Queueing Network Models for Parallel Processing of Task Systems: An Operational Approach

Victor W. K. Mak

Technical Report: CSL-TR-86-306

September 1986

The work described herein was supported by NASA Ames Research Center under contract NAG 2-248 and using facilities provided by NAGW 419.

Queueing Network Models for Parallel Processing of Task Systems: An Operational Approach

by

Victor W. K. Mak

Technical Report: **CSL-TR-86-306**

September **1986**

Computer Systems Laboratory

Departments of Electrical Engineering and Computer Science

Stanford University

Stanford, California 94305

Abstract

Computer performance modeling of possibly complex computations running on highly concurrent systems is considered. Earlier work in this area either dealt with a very simple program structure or resulted in methods with exponential complexity. An efficient procedure is developed to compute the performance measures for series-parallel-reducible task systems using queueing network models. The procedure is based on the concept of hierarchical decomposition and a new operational approach. Numerical results for three test cases are presented and compared to those of simulations.

Key Words and Phrases: Decomposition, Modeling, Operational Approach, Parallel Processing, Performance Evaluation, Performance Prediction, Queueing Network, Series-Parallel-Reducible, Task System.

Contents

List of Figures

List of Tables

1 Introduction

As interest in highly concurrent system research expands, a means for predicting the probable performance of application computation on such systems would be an important tool for evaluating the effectiveness of how these systems are being utilized. Unfortunately, most existing analytical approaches either are not suitable to study parallel systems, or are computationally intractable. Simulation approaches are also unrealistic if the systems being studied are much more capable than the systems supporting the simulation.

This report describes a computationally efficient procedure for predicting the probable performance of a possibly complex computation on a highly concurrent system using queueing network models. The computation is specified by a series-parallel-reducible task system which consists of a set of tasks related by a deterministic precedence graph. Each task is characterized by its expected total loadings (or service demand) on the resources of the computer system. The procedure presented here can be used to determine key performance measures such as mean execution time for each task, mean completion time for the task system, and utilizations for the resources in the computer system.

Heidelberger and Trivedi [HT82,HT83] have used analytic queueing models to predict performance for programs with internal concurrency. In [HT82], they considered systems in which a parent task subdivides into two or more tasks which require no synchronization with the parent task. In [HT83], a task can spawn two or more concurrent tasks but has to wait for their completions before it can proceed. Both papers considered only very simple task systems. The task systems considered in this report are much more complicated and include all concurrent task systems programmed using block-oriented constructs like ***cobegin, coend***, *DOALL*, ***fork, join,*** etc..

Thomasian and Bay [TB83] considered a more general task system with deterministic precedence constraints expressed as a directed acyclic graph. Their method is based on the concept of hierarchical decomposition [Cou77]. At the higher level, a Markov chain corresponding to the transitions among system states is generated. **At** the lower level, the transition rates among the states are computed using a queueing network solver. Their method is a significant improvement over the exact method using Markov chain alone. However, the number of states in their method is still too high for large systems. For a system with N tasks, in the worst case, there may be as many as 2^N states in their higher level Markov chain.

Mohan in his thesis [Moh84] considered similar task structures as [TB83]. He also made use of queueing network models to find throughput of system states in the lower level. In the higher level, he used simulation to determine the mean completion time for the system by tracing different execution paths. For a system with N tasks, there

may also be as many as 2^N execution paths. Hence, it requires at least the same complexity as the method in [TB83].

The task system model as used in the method described later is a subset of that considered in [TB83] : it has to be series-parallel-reducible. This method is also based on the concept of hierarchical decomposition, but instead of forming a Markov chain and computing state probabilities, an operational approach is used to determine performance measures directly from measurable quantities. This approach reduces the complexity of the method to be polynomial.

In section 2 of this report, the task system and computer system models used in this procedure are described. The operational approach is presented in section 3. The three key steps in the procedure: estimation of contention, estimation of task execution time, and estimation of system completion time, are discussed in sections **3.1, 3.2,** and 3.3, respectively. This procedure has been validated through simulation. Three test cases are presented and the numerical results are compared to those of simulations in section 4.

2 Task System and Computer System Models

The task system model used in this report follows closely with the one defined in [TB83] except with one additional constraint: the precedence graph is series-parallel-reducible (see Fig. 1). A task system is specified by a 3-tuple (T, $[\prec]$, $[D_{nk}]$) as follows:

1. $T = (T_1, T_2, \ldots, T_N)$ is a set of tasks to be executed on the computer system. Except for queueing effect, the tasks execute independent of each other.

2. $[\prec]$ is a partial order defined on T specifying deterministic precedence constraints. $T_i \prec T_j$ means that T_i must be completed before T_j can begin. Only series-parallel-reducible directed acyclic graphs are considered.

3. $[D_{nk}]$ is an N $\times$ K matrix, such that D_{nk} is the service demand (expected total loadings) of task n on resource k.

The computer system is modeled as a central server queueing network (see Fig. 2). Each resource in the system is modeled as a service center. During the course of execution of a task system, the computer system processes different combinations of tasks according to the precedence constraints until all tasks are completed. Each task combination in progress at the computer system can be represented as a closed queueing network with multiple job types. Tasks are assumed to execute as soon as their precedence constraints are satisfied. Other task scheduling disciplines are also possible but are not considered in this report. To simplify the analysis, the queueing network model is assumed to be separable and has a product-form solution [LZGS84].

