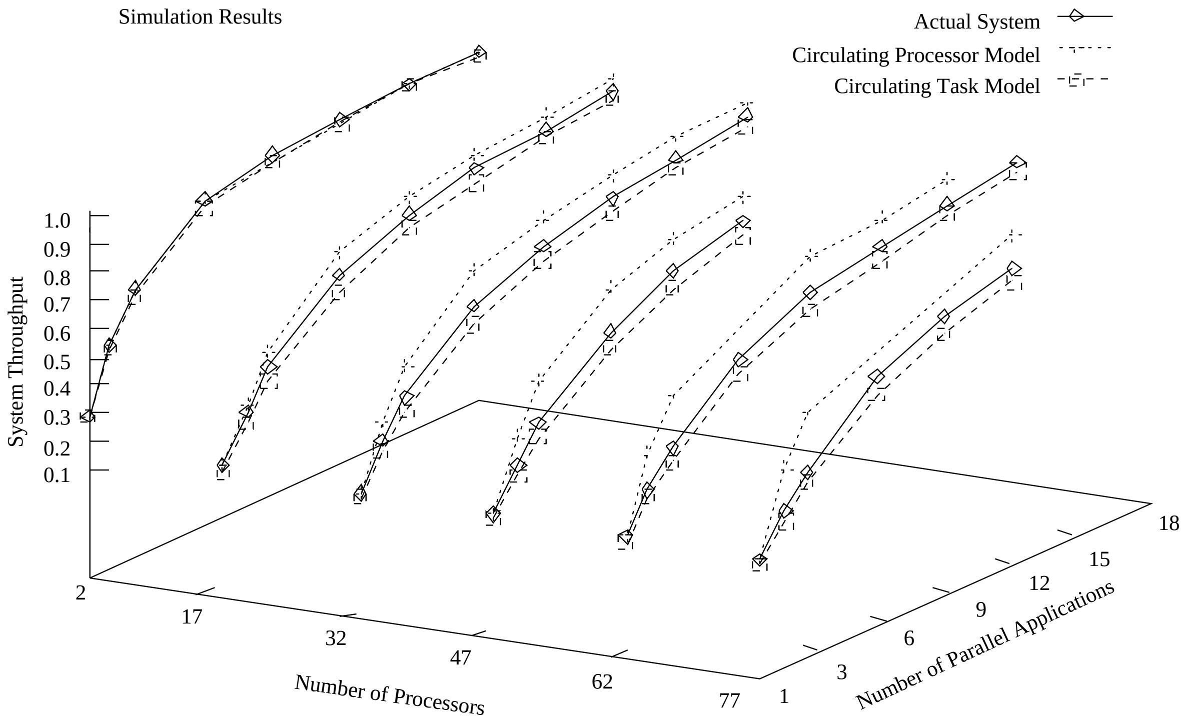


Figure 18: Throughput Comparisons, Parallel Scheduling

average measured number of tasks, or task multiprogramming level, M_T, is measured as the sum of the mean queue lengths over all service centers [1] Because there may be multiple parallel tasks per application, M_T will be larger than the number of circulating applications, M.

Figure 18 compares the throughput of actual fork-join systems against both the load dependent circulating processor model and the circulating task model. Fully parallel scheduling is assumed. The curves in Figure 18 were generated via simulation. The number of processors varies from 2 to 77 and the number of parallel application varies from 1 to 18.

Figure 18 shows that the load dependent circulating processor model tends to overestimate throughput, while the circulating task model tends to underestimate throughput. Both models give results which are comparable to the actual system.

