A QMIPS Significant Example: Multiprocessor Performance with Task-Graph Models

Erol Gelenbe
Centre de Recherche de l'EHEI,
UFR de Mathématiques et d'Informatique,
Université de Paris V,
45 rue des Saints-Pères,
75006 Paris, France

Abstract

In this report, we address a *QMIPS Significant Example*. We consider a stream of jobs represented by finite task graphs. We show how the response time of such jobs, executing on multiprocessor systems with a fixed number of processors, can be estimated. We use an approximate modeling technique based on *The Precedence Wait Assumption* to compute the relevant performance measures.This report is based on joint work with Z. Liu [9].

1 Introduction

The multiprocessor system under consideration consists of a finite number of homogeneous processors which share a central memory. The workload model has the following salient features:

- Each *job* models a concurrent program; it consists of a finite set of *tasks* with (in general) different execution times.
- The concurrency and precedence relationship between tasks are represented by a directed acyclic graph with some arbitrary structure. Communication times between tasks are not represented in this significant example, though this aspect may be considered in future work.
- Jobs arrive to the system in an infinite stream. We assume that the stream of arrival instants is a Poisson process.

Because of the concurrency, the queueing network resulting from such a model does not - to the best of our knowledge - have a product-form or other simple solution.

An approximate solution method which models the system by a set of waiting queues for tasks whose predecessors have not yet terminated their execution, and a multiple-server queue representing the multiprocessor, is presented. Some simplifying assumptions are made to derive the estimates of the response time of jobs. The idea in the proposed solution method is to imbed the effect of the precedence constraints of the task graph, into additional fictitious queueing delays for the tasks. These delays depend on the "level" of the tasks in each graph, i.e. on the longest path of predecessors for each task. We call this simplifying assumption *The Precedence Wait Assumption.*

The accuracy of the approximation is evaluated through comparisons against simulation results. Some arguments are provided to indicate that these approximations are lower bounds to the expected response time of jobs.

We describe the problem in detail in the next section. In Section 3, an approximate analytical solution which covers the case where all tasks are statistically identical is described, and in Section 4 the approximate solution results are compared to simulation results. Section 5 provides an extension of the method to the case where the workload consists of concurrent programs having "different structures".

This report is based on joint work with Z. Liu [9].

1.1 The Significant Example's Assumptions

We assume that the multiprocessor system under consideration has M identical processors. A main memory is shared by all processors. It is assumed that this memory is of unlimited size so that it can store the address space of all programs which are present in the system.

The workload consists of a set of structurally identical concurrent programs, modelled as "jobs", consisting of a fixed number of interdependent tasks. For any job, the precedence constraints between tasks are represented by a fixed and identical acyclic precedence graph G in which each node represents a task. Each task has an execution time which is assumed to be an exponentially distributed random variable. Assume for the time being that any task (of identical or of different jobs) requires an exponentially distributed processing time with parameter μ. Processing times of different tasks are assumed to be independent. We consider a stream of jobs arriving to the system according to a Poisson process of rate λ.

Let N be the maximum number of predecessors that tasks of graph G may have: $(N \leq m-1)$, where m is the number of tasks in the job (or nodes in G).

Upon arrival, a job is immediately split into its constituent tasks so that tasks which have no predecessors are ready for immediate execution and enter directly into the processor queue waiting for service by one of the processors. The other tasks, which are not yet available for execution because of precedence constraints, have to wait in a buffer until their predecessors have been serviced. The multiprocessor is allowed to simultaneously serve tasks of different jobs on a FIFO basis. Thus a task which is released for execution, i.e. all of whose predecessors have terminated execution, will enter a FIFO execution queue.

2 Steps Towards an Exact Representation

An exact representation of the system could be the following. The state of the system is a collection of tasks which are either waiting in a buffer for their predecessor tasks to be terminated, or waiting in queue, or being processed by one of the processors. For a task to be waiting in queue or to be processed, all of its predecessor tasks must be processed. Assuming FIFO service at the processors, the state must also represent the order of arrival of tasks, which is determined by the order in which the programs they belong to have arrived.

Each task will be denoted by a pair of integers (u, v) where u is the program number and v is the task number within the program. The state of the system will be represented by

$$S \equiv (Q, W)$$

where Q and W are defined as follows:

$$Q \equiv \begin{cases} (t_1, t_2, \ldots, t_{|Q|}), & t_i \ (1 \leq i \leq |Q|) \text{ is a task, and we denote by } |Q| \\ & \text{the number of tasks running or ready to run,} \\ \emptyset, & \text{if there are no tasks in the system} \end{cases}$$

The tasks t_l in Q are given in FIFO order with t_1 being at the head of the queue. Therefore the first M of these tasks will actually be running.

$$W \equiv \begin{cases} \emptyset, & \text{if no tasks are waiting} \\ (w_1, w_2, \ldots, w_{|W|}), & \text{if there are } |W| \text{ tasks waiting,} \end{cases}$$

where each w_l is a pair

$$w_l \equiv \{T_l, A_l\}$$

in which T_l is a task and A_l is the set of tasks which are the immediate predecessors of T_l. In fact, A_l will be unnecessary if all the programs running in the system have exactly the same precedence graph; in that particular case the task T_l uniquely identifies the set A_l for all programs. However if programs can have different task graphs, the set A_l will have to be specified.