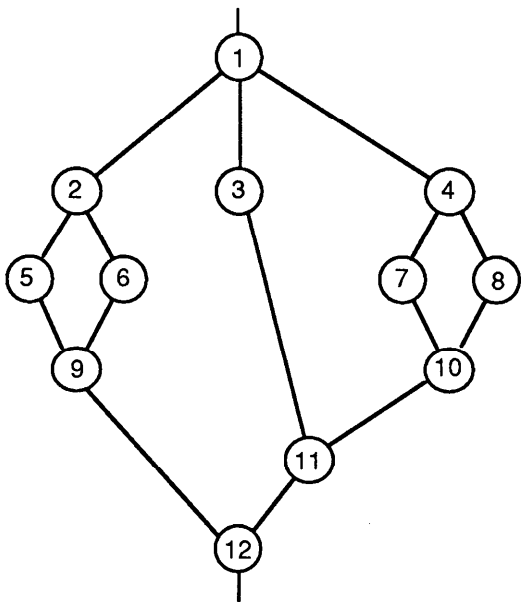

Figure **1:** A Series-Parallel-Reducible Task System

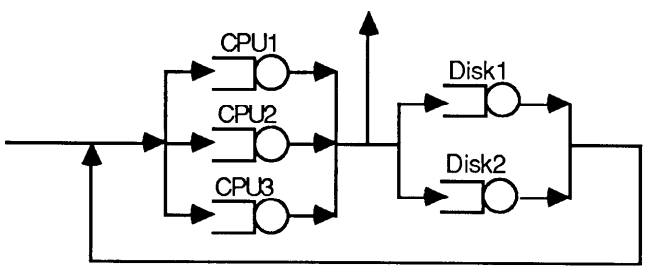

Figure 2: A Concurrent Computer System

3 An Operational Approach

One of the most difficult problems in predicting performance of a concurrent task system is that service demands on system resources are probabilistic. This results in probabilistic task execution times and hence different possible execution paths for the same task system. Each execution path has its own path probability and a corresponding system completion time. To determine exactly the mean system completion time, all these possible execution paths must be taken into account. Earlier efforts have tried to use either mathematical analysis [TB83] or simulation [Moh84] to find all possible execution paths and their path probabilities. Although accurate, those methods are infeasible for large task systems due to their exponential complexity.

The main difference of the operational approach described here compared with earlier efforts is in estimating task mean execution times and system mean completion time directly without tracing all possible execution paths. In [DB78], Denning and Buzen have used an operational approach for the analysis of queueing network. Instead of using stochastic models to compute performance measures, algebraic relationships among measurable quantities (such as throughput and response time) are derived. Using these relationships, performance measures of queueing network models can be computed directly without resorting back to stochastic models. Likewise, in this procedure, some algebraic relationships are derived among measurable quantities, such as task mean initiation times, task mean execution times, system mean completion time, and task service demands. Using this set of relationships, the performance of the system can be estimated directly without tracing all execution paths.

This approach is based on the concept of hierarchical decomposition [Cou77], i.e., the system is assumed to reach equilibrium between task initiation and completion instants. As mentioned in the last section, each task combination in progress is represented as a closed queueing network with multiple job types. Resource utilizations and mean queue lengths can be assumed to reach their steady state values during this interval because of the decomposition approximation.

The execution time of a task consists of two components: its actual service time by the resources in the computer system, and the waiting time spent at the queues. The former is the service demand and is given in $[D_{nk}]$. The later depends on the amount of contention experienced in the system, which in turn depends on the loadings of the system during the task's execution interval. By comparing with mean initiation times and mean completion times of other tasks in the system, the amount of contention experienced by a task during its execution interval can be estimated. With the amount of contention, the mean execution time of a task can be estimated. Knowing all task mean execution times, the mean initiation time for each task and the mean completion time for the task system can be estimated by reducing the series-parallel precedence

graph. With the new set of mean initiation times and mean execution times, a better estimate of the amount of contention can be obtained. The iterative process continues until the estimate of the mean completion time converges.

Following is the outline of the iterative procedure:

1. Initiate the system as contention free, i.e., task execution times equal to service demands.

2. Estimate amount of contention experienced by each task during its execution interval.

3. Estimate mean execution times for all tasks using the contention found in the previous step.

4. Estimate mean initiation times for all tasks and mean completion time for the task system by reducing the series-parallel precedence graph.

5. Repeat steps 2, 3, and 4 until successive estimates of mean completion time converge to within some tolerance (say **0.1%).**

Steps 2, 3, and **4** are the key components of this iterative procedure. They will be discussed in more detail in the next three sections. The complexity of each iteration is $O(N^3K)$ where N is the number of tasks and K is the number of resources in the system.

3.1 Estimation of Contention

The amount of contention experienced by a task during its execution interval depends on the number of tasks competing with it for system resources. If there is no other task executing concurrently with it, it can obtain service immediately at every resource, and its execution time is just equal to its service demand. On the other hand, if there are many concurrent tasks, each task spends more time waiting at the queues for service. The amount of contention can be estimated in terms of the arrival instant queue length, A_{ik}, which is the mean queue length (including the one in service) as seen by task i when it first arrives at resource k. If Q_{nk} is the steady state queue length of task n at resource k, and W_{in} is the fraction of task i's execution interval such that tasks i and n overlap with each other, then A_{ik} can be computed as follows:

$$A_{ik} = \sum_{n=1, n \neq i}^{N} Q_{nk} \; W_{in} \qquad (1)$$

Q_{nk} is equal to the fraction of E_{nk}, the time task n spent in resource k, during E_n, n's execution interval,

$$Q_{nk} = \frac{E_{nk}}{E_n} \tag{2}$$

Let p_{in} be the probability that tasks i and n overlap, and d_{in} be the mean duration of the overlapping interval if tasks i and n overlap, then

$$W_{in} = \frac{p_{in}\, d_{in}}{E_i} \tag{3}$$

Substituting equations 2 and 3 into equation 1, we get

$$A_{ik} = \frac{1}{E_i} \sum_{n=1, n\neq i}^{N} \frac{E_{nk}\ p_{in}\ d_{in}}{E_n} \tag{4}$$

Consider task i with mean initiation time, I_i, and mean completion time C_i, and task n with mean initiation time, I_n, and mean completion time C_n. Tasks i and n will overlap with each other unless i initiates after n has completed, or vice versa. Therefore,

$$p_{in} = 1 - \Pr(C_i < I_n) - \Pr(C_n < I_i) \tag{5}$$

If A and B are two independent non-negative continuous random variables, then

$$\begin{aligned} \Pr(A < B) &= \int_0^\infty \Pr(B > x) f_A(x)\, dx \\ &= \int_0^\infty [1 - F_B(x)] f_A(x)\, dx \end{aligned} \tag{6}$$

where $f_A(x)$ is the probability density function of A and $F_B(x)$ is the probability distribution function of B.