6. An Image Processing Application

A case study to measure the performance of a real parallel application was performed in order to show the potential usefulness of the circulating processor model. An image processing application was selected. Figure 19 shows the task graph of the image processing application. The application first receives a data image, and the data image is split (via task A_S) into a number of equal sized components. Each of these components (tasks A_1, A_2, and A_3) is passed to a (usually unique)

[1] In the simplest model, M_T is a single number and generally non-integral. When M_T is non-integral, interpolation is used to calculate the performance measures. For example,

$$M_T = \lfloor M_T \rfloor + \alpha, \quad \text{where } 0 \le \alpha \le 1$$

$$\text{TPUT}(M_T) \approx (1-\alpha)\text{TPUT}(\lfloor M_T \rfloor) + \alpha \text{TPUT}(\lceil M_T \rceil)$$

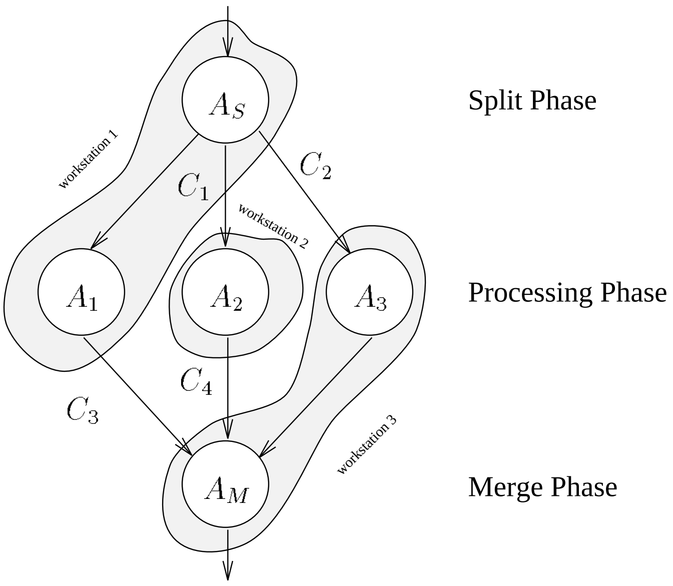


Figure 19: Image Processing Application Task Graph

Service	Service Rate
splitter/merger tasks (A_S, A_M)	0.7398
processing tasks (A_1, A_2, A_3)	0.0572
communications (C_1, C_2, C_3, C_4)	2.2367

Table 1: Service Rates for Image Processing Application

processor, where processing takes place. The final phase (task A_M) of the application merges the resultant components back into a single image.

This image processing application was executed on a network of workstations connected by an Ethernet LAN. In the version illustrated in Figure 19, tasks A_S and A_1 were assigned to workstation 1, task A_2 was assigned to workstation 2, and tasks A_3 and A_M were assigned to workstation 3. Communication was required between tasks assigned to different workstations. The communications between workstations are illustrated in Figure 19 as C_1, C_2, C_3, and C_4. Each of the communications sends the same amount of data, so that the mean service rates for all communications are equal. Also, each of the tasks A_1, A_2, and A_3 are identical, since they process equal sized datasets. The tasks A_M and A_S are known to have approximately the same service rates. The measured rates of service are shown in Table 1.

In the image processing system, a set of data arrives and is processed. As soon as the processing of one set of data is completed, a image (i.e., new set of data) arrives to the system. Thus, the system can be modeled as a closed queueing network with three processors. Communication is modeled by a delay server (e.g., Figure 1), since task processing is blocked during the time that communication occurs between processors. Time at the delay server includes contention which may occur for communication.

Figure 20 shows the Markov diagram for the image processing application when scheduled and executed on a network of three workstations. The states are divided into partitions according to the number of tasks (processors) which are active in that state. Since the initiation of tasks on separate workstations requires an interleaving communication, the tasks initiated on remote workstations