The analysis of this general state-space model seems beyond the possibilities offered by exact analytical modelling. In order to illustrate the complexity of the model consider the state transition associated with the arrival of a program, from $S = (Q, W)$ to $S' = (Q', W')$:

$$Q' \equiv (t_1, \ldots, t_{|Q|}, t'_{|Q|+1}, \ldots, t'_{|Q|+j}), \quad \text{if there are } j \text{ tasks without predecessors in the arriving program}$$

$$W' \equiv \begin{cases} (w_1, \ldots, w_{|W|}, w_{|W|+1}, \ldots, w_{|W|+k}), & \text{if there are } k \text{ tasks with predecessors in the arriving program} \\ \emptyset, & \text{if } |W| = 0 \text{ and } k = 0. \end{cases}$$

Obviously $w'_l = \{T'_l, A'_l\}$, so that the set of tasks of the arriving program is given by

$$\{T'_{|W|+1}, \ldots, T'_{|W|+k}, t'_{|Q|+1}, \ldots, t'_{|Q|+j}\}.$$

Furthermore the order in which the tasks t'_i which have no predecessors are placed in the processor queue Q' is of importance; we shall assume that they are placed in the order of increasing task number, so that the task with no predecessors and with the smallest task number is placed closest to the head of the queue. This will be the rule which will be used in general for placing tasks in the processor queue.

The representation of the other transactions related to task departures is even more complex. That is why in the rest of this chapter we shall deal with a simplified model in which tasks and their predecessors will not be identified individually, and certain simplifying assumptions will be made about the state transitions.

3 The Simplified Model: The Precedence Wait Assumption

Because an exact representation of system state, let alone an attempt at an analytical solution, appears so complex, we have developped an approximate solution technique. This technique is based on a simplification which we call the *Precedence Wait Assumption.* The idea is to introduce a fictitious waiting time for each task which depends on the longest path of its predecessors, as well as on the execution time of each task. This assumption is presented below:

- Job arrivals of rate λ are replaced by task arrivals of rate $m.\lambda$.
- Tasks having i ($i = 1, 2, \ldots, N$) predecessors enter a "fictitious" i-th buffer, to wait for their i predecessors' service completion.
- When the execution of a task at one of the $CPU's$ is completed, it leaves the system and may free some tasks waiting in some of the "fictitious buffers". The rate at which tasks are released from the buffers is described below.
- Once released by all its serviced predecessor tasks, a task leaves the buffer to join the FIFO CPU queue.

An arriving job remains active in the system until all of its tasks have completed execution.

In the simplified model the state of the system will be represented by the vector

$$\mathbf{k} = (k_0, k_1, \ldots, k_N)$$

where k_i is the number of tasks in the system which have i ($1 \leq i \leq N$) predecessors. Recall that N is the maximum number of predecessors which a task may have. k_0 denotes the number of tasks which have no unfinished predecessors at the instant considered. This state representation is obviously much simpler than the general one. However it does not capture in a precise manner the structure of the task graph. The transitions from some state $\mathbf{k}$ to a state $\mathbf{k}'$ are given as follows for the simplified model.

- The arrival of a program or job represented by a sequence of task arrivals:

  Let m_i be the number of tasks having i predecessors. Upon arrival of a task,

  $$\mathbf{k} = (k_0, \ldots, k_i + 1, \ldots, k_N), \qquad 0 \leq i \leq N$$

  with probability chosen to be $p_i \equiv m_i/m$. Notice that $\sum_{i=0}^{N} m_i = m$. The rate of transition from $\mathbf{k}$ to this particular $\mathbf{k}'$ is thus $\lambda m p_i$.

- The departure of a task from one of the wait buffers:

  We assume that transitions from state $\mathbf{k}$ to some state

  $$\mathbf{k}' = (k_0 + 1, \ldots, k_i - 1, \ldots, k_N), \qquad 1 \leq i \leq N$$

  where $k_i > 0$, occur with rate $\mu_i(k_i)$. These represent the departure rate of a task from the wait buffer into the processor queue. $\mu_i(k_i)$ will be determined as a function of model parameters as given below.

Figure 1: The model description

- The departure of a task from the processor queue:

  This will lead into the state

$$\mathbf{k}' = (k_0 - 1, k_1, \ldots, k_N), \qquad k_0 > 0$$

  with rate

$$\mu_0(k_0) = \begin{cases} k_0\mu, & \text{if } 1 \leq k_0 \leq M \\ M\mu, & \text{if } k_0 > M \end{cases} \tag{1}$$

  Thus we are assuming that all individual tasks have independent exponentially distributed execution times of average value $1/\mu$. We may take $\mu = 1$ without loss of generality.

No other state transitions, except for those given above, are allowed to occur. Fig. 1 illustrates the simplified model. In the sequel let $k = \sum_0^N k_i$ denote the total number of tasks in the system.