Equation 6 involves both the distribution and density functions of A and B, and is very expensive to compute. It is computationally more economical to consider only the means and variances of the initiation and completion times. Assume that A and B are both Erlang distributed with parameters (λ_A, r_A) and (λ_B, r_B), respectively. Substituting these parameters into equation 6, and after simplification (see Appendix A),

$$\Pr(A < B) = \left(\frac{\lambda_A}{\lambda_A + \lambda_B}\right)^{r_A} \sum_{k=0}^{r_B - 1} \left(\frac{\lambda_B}{\lambda_A + \lambda_B}\right)^{k} \frac{(r_A + k - 1)!}{(r_A - 1)!\, k!} \tag{7}$$

Notice that the mean initiation and completion times of tasks i and n in equation 5 are not independent of each other. Although tasks execute independently of each other, they may still have a common ancestor chain in the precedence graph. If t is

the duration of their common ancestor chain, then it has to be subtracted from their mean initiation and completion times before equation 7 can be used to compute the probabilities.

Task execution times are assumed to be exponentially distributed. This is asymptotically true if the task cycles back through the queueing network with probability p and exits with probability $(1 - p)$, and if p is close to 1 [LS79]. Assume that λ_i $(= \frac{1}{E_i})$ and λ_n $(= \frac{1}{E_n})$ be parameters of tasks i and n's execution time distributions respectively. When tasks i and n overlap, the overlapping region will end when either task terminates. Since i and n are two independent tasks with exponentially distributed execution times, using the memoryless property of the exponential distribution function, the duration of the overlapping region is also exponentially distributed with parameter equals to $\lambda_i + \lambda_n$. Therefore, the mean duration of the overlapping region is,

$$d_{in} = \frac{1}{\lambda_i + \lambda_n} \tag{8}$$

In summary, using equations 4, 5, and 8, we can determine the arrival instant queue length for each task at each resource in the system, and hence the amount of contention experienced by each task.

3.2 Estimation of Task Mean Execution Time

The mean execution time of task i at resource k can be estimated using the following equation [RL80],

$$E_{ik} = D_{ik}(1 + A_{ik}) \tag{9}$$

D_{ik} is the service demand of task i on resource k and is given as an input parameter of the task system. A_{ik}, the arrival instant queue length, is computed as described in the last section. The mean execution time for task i is then,

$$E_i = \sum_{k=1}^{K} D_{ik}(1 + A_{ik}) \tag{10}$$

3.3 Estimation of System Mean Completion Time

After finding the mean execution time for each task, the next step is to make use of the series-parallel-reducible precedence graph to determine mean initiation times and mean completion time for the system. Because of the special structure of the precedence graph, these values can be estimated by reducing the series-parallel graph, i.e., finding the equivalent execution time for tasks in series and in parallel [Kle85]. The whole task system can be reduced to a single equivalent execution time, which is

the mean completion time for the system. Mean initiation time for a task is actually the mean completion time for the task subsystem consisting of all of its ancestors.

Execution time for a task is not a fixed number, but a random variable with some distribution. As explained in section 3.1, task execution time is assumed to have an exponential distribution. Task execution times are also assumed to be independent of each other except for queueing effect.

Assume that tasks i and n have probability distribution functions F_i and F_n, and density functions f_i and f_n, respectively. If i and j are in series, then the equivalent probability density function of their series combination will be the convolution of the two density functions,

$$f_{eq} = f_i * f_j \tag{11}$$

If i and j are in parallel, then the equivalent probability distribution function of their parallel combination will be the product of the two distribution functions,

$$F_{eq} = F_i\, Fj \tag{12}$$

However this approach is very expensive computationally, since convolution and integration have to be performed numerically. A more economical way to find the series and parallel equivalences is to consider only the means and variances of distributions.

Let E_i and V_i be the mean and variance of the execution time of task subsystem i, respectively. If task subsystems i and j are in series, then using the Central Limit Theorem, the equivalent mean and variance of the series combination are:

$$E_{eq} = E_i + Ej \tag{13}$$

and

$$V_{eq} = V_i + V_j \tag{14}$$

For task subsystems in parallel, the determination of the equivalent mean and variance is a little bit more complicated than the serial case. As in section 3.1, assume that the execution time of task subsystem i is Erlang distributed with parameters (λ_i, r_i), and for j, (λ_j, r_j). These parameters are substituted into equation 12 to solve for F_{eq}, which can then be used to solve for the equivalent mean and variance (see Appendix B). After simplification, the equivalent mean and variance can be shown to be:

$$\begin{aligned} E_{eq} &= E_i + Ej - \frac{\lambda_i^{r_i}}{(\lambda_i+\lambda_j)^{r_i+1}} \sum_{k=0}^{r_j-1} \left(\frac{\lambda_j}{\lambda_i+\lambda_j}\right)^k \frac{(r_i+k)!}{(r_i-1)!\,k!} \\ &\quad - \frac{\lambda_j^{r_j}}{(\lambda_i+\lambda_j)^{r_j+1}} \sum_{k=0}^{r_i-1} \left(\frac{\lambda_i}{\lambda_i+\lambda_j}\right)^k \frac{(r_j+k)!}{(r_j-1)!\,k!} \end{aligned} \tag{15}$$

$$V_{eq} = E_i^2 + E_j^2 + V_i + V_j - E_{eq}^2$$
$$- \frac{\lambda_i^{r_i}}{(\lambda_i+\lambda_j)^{r_i+2}} \sum_{k=0}^{r_j-1} \left(\frac{\lambda_j}{\lambda_i+\lambda_j}\right)^k \frac{(r_i+k+1)!}{(r_i-1)!\,k!} \quad (16)$$
$$- \frac{\lambda_j^{r_j}}{(\lambda_i+\lambda_j)^{r_j+2}} \sum_{k=0}^{r_i-1} \left(\frac{\lambda_i}{\lambda_i+\lambda_j}\right)^k \frac{(r_j+k+1)!}{(r_j-1)!\,k!}$$

Equations 13 to 16 are used to reduce the precedence graph to obtain mean initiation times and mean completion time for the task system. This set of newly computed values can then be used to estimate the contention in the system.