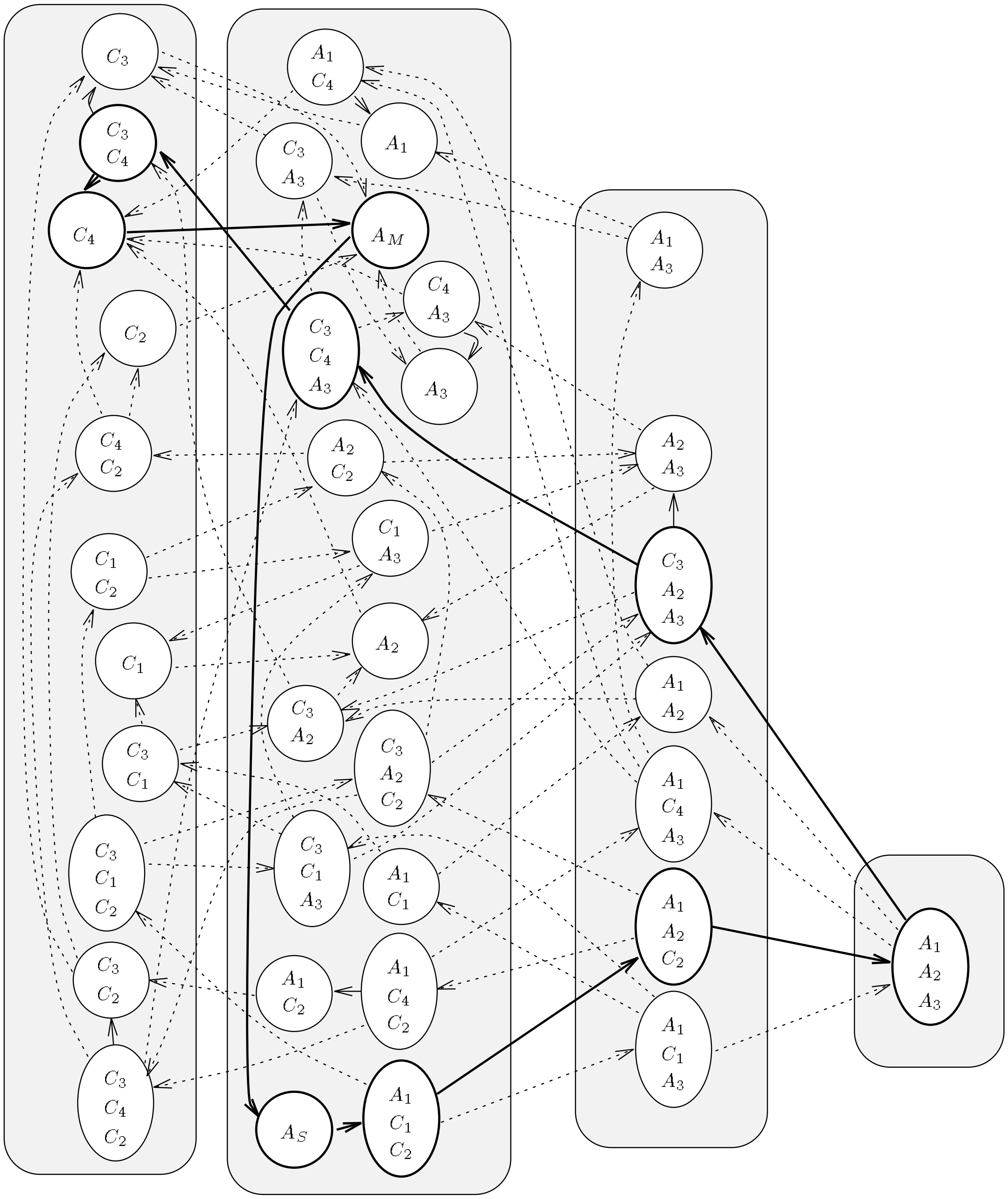
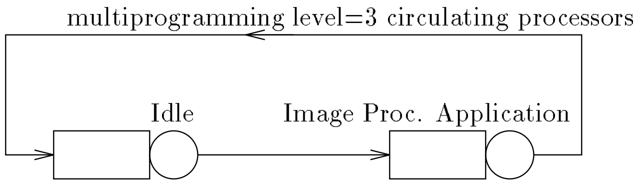


Figure 20: Markov Diagram for the Image Processing Application



$$\begin{array}{rr@{\qquad\qquad}rr} L_I(1) = & 9.0867 & L_A(1) = & 20.5504 \\ L_I(2) = & 20.5504 & L_A(2) = & 9.0867 \\ L_I(3) = & 0.3169 & L_A(3) = & 5.4061 \end{array}$$

Figure 21: Load Dependent CP Model for the Image Processing Application

do not all begin at the same time. Rather, the state transitions occur only between neighboring state partitions.