3.1 Derivation of the Job Response Time

The determination of the response time of a job described by a task graph G (i.e. the overall delay of graph G's tasks through the system), denoted by R_G, is of crucial importance in qualifying the performance of parallel processing. However hardly any theory exists to tackle such problems. The problem seems to be open in its general form.

In order to obtain a tractable solution to the model, we restrict our attention to the simplified model for which the following two major assumptions have been made in Section 2:

A1 Every buffer is replaced by a First-Come-First-Served queue. The service time of queue i ($1 \leq i \leq N$) is exponentially distributed with parameter μ_i which is given below in (2). When a task of queue i is released (or completes its 'service'), it joins queue 0 (the multiprocessor queue).

A2 The arrivals of tasks to the system constitute a Poisson process of parameter Λ, $\Lambda = m\lambda$. Let $T_1, T_2, \ldots, T_n, \ldots$ be the arriving tasks. We assume that tasks

$T_{im+1}, T_{im+2}, \ldots, T_{im+m}$ belong to the same precedence graph ($i = 1, 2, \ldots$). Upon arrival, a task enters queue j ($0 \leq j \leq N$) with probability $p_j = m_j/m$.

With assumptions **A1** and **A2**, we obtain a product form network of queues which provides the basis of our approach.

Before proceeding with the analysis, we introduce the following notation:

- $P(\mathbf{k})$: joint stationary queue length probability distribution,
- $P_i(k_i)$: stationary probability for queue i ($0 \leq i \leq N$) to have k_i ($k_i = 0, 1, 2, \ldots$) customers,
- $E(k_i)$: mean number of tasks in queue i ($0 \leq i \leq N$),
- $E(k)$: mean number of tasks in the system,
- W_i: mean response time of tasks in queue i ($0 \leq i \leq N$),
- W: mean response time of tasks in the system,
- Q: mean number of jobs in the system,
- R: mean response time of a job or program which is represented by the graph G,
- $\bar{w}_j$: expected waiting time of j tasks which arrive simultaneously to queue 0,
- q_i: 'service contribution' to queue i ($0 \leq i \leq N$) of a customer leaving queue 0.

Let V be the set of nodes of graph G, $Suc(v)$ the set of successors and $rank(v)$ the number of predecessors of node v. q_i is defined as the average number of successors having i predecessors:

$$q_i = \frac{1}{m} \left(\sum_{v \in V} \sum_{v' \in Suc(v)} \mathbf{1}(rank(v') = i) \right)$$

where $\mathbf{1}(x)$ is the characteristic function:

$$\mathbf{1}(x) = \begin{cases} 1, & \text{if } x \text{ is true} \\ 0, & \text{if } x \text{ is false} \end{cases}$$

We shall represent the effect of the departure of a task from the system on the set of waiting tasks, as the departure of (q_i/i) tasks on the average from the set of tasks in the ith waiting queue in order to join queue 0. This leads to the expression:

$$\mu_i = \gamma q_i / i \tag{2}$$

where γ is the computed effective departure rate from the multiprocessor service queue:

$$\gamma = \sum_{j=1}^{\infty} \mu_0(j).P_0(j) \tag{3}$$

Our approximate model will be an open product-form network; therefore we will use the following formula for the joint queue length probability distribution using the well known product form theorem [1]:

$$P(\mathbf{k}) = C \prod_{i=0}^{N} \prod_{n=1}^{k_i} \frac{\Lambda . e_i}{\mu_i(n)}$$

where C is a normalizing constant, μ_0 is given by (1) and the remaining parameters are the average number of visits to each queue:

$$e_0 = 1$$

$$e_i = p_i, \quad \text{if } 1 \leq i \leq N$$

and for $1 \leq i \leq N$:

$$\mu_i(n) = \mu_i, \quad n = 1, 2, \ldots$$

We may write $P(\mathbf{k})$ as follows:

$$P(\mathbf{k}) = \prod_{i=0}^{N} P_i(k_i)$$

where the

$$P_i(k_i) = C_i \prod_{n=1}^{k_i} \frac{\Lambda . e_i}{\mu_i}, \quad i = 0, \ldots, N$$

are the marginal probability distributions and the C_i are their normalizing constants:

$$C_i = \left(\sum_{k_i \geq 0} \prod_{n=1}^{k_i} \frac{\Lambda e_i}{\mu_i(n)} \right)^{-1}$$

These formulae lead to the following computational procedure which will be used to obtain the model's queue length probability distributions:

(1) $P_0(j)$ is obtained formally using

$$P_0(j) = C_0 \prod_{n=1}^{j} \frac{\Lambda}{\mu_0(n)}$$

(2) For each $i = 1, 2, \ldots, N$ we compute

$$\mu_i = \frac{q_i}{i} \sum_{j=1}^{\infty} \mu_0(j) . P_0(j)$$

(3) We can now calculate

$$P_i(k_i) = C_i \prod_{n=1}^{k_i} \frac{\Lambda p_i}{\mu_i}, \qquad i = 1, \ldots, N$$

using steps (1) and (2).