The mean completion time, C, can also be used to determine the utilizations and mean queue lengths of the resources in the system. For a system with N tasks, the utilization of resource ***k*** is

$$U_k = \frac{1}{C} \sum_{i=1}^{N} D_{ik} \quad (17)$$

The mean queue length of resource ***k*** is

$$Q_k = \frac{1}{C} \sum_{i=1}^{N} E_{ik} \quad (18)$$

4 Test Cases and Validation

In this section, three test cases are used to evaluate the accuracy of the above procedure by comparing the numerical results to those of simulations. In the above method, since the queueing network is separable and has a product-form solution, the specification of the task system model requires only the expected total loading (the product of the mean number of visits to the resource and the mean service time of the resource) of each task on each resource. However, for simulation purpose, both the expected total loading and the mean service time of the resource have to be specified. The simulator was written using CSIM, a C-based process oriented simulation language developed by Herb Schwetman at MCC [Sch86].

The first test case is a 6-task system running on a uniprocessor system. This is the same example used in [TB83] and is chosen for comparison purpose. Both the second and the third test cases use a shared memory multiprocessor model in which there are 4 processors and 4 memory units connected by some interconnection network. The second test case has a master-slave task structure that corresponds to computations written in concurrent programming languages with constructs like *fork* and *join*. The third test case has a partitioning task structure which is common for

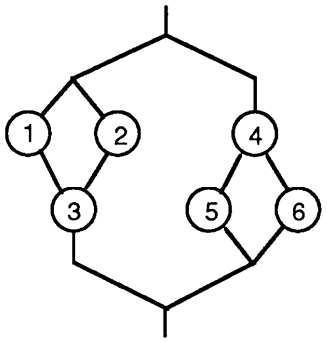

Figure 3: 6-Task System

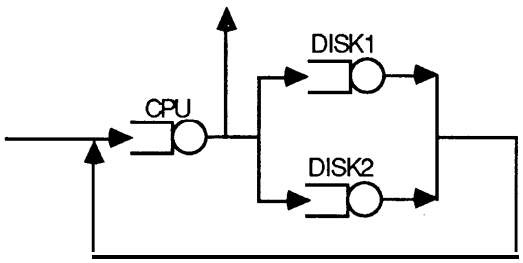

Figure 4: Uniprocessor System

divide-and-conquer algorithms. All simulations for these three test cases were run for 5000 replications.

4.1 6-Task System

The first test case is a 6-task system (Fig. 3) as used in [TB83]. The uniprocessor model (Fig. 4) used consists of one CPU (service time 0.020) and two identical disks (service time 0.040). The service demands of the 6-task system on the uniprocessor is shown in Table **1.** The operational method takes only 3 iterations to converge to the final estimation. The results are shown in Table 2.

4 . 2 Master-Slave System

The second test case has a master-slave task structure (Fig. 5) that is common for computations written in concurrent languages with constructs like *fork* and join. The master task spawns off a number of slave tasks and wait for their completions before proceeding. Two cycles of this synchronization pattern are shown in this test case. The computer system model used is a shared memory multiprocessor system (Fig 6). It has 4 processors (service time 0.020) and 4 memory units (service time 0.020) connected

Table **1:** Service Demands of 6-Task System on Uniprocessor

Task	CPU	DISK1	DISK2
1	0.420	0.400	0.400
2	0.420	0.400	0.400
3	0.620	0.600	0.600
4	0.620	0.600	0.600
5	0.420	0.400	0.400
6	0.420	0.400	0.400

Table 2: Results of 6-Task System

	Operational	Simulation	Error
c	6.235	6.140	1.55%
E_1	1.825	1.795	1.67%
E_2	1.825	1.876	-2.72%
E_3	2.233	2.222	0.50%
E_4	2.470	2.469	0.04%
E_5	1.716	1.705	0.65%
E_6	1.716	1.720	-0.23%
I_1, I_2, I_4	0.000	0.000	0.00%
I_3	2.738	2.643	3.59%
I_5, I_6	2.470	2.469	0.04%