A sample execution path is illustrated by the thick arrows. The application begins executing in state $\langle A_S \rangle$, near the bottom of Figure 20. In state $\langle A_S \rangle$ one processor is busy. Task A_1 and communications C_1 and C_2 are initiated, and execution proceeds to state $\left\langle \begin{smallmatrix} A_1 \\ C_1 \\ C_2 \end{smallmatrix} \right\rangle$, where again one processor is busy. If it happens that communication C_1 completes first, then task A_2 will begin, and execution will proceed to state $\left\langle \begin{smallmatrix} A_1 \\ A_2 \\ C_2 \end{smallmatrix} \right\rangle$, and two processors will be busy. If communication C_2 completes next, then execution will proceed to state $\left\langle \begin{smallmatrix} A_1 \\ A_2 \\ A_3 \end{smallmatrix} \right\rangle$ (where all three processors are busy), and so on. In each state transition, exactly one communication or one task completes. After task A_M completes (in state $\langle A_M \rangle$), a new copy of the application is initiated, and execution begins again in state $\langle A_S \rangle$.

The image processing application can be modeled using the load dependent circulating processor model, using three circulating processors and an idle server, as shown in Figure 21. The measured service rates in Table 1 parameterize the Markov model in Figure 20. As before, the load dependent demands for the circulating processor model can be measured directly from the system or calculated from mean service time for each set of states of the Markov model. The mean load dependent demand for each number of processors is the mean service time weighted by the relative throughput (i.e., visit ratios) of each state. The mean load dependent demands are illustrated in Figure 21.

The circulating processor model yields an application throughput of 0.0283. The measured system throughput is 0.0282. As expected, since this application has multiprogramming level equal to one, the model matches the observed performance.

7. Conclusions and Future Work

The circulating processor model is a novel approach to the modeling of parallel systems. In general, models of parallel systems do not have efficient solution techniques. Product form queueing network models have efficient solution techniques. However, due to task forking and joining, gang scheduling, and barrier synchronization, many parallel applications violate the product form assumptions. The circulating processor model is an approximate product form model that can be applied to parallel systems captures certain parallel behavior accurately. The model is exact in certain cases and the approximation is good in other cases.

The equivalences that have been shown are useful from both a theoretical and a practical viewpoint. From a theoretical viewpoint, the comparison of the circulating processor model to the traditional circulating task model is promising and requires further study. The use of the parallelism shape to parameterize the circulating processor model is a direct application of a practical workload characterization of a parallel program. Traditional queueing network models are parameterized by low level measurements of system devices. These parameters do not correspond to any particular characterization of parallel applications.

From a practical viewpoint, the circulating processor model is exact for a single parallel application in the system. In this case, the model is easy to parameterize, since the parallelism shape is generally easy to obtain for an individual parallel application. Thus, the model serves as a good descriptive tool for the parallel system and its application. When the multiprogramming level greater than one, the greatest error between the load dependent circulating processor model and the actual system occurs at low multiprogramming levels. The smallest errors occur (i.e., the approximation is most accurate) when the multiprocessor is heavily loaded and demands at the multiprocessor are balanced.

There are certain disadvantages to the circulating processor model. First, the metrics obtained are not guaranteed to be either an upper or a lower bound. Second, the relative error can be as high as 100% in cases where the multiprocessor is lightly loaded or not balanced. A third disadvantage is the difficulty in using the model to make predictions. For example, suppose that the model for a system with multiprogramming level one is constructed using the parallelism shape of the application. If the communication medium (modeled as the idle server in the circulating processor model) is made faster, then the predicted effect on the load dependent demands is not clear. Not only will the idle time be affected, but the load dependent demands may be affected as well. Similarly, if the multiprogramming level is increased, the effects on the load dependent demands are not clear. Other modifications to the system also have unpredictable effects on the load dependent loadings. Thus, predicted interdependencies between the model parameters remains a difficult issue.