For $(i = 0, \ldots, N)$, the mean queue lengths are expressed as

$$E(k_i) = \sum_{n=1}^{\infty} n \ P_i(n)$$

Therefore the average mean number of tasks in the system $E(k)$ is given by

$$E(k) = \sum_{i=0}^{N} E(k_i)$$

By Little's well-known formula, we obtain the average response times of tasks in each queue

$$W_i = \frac{E(k_i)}{e_i \Lambda}$$

and the average total response time of a task

$$W = \frac{E(k)}{\Lambda}$$

The above formulae provide performance measures at the task level. In particular, W_i $(i = 1, \ldots, N)$ indicates the time that a task having i predecessors must expect to wait before becoming available for execution (i.e. joining queue 0).

In order to obtain R, the average response time of a program, we introduce the concept of *level* of a node, or task.

The *level* of a node v of a directed acyclic graph, denoted by $L(v)$, is defined as:

- $L(v) = 1$, if v has no predecessors,
- $L(v) = (\max_{p \in Pre(v)} L(p)) + 1$, where $Pre(v)$ denotes the set of immediate predecessors of node v.

The *level* of a task is considered to be the *level* of the corresponding node in the task graph. The *depth* of a graph, denoted by d, is given by

$$d = \max_{v \in V} L(v)$$

When a task completes its service at the multiprocessor queue (queue 0), on the average q_i tasks which belong to the same program leave queue i to join queue 0; these tasks are all of the immediate predecessors of the departing task. It is not unreasonable to suppose that tasks of a given program which have the same level in the task graph will be serviced at the multiprocessor queue roughly at the same time. Indeed, if all task execution times were identical it would follow that tasks of level i would only be processed after all tasks of level $(i-1)$ had already been served by the multiprocessor. This leads to the following additional simplifying assumption which we shall make.

A3 Tasks of the same level become simultaneously available for execution.

Let $t_1, t_2, \ldots, t_u$ be a group of tasks simultaneously arriving to the multiprocessor service queue. Without loss of generality, we assume that their service order is $t_1 \prec t_2 \prec \ldots \prec t_u$ where $a \prec b$ means that task b *cannot* execute before a. Let $w_k(n)$ denote the mean waiting time of task t_k $(1 \leq k \leq u)$ before service if there are n $(n = 0, 1, 2, \ldots)$ tasks in the queue at the instant of the arrival of the group. Thus we have the formula

$$w_k(n) = \begin{cases} 0, & \text{if } n \leq M - k \\ (n + k - M)\mu^{-1}/M, & \text{if } n \geq M - k + 1 \end{cases}$$

Therefore, for $j = 1, 2, \ldots,$ the expected waiting time of j concurrent tasks in the server queue can be expressed as

$$\bar{w}_j = \sum_{n=0}^{\infty} P_0(n)\ w_j(n)$$

or

$$\bar{w}_j = \sum_{n=\max(M-j+1,0)}^{\infty} P_0(n)\ (n + j - M)\ \mu^{-1}/M \tag{4}$$

Let $b(l)$ denote the number of tasks of level l $(1 \leq l \leq L_G)$. According to assumption **A3**, tasks are serviced level by level, and tasks of the same level arrive at the same time to the multiprocessor queue. Therefore the response time R of the parallel program may be expressed as the sum of the completion times of the tasks at each level:

$$R = \sum_{l=1}^{d} \left(\frac{1}{\mu} + \bar{w}_{b(l)} \right)$$

And finally, by Little's formula, the average number of jobs in the system Q, is expressed as

$$Q = \lambda\ R$$

4 Numerical Examples and Validation of the Precedence Wait Assumption

In this section we compare the predictions of the approximate solution technique with simulation results. The latter are based on a realistic simulator where parallel programs arrive to a multiprocessor queue with M exponential servers, each of service rate μ. The program arrival process is Poisson of rate λ and the parallel programs are described by a task graph. Simulation runs have been carried out so as to obtain the 95% confidence intervals in each case.

As a first example, consider the parallel program represented by Graph 3 shown in Figure 3, together with the other graphs used in the simulations. Its average response time as a function of system load $m\lambda/\mu M$ is plotted in Fig. 2 using the prediction of our mathematical model, and the results obtained from simulation experiments, for different values of the number of processors M.

Figure 2: Response times of Graph 3 under different system load conditions

Number of processors	System load	Approximate solution	Simulation estimate	Relative error %
1	0.720	32.57	46.84 $\pm$ 2.45	30.3
2	0.360	11.19	12.21 $\pm$ 0.22	8.3
3	0.485	8.71	9.27 $\pm$ 0.09	6.1
5	0.240	8.02	8.39 $\pm$ 0.06	4.4
10	0.072	8.00	8.30 $\pm$ 0.06	3.6

Table 1: Comparison of job response time for Graph 1 ($\lambda = 0.06,\ \mu = 1$)

Clearly the theoretical predictions are optimistic with respect to the more realistic simulation results. However, the discrepancy between the two does not appear to be excessive.

In Tables 1 to 5 we have presented similar results for five different program graphs (Graph 1 to Graph 5). The relative differences in average response time of the program between the mathematical model and the simulation can be less than 2% and as large as 30%. By and large, the difference appears to be less than 20%.