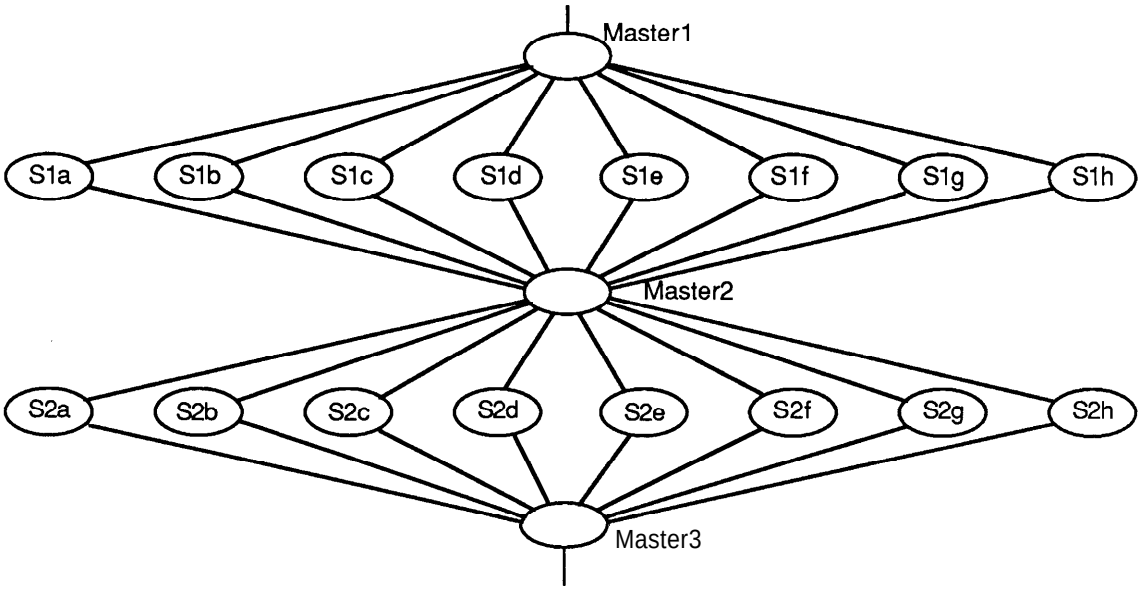

Figure *5:* Master-Slave System

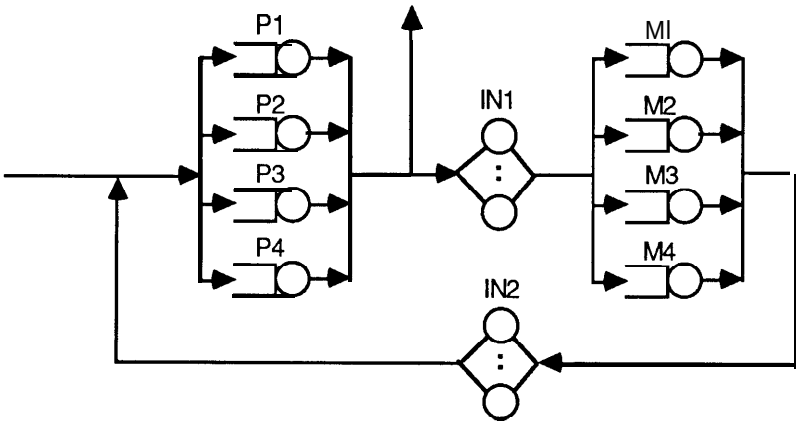

Figure *6:* Shared Memory Multiprocessor System

Table **3:** Service Demands of Master-Slave System on Multiprocessor

Task	P1	P2	P3	P4	IN1	IN2	M1	M2	M3	M4
Master1	0.520				0.025	0.025	0.125	0.125	0.125	0.125
Sla	0.620				0.030	0.030	0.150	0.150	0.150	0.150
Slb	0.520				0.025	0.025	0.125	0.125	0.125	0.125
S1c		0.620			0.030	0.030	0.150	0.150	0.150	0.150
Sld		0.520			0.025	0.025	0.125	0.125	0.125	0.125
Sle			0.620		0.030	0.030	0.150	0.150	0.150	0.150
Slf			0.520		0.025	0.025	0.125	0.125	0.125	0.125
S1g				0.620	0.030	0.030	0.150	0.150	0.150	0.150
Slh				0.520	0.025	0.025	0.125	0.125	0.125	0.125
Master2	0.520				0.025	0.025	0.125	0.125	0.125	0.125
S2a	0.620				0.030	0.030	0.150	0.150	0.150	0.150
S2b	0.620				0.030	0.030	0.150	0.150	0.150	0.150
S2c		0.620			0.030	0.030	0.150	0.150	0.150	0.150
S2d		0.620			0.030	0.030	0.150	0.150	0.150	0.150
S2e			0.520		0.025	0.025	0.125	0.125	0.125	0.125
S2f			0.520		0.025	0.025	0.125	0.125	0.125	0.125
S2g				0.520	0.025	0.025	0.125	0.125	0.125	0.125
S2h				0.520	0.025	0.025	0.125	0.125	0.125	0.125
Master3	0.520				0.025	0.025	0.125	0.125	0.125	0.125

by **2** interconnection networks (modeled as delay centers with delay time **0.001).** The service demands are shown in Table **3.** Three iterations are required by the operational method to converge to the final estimates. The results are shown in Table 4.

4.3 Divide-And-Conquer System

The third test case has a partitioning task structure (Fig. 7) that is common for computations using divide-and-conquer algorithms. The computer system model used is the same as the last test case as shown in Fig. **6.** The service demands of this divide-and-conquer system on the multiprocessor is shown in Table **5.** Only four iterations are required to converge to the final estimates. The results are shown in Table **6.**

Table 4: Results of Master-Slave System

	Operational	Simulation	Error
C	10.452	10.732	-2.61%
$E_{\mathbf{Master1}}$	1.070	1.059	1.04%
$E_{\mathbf{S1a}}$	1.667	1.642	1.52%
$E_{\mathbf{S1b}}$	1.430	1.390	2.88%
$E_{\mathbf{S1c}}$	1.667	1.646	1.28%
$E_{\mathbf{S1d}}$	1.430	1.401	2.07%
$E_{\mathbf{S1e}}$	1.667	1.643	1.46%
$E_{\mathbf{S1f}}$	1.430	1.427	0.21%
$E_{\mathbf{S1g}}$	1.667	1.638	1.77%
$E_{\mathbf{S1h}}$	1.430	1.425	0.35%
$E_{\mathbf{Master2}}$	1.070	1.068	0.19%
$E_{\mathbf{S2a}}$	1.677	1.659	1.08%
$E_{\mathbf{S2b}}$	1.677	1.642	2.13%
$E_{\mathbf{S2c}}$	1.677	1.632	2.76%
$E_{\mathbf{S2d}}$	1.677	1.666	0.66%
$E_{\mathbf{S2e}}$	1.421	1.432	-0.77%
$E_{\mathbf{S2f}}$	1.421	1.398	1.65%
$E_{\mathbf{S2g}}$	1.421	1.387	2.45%
$E_{\mathbf{S2h}}$	1.421	1.430	-0.63%
$E_{\mathbf{Master3}}$	1.070	1.075	-0.47%
$I_{\mathbf{Master1}}$	0.000	0.000	0.00%
$I_{\mathbf{S1}}$	1.070	1.059	1.04%
$I_{\mathbf{Master2}}$	4.712	4.814	-2.12%
$I_{\mathbf{S2}}$	5.782	5.882	-1.70%
$I_{\mathbf{Master3}}$	9.382	9.657	-2.85%