Future research includes:

1. Investigation of the effect of non-product form service disciplines on the accuracy of the model. Preliminary work indicates that the model has error in this case, but that the error is about 50% less than that obtained for the traditional circulating task model.

2. Investigation of the effectiveness of the circulating processor model for arbitrary task graphs. Only variations of fork-join behavior have been investigated in this paper.

3. Investigation of alternate processor allocation policies on the effectiveness of the basic model. Only fully sequential and fully parallel policies have been analyzed. Many other policies are possible, beginning with simple state policies in which each application is always allocated n processors, where $2 \leq n \leq N$. (N is the total number of available processors.)

4. Investigation of special cases in the which the model provides an upper bound on system performance. Preliminary results show that the model tends to be an upper bound in cases where the system is heavily loaded and balanced.

5. Investigation into the effective use of the model for prediction.

6. Experimental validation of the circulating processor model on a variety of parallel platforms, including both shared memory and distributed memory platforms.

References

[1] M. A. Marsan, G. Balbo, and G. Conte, *Performance Models of Multiprocessor Systems.* Cambridge, Mass: MIT Press, 1986.

[2] M. Reiser and S. S. Lavenberg, "Mean-value analysis of closed multichain queueing networks," *Journal of the Association for Computing Machinery*, vol. 27, pp. 313–322, Apr. 1980.

[3] P. Heidelberger and K. S. Trivedi, "Queueing network models for parallel processing with asynchronous tasks," *IEEE Transactions on Computers*, vol. C-31, pp. 1099–1109, Nov. 1982.

[4] P. Heidelberger and K. S. Trivedi, "Analytic queueing models for programs with internal concurrency," *IEEE Transactions on Computers*, vol. C-32, pp. 73–82, Jan. 1983.

[5] M. K. Vernon, E. D. Lazowska, and J. Zahorjan, "An accurate and efficient performance analysis technique for multiprocessor snooping cache-consistency protocols," in *Proceedings of the 15th Annual International Symposium on Computer Architecture*, pp. 308–317, May 1988.

[6] R. Nelson, D. Towsley, and A. N. Tantawi, "Performance analysis of parallel processing systems," *IEEE Transactions on Software Engineering*, vol. 14, pp. 532–540, Apr. 1988.

[7] R. Nelson, "A performance evaluation of a general parallel processing model," in *Proc., ACM Sigmetrics Conference on Measurement and Modeling of Computer Systems*, pp. 13–26, May 1990.

[8] K. C. Sevcik and S. Zhou, "Performance benefits and limitations of large NUMA multiprocessors," in *Performance '93* (G. Iazeolla and S. Lavenberg, eds.), pp. 183–204, Oct. 1993.

[9] K. C. Sevcik, "Characterization of parallelism in applications and their use in scheduling," in *Proc., ACM Sigmetrics Conference on Measurement and Modeling of Computer Systems*, pp. 171–180, May 1989.

[10] C.-S. Chang, R. Nelson, and D. D. Yao, "Optimal task scheduling on distributed parallel processors," in *Performance '93* (G. Iazeolla and S. Lavenberg, eds.), pp. 205–219, 1993.

[11] J. D. C. Little, "A proof of the queueing formula $L = \lambda W$," *Operations Research*, vol. 9, pp. 383–387, 1961.

[12] G. Franceschinis and R. R. Muntz, "Bounds for quasi-lumpable Markov chains," in *Performance '93* (G. Iazeolla and S. Lavenberg, eds.), pp. 220–244, 1993.

[13] Simulog Corp., *QNAP2 Reference Manual*, 1989.

[14] F. Baskett, K. M. Chandy, R. R. Muntz, and F. G. Palacios, "Open, closed, and mixed networks of queues with different classes of customers," *Journal of the Association for Computing Machinery*, vol. 22, no. 2, pp. 248–260, 1975.