The fact that the approximation seems to underestimate the overall average response time can be explained by the three simplifying assumptions. Indeed:

- Bulk arrival queueing systems in general will behave worse than single arrival systems under the same load, with respect to the response time.
- Dependent services (in our model, services of queue 1 to N) generally imply a larger

Number of processors	System load	Approximate solution	Simulation estimate	Relative error %
1	0.960	184.00	241.30 ± 36.23	23.7
2	0.480	14.13	17.69 ± 0.44	20.2
3	0.320	10.85	12.48 ± 0.18	3.1
5	0.192	7.35	10.49 ± 0.07	30.0
10	0.096	7.00	10.21 ± 0.06	31.4

Table 2: Comparison of job response time for Graph 2 ($\lambda = 0.06$, $\mu = 1$)

Number of processors	System load	Approximate solution	Simulation estimate	Relative error %
1	0.960	256.00	260.20 ± 37.61	1.6
2	0.480	14.94	18.69 ± 0.44	20.1
3	0.320	10.92	13.78 ± 0.16	20.8
5	0.192	10.03	12.59 ± 0.08	20.3
10	0.096	10.00	12.49 ± 0.07	19.9

Table 3: Comparison of job response time for Graph 3 ($\lambda = 0.06$, $\mu = 1$)

Number of processors	System load	Approximate solution	Simulation estimate	Relative error %
1	0.900	78.00	134.30 ± 14.89	41.9
2	0.450	12.78	16.10 ± 0.45	20.6
3	0.300	9.79	10.10 ± 0.12	3.0
5	0.180	7.21	8.21 ± 0.06	12.2
10	0.009	7.00	7.89 ± 0.05	11.3

Table 4: Comparison of job response time for Graph 4 ($\lambda = 0.06$, $\mu = 1$)

Number of processors	System load	Approximate solution	Simulation estimate	Relative error %
1	0.960	208.00	296.10 ± 44.88	29.8
2	0.480	14.75	18.32 ± 0.52	19.5
3	0.320	11.06	11.79 ± 0.19	6.2
5	0.192	8.25	9.36 ± 0.06	11.9
10	0.096	8.00	8.98 ± 0.05	11.0

Table 5: Comparison of job response time for Graph 5 ($\lambda = 0.06$, $\mu = 1$)

Figure 3: Task Graphs 1 to 5 used in the simulation experiments

variance of service time, so that the waiting time is greater.

- Synchronization of tasks introduces additional delays over and above the task execution times.

These intuitive arguments indicate that our approximation method provides reasonably lower estimates to the expected response time of parallel programs.

5 Extension of the Method to the Analysis of Multiprocessor Systems with Different Types of Parallel Programs

The model we have discussed so far concerns systems in which programs arrive at random, and are composed of a set of tasks whose precedence graph is fixed and deterministic. Furthermore all programs have the same precedence graph, and all tasks have independent exponentially distributed service times with the same parameter μ.

In many practical cases of interest, different types of parallel programs will run on the same system. In this section we shall consider an extension of our model in order to handle this more general case.

We shall assume now that there are H different types of programs, each type h having a different precedence graph denoted G_h. However, all the tasks will be assumed to have the same exponential processing time distribution of parameter μ. Programs of type h arrive to the system according to a Poisson process of parameter λ_h, and the H different arrival processes are independent.

Programs of class h have m_h tasks which are subdivided into N_h ($N_h \leq m_h - 1$) groups according to their respective number of predecessors' tasks.

We now consider a stream of tasks $t_1, t_2, \ldots, t_n, \ldots$ arriving to the system, where the task t_n belongs to a program of class j with probability $m_j / \sum_1^H m_h$. The global arrival process is Poisson of parameter Λ,

$$\Lambda = \sum_{h=1}^{H} m_h . \lambda_h$$

Upon arrival, a task joins the queue i ($0 \leq i \leq N$) with probability p_i given by

$$p_i = \frac{\sum_{h=1}^{H} m_{hi}}{\sum_{h=1}^{H} m_h}$$

where m_{hi} is the number of tasks in G_h having i predecessors, and

$$\sum_{i=0}^{H} m_{hi} = m_h, \qquad h = 1, 2, \ldots, H$$

The *service contribution* of tasks of rank i for this model becomes

$$q_i = \frac{1}{\sum_{h=1}^{H} m_h \left(\sum_{h=1}^{H} \sum_{v \in G_h} \sum_{v' \in Suc_h(v)} \mathbf{1}(rank(v') = i) \right)}$$

where $Suc_h(v)$ denotes the set of successors of node v in the graph G_h.