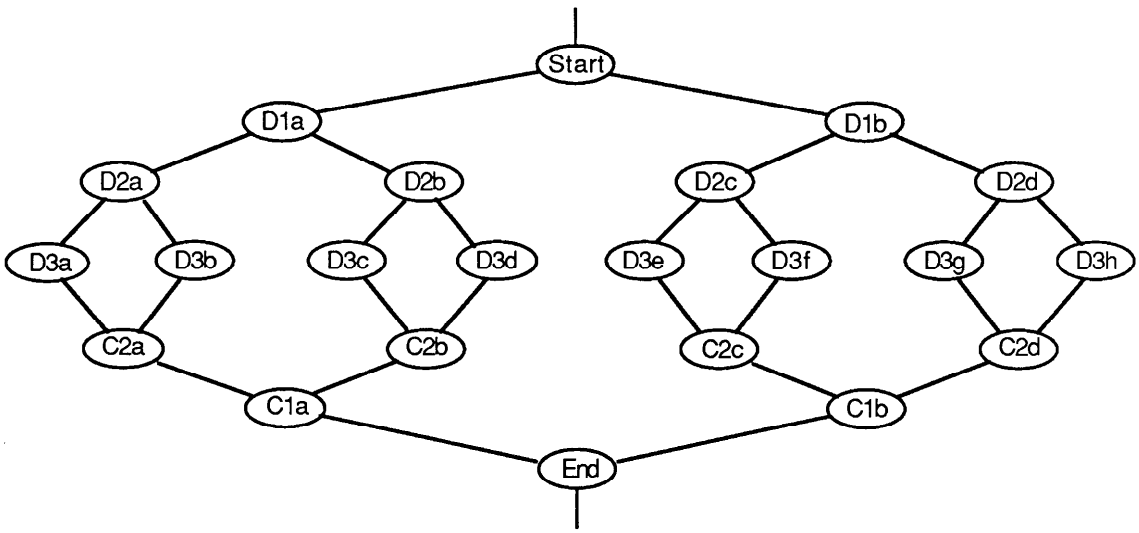

Figure 7: Divide-And-Conquer System

5 Summary

This report has presented an efficient procedure for predicting performance of series-parallel-reducible task system. The complexity of each iteration of the method is $O(N^3K)$, where N is the number of tasks and K is the number of resources in the system. This complexity is a very significant reduction over earlier efforts which either dealt with very simple task structures, or had methods with exponential complexity. By using an operational approach, the performance measures of the system can be estimated directly from measurable quantities without tracing all possible execution paths. The procedure is very accurate as demonstrated by the three test cases presented. In fact, the maximum observed error in the estimates from the procedure are within 4% of simulation results in all three cases. The procedure also has a high convergent rate: for the test cases presented, convergence can be achieved in less than 5 iterations.

Table 5: Service Demands of Divide-And-Conquer System on Multiprocessor

Task	P1	P2	P3	P4	IN1	IN2	M1	M2	M3	M4
Start	0.520				0.025	0.025	0.125	0.125	0.125	0.125
D1a	0.620				0.030	0.030	0.150	0.150	0.150	0.150
D1b			0.620		0.030	0.030	0.150	0.150	0.150	0.150
D2a	0.620				0.030	0.030	0.150	0.150	0.150	0.150
D2b		0.620			0.030	0.030	0.150	0.150	0.150	0.150
D2c			0.620		0.030	0.030	0.150	0.150	0.150	0.150
D2d				0.620	0.030	0.030	0.150	0.150	0.150	0.150
D3a	0.620				0.030	0.030	0.150	0.150	0.150	0.150
D3b	0.620				0.030	0.030	0.150	0.150	0.150	0.150
D3c		0.620			0.030	0.030	0.150	0.150	0.150	0.150
D3d		0.620			0.030	0.030	0.150	0.150	0.150	0.150
D3e			0.620		0.030	0.030	0.150	0.150	0.150	0.150
D3f			0.620		0.030	0.030	0.150	0.150	0.150	0.150
D3g				0.620	0.030	0.030	0.150	0.150	0.150	0.150
D3h				0.620	0.030	0.030	0.150	0.150	0.150	0.150
C2a	0.420				0.020	0.020	0.100	0.100	0.100	0.100
C2b		0.420			0.020	0.020	0.100	0.100	0.100	0.100
C2c			0.420		0.020	0.020	0.100	0.100	0.100	0.100
C2d				0.420	0.020	0.020	0.100	0.100	0.100	0.100
C1a	0.420				0.020	0.020	0.100	0.100	0.100	0.100
C1b			0.420		0.020	0.020	0.100	0.100	0.100	0.100
End	0.520				0.025	0.025	0.125	0.125	0.125	0.125

Table 6: Results of Divide-And-Conquer System

	Operational	Simulation	Error
C	12.002	11.994	0.07%
&tart	1.070	1.083	-1.20%
E_{D1a}	1.407	1.401	0.43%
E_{D1b}	1.407	1.363	3.23%
E_{D2a}	1.504	1.503	0.07%
E_{D2b}	1.504	1.494	0.67%
E_{D2c}	1.504	1.479	1.69%
E_{D2d}	1.504	1.482	1.48%
E_{D3a}	1.653	1.665	-0.72%
E_{D3b}	1.653	1.640	0.79%
E_{D3c}	1.653	1.625	1.72%
E_{D3d}	1.653	1.640	0.79%
E_{D3e}	1.653	1.659	-0.36%
E_{D3f}	1.653	1.653	0.00%
E_{D3g}	1.653	1.642	0.67%
E_{D3h}	1.653	1.660	-0.42%
E_{C2a}	0.972	0.940	3.40%
E_{C2b}	0.972	0.955	1.78%
E_{C2c}	0.972	0.944	2.97%
E_{C2d}	0.972	0.936	3.85%
E_{C1a}	0.909	0.891	2.02%
E_{C1b}	0.909	0.900	1.00%
E_{End}	1.070	1.069	0.09%
I_{Start}	0.000	0.000	0.00%
I_{D1a}, I_{D1b}	1.070	1.083	-1.20%
I_{D2a}, I_{D2b}	2.477	2.484	-0.28%
I_{D2c}, I_{D2d}	2.477	2.446	1.27%
I_{D3a}, I_{D3b}	3.980	3.988	-0.20%
I_{D3c}, I_{D3d}	3.980	3.978	0.05%
I_{D3e}, I_{D3f}	3.980	3.925	1.40%
I_{D3g}, I_{D3h}	3.980	3.928	1.32%
I_{C2a}	6.460	6.373	1.37%
I_{C2b}	6.460	6.330	2.05%
I_{C2c}	6.460	6.312	2.34%
I_{C2d}	6.460	6.300	2.54%
I_{C1a}	8.566	8.550	0.19%
I_{C1b}	8.566	8.522	0.52%
I_{End}	10.932	10.925	0.06%

Appendices

A Derivation of Equation 7

From equation **6,**

$$\Pr(A < \mathrm{B}) = \int_{0}^{\infty} [1 - F_B(x)] f_A(x) dx$$

Assume that A and B are two independent Erlang distributed random variables with parameters (λ_A, r_A) and (λ_B, r_B) respectively, then,

$$f_A(x) = \frac{\lambda_A^{r_A} x^{r_A - 1} e^{-\lambda_A x}}{(r_A - 1)!}$$

and

$$F_B(x) = 1 - \sum_{k=O}^{r_B - 1} \frac{(\lambda_B x)^k}{k!} e^{-\lambda_B x}$$

Substituting the above two equations into equation **6,**

$$\Pr(A < B) = \int_{0}^{\infty} \left[\sum_{k=O}^{r_B - 1} \frac{(\lambda_B x)^k}{k!} e^{-\lambda_B x} \right] \left[\frac{\lambda_A^{r_A} x^{r_A - 1} e^{-\lambda_A x}}{(r_A - 1)!} \right] dx$$

$$= \int_{0}^{\infty} \left[\sum_{k=0}^{r_B - 1} \frac{\lambda_B^k \lambda_A^{r_A} x^{r_A + k - 1}}{(r_A - 1)!\, k!} e^{-(\lambda_A + \lambda_B) x} \right] dx$$

$$= \sum_{k=0}^{r_B - 1} \int_{0}^{\infty} \frac{\lambda_B^k \lambda_A^{r_A} x^{r_A + k - 1}}{(r_A - 1)!\, k!} e^{-(\lambda_A + \lambda_B) x} dx$$

$$= \sum_{k=0}^{r_B - 1} \frac{\lambda_B^k \lambda_A^{r_A}}{(r_A - 1)!\ k!} \int_{0}^{\infty} x^{r_A + k - 1} e^{-(\lambda_A + \lambda_B) x} dx$$

$$= \sum_{k=0}^{r_B - 1} \frac{\lambda_B^k \lambda_A^{r_A}}{(r_A - 1)!\ k!} \frac{(r_A + k - 1)!}{(\lambda_A + \lambda_B)^{r_A + k}}$$

$$= \left(\frac{\lambda_A}{\lambda_A + \lambda_B} \right)^{r_A} \sum_{k=0}^{r_B - 1} \left(\frac{\lambda_B}{\lambda_A + \lambda_B} \right)^k \frac{(r_A + k - 1)!}{(r_A - 1)!\, k!}$$

B Derivation of Equations 15 and 16

From equation 12, the equivalent probability distribution function of the parallel combination is,

$$F_{eq} = F_i \, F_j$$

Therefore, the equivalent probability density function is,

$$f_{eq} = f_i \, F_j + f_j \, F_i$$

Assume that i and j are two independent Erlang distributed random variables with parameters (λ_i, r_i) and (λ_j, r_j) respectively. Substituting these parameters into the above equation,

$$\begin{aligned} f_{eq} &= \frac{\lambda_i^{r_i} \, x^{r_i-1} \, e^{-\lambda_i x}}{(r_i-1)!} \left[1 - \sum_{k=O}^{r_j-1} \frac{(\lambda_j \, x)^k}{k!} \, e^{-\lambda_j x}\right] \\ &\quad + \frac{\lambda_j^{r_j} \, x^{r_j-1} \, e^{-\lambda_j x}}{(r_j-1)!} \left[1 - \sum_{k=0}^{r_i-1} \frac{(\lambda_i \, x)^k}{k!} \, e^{-\lambda_i x}\right] \\ &= f_i + f_j - \sum_{k=0}^{r_j-1} \frac{\lambda_i^{r_i} \, \lambda_j^k \, x^{r_i-1+k}}{(r_i-1)! \, k!} e^{-(\lambda_i+\lambda_j)x} \\ &\quad - \sum_{k=O}^{r_i-1} \frac{\lambda_3^{r_j} \, \lambda_i^k \, x^{r_j-1+k}}{(r_j-1)! \, k!} e^{-(\lambda_i+\lambda_j)x} \\ &= f_i + f_j - S_1 - S_2 \end{aligned}$$

The equivalent mean of the parallel combination is,

$$\begin{aligned} E_{eq} &= \int_0^\infty x f_{eq} dx \\ &= \int_0^\infty x(f_i + f_j - S_1 - S_2) dx \\ &= \int_0^\infty x f_i dx + \int_0^\infty x f_j dx - \int_0^\infty x S_1 dx - \int_0^\infty x S_2 dx \\ &= E_i + E_j - \int_0^\infty x S_1 dx - \int_0^\infty x S_2 dx \end{aligned}$$

Consider the third term of the above equation,

$$\int_0^\infty xS_1dx = \int_0^\infty x \sum_{k=0}^{r_j-1} \frac{\lambda_i^{r_i}\,\lambda_j^k\,x^{r_i-1+k}}{(r_i-1)!\,k!} e^{-(\lambda_i+\lambda_j)x}dx$$

$$= \lambda_i^{r_i} \sum_{k=0}^{r_j-1} \frac{\lambda_j^k}{(r_i-1)!\,k!} \int_0^\infty x^{r_i+k}\, e^{-(\lambda_i+\lambda_j)x}dx$$

$$= \lambda_i^{r_i} \sum_{k=0}^{r_j-1} \frac{\lambda_j^k}{(r_i-1)!\,k!} \frac{(r_i+k)!}{(\lambda_i+\lambda_j)^{r_i+k+1}}$$

$$= \frac{\lambda_i^{r_i}}{(\lambda_i+\lambda_j)^{r_i}} \sum_{k=0}^{r_j-1} \left(\frac{\lambda_j}{\lambda_i+\lambda}\right)^k \frac{(r_i+k)!}{(r_i-1)!\,k!}$$

Similarly,

$$\int_0^\infty xS_2dx = \frac{\lambda_j^{r_j}}{(\lambda_i+\lambda_j)^{r_j+1}} \sum_{k=0}^{r_i-1} \left(\frac{\lambda_i}{\lambda_i+\lambda_j}\right)^k \frac{(r_j+k)!}{(r_j-1)!\,k!}$$