We can now use again the previous results for computing the queue length probabilities, mean queue lengths, mean number of tasks in the system, and average delays. Furthermore, the mean number of tasks in the system which belong to jobs of class j $(j = 1, 2, \ldots, H)$ in queue i $(i = 1, 2, \ldots, N)$, denoted by $E(k_{ji})$, can be expressed as

$$E(k_{ji}) = \frac{m_{ji}\lambda_j}{\sum_{h=1}^{H} m_{hi}\lambda_h E(k_i)}$$

where $E(k_i)$ is obtained as in Section 3 with the new values of Λ and q_i. And the mean number of tasks which belong to jobs of class j $(j = 1, 2, \ldots, H)$ in the system, denoted by $E(k^j)$, can be expressed as

$$E(k^j) = \frac{m_j\lambda_j}{\sum_{h=1}^{H} m_h\lambda_h E(k)}$$

The average response time of a job of class h is also obtained using the approach developed in Section 3. With assumption **A3**, we can write

$$R_h = \sum_{l=1}^{L_k} \left(\frac{1}{\mu} + \bar{w}_{b_h}(l) \right)$$

where R_h denotes the expected response time of a program of class h, L_h denotes the level of graph G_h, $b_h(l)$ denotes the number of tasks in graph G_h at level l $(1 \leq l \leq L_{G_h})$, and $\bar{w}_j$ $(j = 1, 2, \ldots)$ is given by (4).

The average number of programs of class h in the system is expressed as:

$$Q_h = \lambda_h \ R_h$$

and the average number of programs in the system is

$$Q = \sum_{h=1}^{H} Q_h$$

We thus obtain a computationally tractable estimate of the average response time of concurrent programs of different types which run on the same multiprocessor system.

6 The Case of Systems with Processors of Different Types

Many multiprocessor systems contain processors of different types. Consider for instance the following two examples. Supercomputers will often include three types of processors: scalar arithmetic processors, vector or array (arithmetic) processors, and processors for simple instructions such as jumps, address decoding, etc. Other multiprocessor systems may include specialized processors which handle operating system calls, input-output processors, as well as processors which execute user program code. The CM-2 includes two types of processors: elementary logical processors and floating-point arithmetic processors.

We thus see that as a general rule multiprocessor systems will contain more than one type of processor. Programs will contain tasks which are directed to different types of processors, for instance floating point instructions and control instructions in a numerical analysis program, if we consider individual instructions to be tasks. Thus we shall consider a class of models with different types of tasks being executed on the corresponding types of processors.

In this section we shall address the analysis of this class of models with dependencies between tasks of the same program.

The general model described in Section 2 can also be applied to the case which is of interest to us here. However due to its intrinsic complexity we shall present a simplified model which yields very satisfactory approximate results.

Consider the model shown in Figure 4 . It represents a system having K types of processors; there are M_k processors of type k, $(1 \leq k \leq K)$. The arrival of a program to the system is represented by the arrival of a sequence of tasks. Tasks will be represented by customers moving through the queues and their execution times will be represented by service times at the servers. The state of the system will be represented by the vector

$$\mathbf{S} = (n_1, \ldots, n_K, b_1, \ldots, b_K)$$

where $n_1, \ldots, n_K$ are the numbers of tasks in the processor queues $(1, \ldots, K)$, respectively, and $(b_1, \ldots, b_K)$ are the numbers of tasks in the waiting queues $(1, \ldots, K)$, respectively.

The waiting queues contain those tasks, all of whose predecessors have not yet completed execution. A task from waiting queue i will request service from a processor of type i as soon as its predecessors (which may belong to any type) have completed execution. Processors serve the tasks in the processor queues in first-come-first-served order.

We shall assume that all processor service or execution times are independent and exponentially distributed, and that the waiting times in the waiting queue are also independent and exponentially distributed.

Furthermore, arrivals of tasks to the system will be represented by a Poisson process of parameter $\Lambda = \lambda m$, where λ is the rate of arrival of programs to the system (in programs per unit time) and m is the average number of tasks in a program.

We shall denote by m_k the average number of tasks of type k in a program, so that

$$m = \sum_{1}^{K} m_k$$

The tasks of a program may or may not have predecessors; thus there will be on the average m_{k_0} tasks of type k without predecessors and m_{k_1} on the average with predecessors, where $m_k = m_{k_0} + m_{k_1}$.

The state transitions of the model from some state S to some state S' are described as follows:

- Arrival of a task.

  The arrival of tasks to the system occur according to a Poisson process with rate $m\lambda$. Upon arrival, a task becomes a customer of processor queue k or of waiting queue k:

$$\mathbf{S'} = (n_1, \ldots, n_k + 1, \ldots, n_K, b_1, \ldots, b_K)$$

Figure 4: Simplified model description for the non-homogeneous processor case

or

$$\mathbf{S}' = (n_1, \ldots, n_K, b_1, \ldots, b_k + 1, \ldots, b_K)$$

with probability $p_{k_0} = m_{k_0}/m$, and $p_{k_1} = m_{k_1}/m$, respectively. The rates of transition from state $\mathbf{S}$ to these particular $\mathbf{S}'$s are thus $\lambda m p_{k_0}$, and $\lambda m p_{k_1}$, respectively.

- Departure of a task from one of the waiting queues.

  Departures from the waiting queues form independent Poisson processes. The state transition from $\mathbf{S}$ to some

  $$\mathbf{S}' = (n_1, \ldots, n_k + 1, \ldots, n_K, b_1, \ldots, b_k - 1, \ldots, b_K)$$

  for $b_k > 0$, occurs with rate $\lambda m p_{k_1}$.

- Departure of a task from a processor queue.