Therefore, the equivalent mean of the parallel combination is

$$E_{eq} = E_i + E_j - \frac{\lambda_i^{r_i}}{(\lambda_i+\lambda_j)^{r_i}} \sum_{k=0}^{r_j-1} \left(\frac{\lambda_j}{\lambda_i+\lambda_j}\right)^k \frac{(r_i+k)!}{(r_i-1)!\,k!}$$

$$- \frac{\lambda_j^{r_j}}{(\lambda_i+\lambda_j)^{r_j}} \sum_{k=0}^{r_i-1} \left(\frac{\lambda_i}{\lambda_i+\lambda}\right)^k \frac{(r_j+k)!}{(r_j-1)!\,k!}$$

The equivalent variance of the parallel combination is

$$V_{eq} = \int_0^\infty x^2 f_{eq}dx - E_{eq}^2$$

$$= \int_0^\infty x^2 f_i dx + \int_0^\infty x^2 f_j dx - \int_0^\infty x^2 S_1 dx - \int_0^\infty x^2 S_2 dx - E_{eq}^2$$

$$= E_i^2 + V_i + E_j^2 + V_j - E_{eq}^2 - \int_0^\infty x^2 S_1 dx - \int_0^\infty x^2 S_2 dx$$

Consider the second last term in the above equation,

$$\int_0^\infty x^2 S_1 dx = \int_0^\infty x^2 \sum_{k=0}^{r_j-1} \frac{\lambda_i^{r_i}\,\lambda_j^k\, x^{r_i+k-1}}{(r_i-1)!\,k!} e^{-(\lambda_i+\lambda_j)x} dx$$

$$= \lambda_i^{r_i} \sum_{k=0}^{r_j-1} \frac{\lambda_j^k}{(r_i-1)!\,k!} \int_0^\infty x^{r_i+k+1} e^{-(\lambda_i+\lambda_j)x} dx$$

$$= \lambda_i^{r_i} \sum_{k=0}^{r_j-1} \frac{\lambda_j^k}{(r_i-1)!\,k!} \frac{(r_i+k+1)!}{(\lambda_i+\lambda_j)^{r_i+k+2}}$$

$$= \frac{\lambda_i^{r_i}}{(\lambda_i+\lambda_j)^{r_i+2}} \sum_{k=0}^{r_j-1} \left(\frac{\lambda_j}{\lambda_i+\lambda_j}\right)^k \frac{(r_i+k+1)!}{(r_i-1)!\,k!}$$

Similarly,

$$\int_0^\infty x^2 S_2 dx = \frac{\lambda_j^{r_j}}{(\lambda_i+\lambda_j)^{r_j+2}} \sum_{k=0}^{r_i-1} \left(\frac{\lambda_i}{\lambda_i+\lambda_j}\right)^k \frac{(r_j+k+1)!}{(r_j-1)!\,k!}$$

Therefore, the equivalent variance of the parallel combination is

$$V_{eq} = E_i^2 + E_j^2 + V_i + V_j - E_{eq}^2$$

$$- \frac{\lambda_i^{r_i}}{(\lambda_i+\lambda_j)^{r_i+2}} \sum_{k=0}^{r_j-1} \left(\frac{\lambda_j}{\lambda_i+\lambda_j}\right)^k \frac{(r_i+k+1)!}{(r_i-1)!\,k!}$$

$$- \frac{\lambda_j^{r_j}}{(\lambda_i+\lambda_j)^{r_j+2}} \sum_{k=0}^{r_i-1} \left(\frac{\lambda_i}{\lambda_i+\lambda_j}\right)^k \frac{(r_j+k+1)!}{(r_j-1)!\,k!}$$

References

[Cou77] P. J. Courtois. ***Decomposability: Queueing and Computer System Applications.*** Academic Press, 1977.

[DB78] Peter J. Denning and Jeffrey P. Buzen. "The Operational Analysis of Queueing Network Models". ***Computing Surveys,*** 10(3):225–261, September 1978.

[HT82] Philip Heidelberger and Kishor S. Trivedi. "Queueing Network Models for Parallel Processing with Asynchronous Tasks". ***IEEE Transactions on Computers,*** **C-31(11):1099-1109,** November 1982.

[HT83] Philip Heidelberger and Kishor S. Trivedi. "Analytic Queueing Models for Programs with Internal Concurrency". ***IEEE Transactions on Computers,*** C-32(1):73-82, January **1983.**

[Kle85] Wolfgang Kleinoder. "Evaluation of Task Structures for a Hierarchical Multiprocessor System". In D. Potier, editor, ***Modeling Techniques and Tools for Performance Analysis,*** pages 403-419, Elsevier Science Publishers B.V. (North Holland), 1985.

[LS79] Edward D. Lazowska and Kenneth C. Sevcik. "Exploiting Decomposability to Approximate Quantiles of Response Times in Queueing Networks". In D. Lanciaux, editor, ***Operating Systems: Theory and Practice,*** pages 149-166, North-Holland Publishing Company, **1979.**

[LZGS84] Edward D. Lazowska, John Zahorjan, G. Scott Graham, and Kenneth C. Sevcik. ***Quantitative System Performance.*** Prentice Hall, Englewood Cliffs, New Jersey, 1984.

[Moh84 Joseph Mohan. ***Performance*** *of* ***Parallel Programs: Model and Analyses.*** PhD thesis, Carnegie-Melon University, July **1984.**

[RL80] M. Reiser and S.S. Lavenberg. "Mean Value Analysis of Closed Multichain Queueing Networks". ***Journal*** *of* ***the ACM,*** 27(2):313–322, April 1980.

[Sch86] Herb Schwetman. "CSIM: A C-Based, Process-Oriented Simulation Language". In ***Proceedings for Winter Simulation Conference*** '86, Washington DC, December 1986.

[TB83] Alexander Thomasian and Paul Bay. "Queueing Network Models for Parallel Processing of Task Systems". In ***Proceedings*** *of* ***the 1983 International Conference on Parallel Processing,*** pages 421-428, IEEE, 1983.