  This will lead into state

  $$\mathbf{S}' = (n_1, \ldots, n_k - 1, \ldots, n_K, b_1, \ldots, b_K)$$

  where $n_k > 0$. Such a state transition takes place with rate

  $$\mu_k(n_k) = \begin{cases} n_k \mu_k, & \text{if } 1 \leq n_k \leq M_k \\ M_k \mu_k, & \text{if } n_k > M_k \end{cases}$$

No other state transitions, except for those given above, can occur.

The total response time of a program will be derived from the total waiting and execution times of its tasks. These are composed of task precedence waiting times, task queueing times for executions, and task execution times. The precedence waiting times are due to the precedence constraints or the synchronization delays, and are the time needed for tasks to be enabled by their predecessors in the task graph. The queuing times result from waiting for processors.

The program response time will be computed in two steps. We first compute the task response times, and then compute program response time by taking into account the precedence waiting times using the task graph.

From the construction of the simplified model, every server queue forms an $M/M/m$ queueing system, with customer arrival rate $\lambda_k = p_k m\lambda$, where $p_k = m_k/m$, $(1 \le k \le K)$, and with service rate

$$\mu_k(j) = \begin{cases} j\mu_k, & \text{if } 1 \le j \le M_k \\ M_k\mu_k, & \text{if } j > M_k \end{cases}$$

The stationary probability distribution of the length of the kth processor queue, $(1 \le k \le K)$, denoted by $P_k(n_k)$, is

$$P_k(n_k) = C_k \prod_{j=1}^{n_k} \frac{\lambda_k}{\mu_k(j)}, \qquad n_k \ge 0 \tag{5}$$

where C_k is the normalizing constant:

$$C_k = \left(\sum_{n=0}^{\infty} \prod_{j=1}^{n} \frac{\lambda_k}{\mu_k(j)} \right)^{-1}$$

If there are j tasks (under execution or ready for execution) in the kth processor queue when a task arrives to it, its waiting time is the time which elapses until there are at most $M_k - 1$ tasks before it in queue.

Let $w_t(k,j)$ be the queueing time of task t in queue k if there are $(j \ge 0)$ tasks in the queue when it arrives, and let y_i^k $(i \ge 1)$ be the total time for i departures from queue k, given that all the M_k processors of queue k are busy (i.e., there are always tasks in the queue ready for execution when a processor terminates the execution of a task). Then

$$w_t(k,j) = \begin{cases} 0, & j \le M_k - 1 \\ y_j^k - M_k + 1, & j \ge M_k \end{cases}$$

If $X_k(x)$ is the probability distribution function of the time between successive departures from queue k given that all M_k processors of queue k are busy, and $Y_i^k(x)$ is the distribution function of y_i^k, then

$$X_k(x) = 1 - e^{-M_k\mu_k x}, \qquad x \ge 0,$$

and

$$Y_i^k = \mathop{\ast}_{n=0}^{i} X_k$$

where $\ast$ denotes convolution in the sense of the densities.

Let $T_k(x)$ $(1 \le k \le K)$ be the probability distribution function of the processing time of tasks on processors of type k, given by

$$T_k(x) = 1 - e^{-\mu_k x}, \quad x \ge 0,$$

and $R_k(x)$ be the probability distribution function of response time of the tasks run on processors of type k. The task response time distribution can be obtained from

$$R_k = \sum_{j=0}^{M_k-1} P_k(j).T_k + \sum_{j=M_k}^{\infty} P_k(j).(Y^k_{j-M_k+1} * T_k) \tag{6}$$

In practical terms, the above formula can be approximated by

$$R_k = \sum_{j=0}^{M_k-1} P_k(j).T_k + \sum_{j=M_k}^{Z} P_k(j).(Y^k_{j-M_k+1} * T_k) \tag{7}$$

where Z is chosen to be sufficiently large so that

$$\sum_{j=0}^{Z} P_k(j) \sim 1$$

Once the distribution functions $R_k(x)$ of the executions times of tasks is known using (7), we can proceed to compute the response time of a program as follows:

(i) We assign to each leaf (a node with no successors) of the task graph, a weight which is the function $R_k(x)$ if the leaf is a task of type k.

(ii) For every node i, all of whose successors already have an assigned weight, we assign a weight $W_i(x)$:

$$W_i(x) = R_k(x) \ * \prod_{j \in Suc(i)} W_j(x) \tag{8}$$

if node i is of Type k. In other words, we calculate the maximum of the weights of its set of successors $Suc(i)$, and then add to it its own execution time. Of course, this is done in terms of the distributions of the times.

This procedure will necessarily stop at the roots of the program graph, since it is acyclic.

The response time of the program will now have the following approximate distribution:

$$W(x) = \prod_{l \in \mathrm{Root}(G)} W_l(x) \tag{9}$$

where $\mathrm{Root}(G)$ is the set of roots of the program graph G. The expression (9) is based on the assumption that the paths starting from different roots are quasi-independent so that it is justified to calculate the maximum path length distribution as the product of the individual path length distributions.

This approach is illustrated in the following section.

Figure 5: Multiprocessor system example

6.1 An Example with a Workload Composed of Series-Parallel Task Graphs

In this section we shall use the method developed above for the evaluation of the response time in a multiple processor system in which the work-load is represented by a set of programs having the same series-parallel task graph structure.

The characteristics of the multiprocessor are as follows:

- It has a total of seven processors.
- They are subdivided into three types ($K = 3$).
- There are three processors of Type 1, two processors of Type 2, and two processors of Type 3 ($M_1 = 3,\ M_2 = 2,\ M_3 = 3$).

The system considered is shown on Fig. 5.

The programs arriving to the system have an identical series-parallel structure composed of ten tasks $T_1, \ldots, T_{10}$ as shown on Fig. 6, with the following characteristics:

- Tasks $T_1, T_3, T_5, T_9, T_{10}$ are of Type 1.
- Tasks T_2, T_4, T_8 are of Type 2.
- Tasks T_6, T_7 are of Type 3.
- Type 1 tasks have an average processing time of 1 $\mu_1 = 1$
- Type 2 tasks have an average processing time of 2 $\mu_2 = 0.5$
- Type 3 tasks' average processing time is 3 $\mu_3 = 0.33$.

Figure 6: The example series-parallel program structure with ten tasks

If the arrival rate of programs to the multiprocessor system is λ, then we will have

$$\lambda_1 = 5\lambda, \quad \lambda_2 = 3\lambda, \quad \lambda_3 = 2\lambda,$$

and we use (5) to obtain the stationary queue length distribution at each of the three processor queues.

The average response time is now calculated using the method of this section as a function of system load ρ defined as

$$\rho = \max_k \frac{\lambda m_k}{M_k \mu_k}$$

Notice that ρ describes the maximum load for any type of processor. It can be readily seen that

$$\frac{m_1}{M_1\mu_1} = 1.67, \qquad \frac{m_2}{M_2\mu_2} = 3, \qquad \frac{m_3}{M_3\mu_3} = 3$$

so that ρ will be determined by processors and tasks of Type 2 and 3. The results concerning the response times are shown on Figure 7 where the analytical approximation method is compared with simulation results; the latter are estimated with 95% confidence. The analytical approximation is seen to be highly accurate.

7 Bibliography

In this section we conduct a brief review of some literature. There has been much interest in the performance evaluation of multiprocessor systems with concurrent programs in recent years, and this bibliography is not meant to be exhaustive; the QMIPS Prposal's bibliography serves as a more thorough review of work up to 1992.

A concurrent program consists of a set of interdependent tasks with precedence constraints which are usually described by a task graph, also referred to as a precedence graph, whose nodes represent tasks and whose directed edges represent precedence relations between tasks. One of the most important performance measures considered is the overall response time of a concurrent program running on multiprocessor systems. When the duration of tasks is fixed and there are infinitely many processors, it is in principle easy to determine the completion time of a concurrent program by using PERT

Figure 7: Mean job response times under different system load conditions

diagrams. However computing the completion time from a PERT diagram can be very difficult if task execution times are expressed in probabilistic terms. Indeed, it can be shown that this computation can be NP-hard in the neumber of discretization steps used to numerically represent the task execution time distribution. If resources are limited, the probel becomes even harder. Thus when task durations are allowed to be random, the performance analysis of concurrent programs becomes even more difficult.

In order to make the analysis tractable, simple structures have been considered. Robinson [2], and Sahner and Trivedi [6] obtain the overall execution time by restricting the graph structure to the *series-parallel* case. Gelenbe et al. [3, 4] consider various random graph models and derive closed form expressions, and numerical fixed-point equations, for the maximum speed-up which can be expected assuming an unlimited number of processors.

Dodin [5] has studied general directed acyclic graph structures. He develops techniques based on removing edges or splitting nodes so that a directed acyclic graph can be transformed into a *series-parallel* graph. This leads to the derivation of lower bounds and upper bounds for overall execution times.

Studies along similar lines to those presented in this chapter, which are based on the use of queueing network models [1], can be found in [7, 8, 11] where further references are given. The presentation of this report is based on our previous work [9, 10].

[1] Baskett, F., Chandy, K.M., Muntz, R.R. and Palacios, F. G. 'Open, Closed and Mixed Networks of Queues with Different Classes of Customers',*J.ACM*, Vol. 22, pp. 248–260 (1975).

[2] Robinson, J. T. 'Some Analysis Techniques for Asynchronous Multiprocessor Algorithms', *IEEE Trans. on Software Engineering*, Vol.SE-5, No.1, pp. 24–31 (January 1979).

[3] Gelenbe, E., Montagne, E., Suros, R. and Woodside, C. M. 'A Performance Model of Block-Structured Parallel Programs', in *Parallel Algorithms and Architectures*,

M. Cosnard et al. (eds), North-Holland, Amsterdam, pp. 127–138 (1986).

[4] Gelenbe, E., Nelson, R., Philips, T. and Tantawi, A. 'The Asymptotic Processing Time for a Model of Parallel Computation', *Proceedings of National Computer Conference*, Las Vegas, USA (1986).

[5] Dodin, B. 'Bounding the Project Completion Time Distribution in PERT Networks', *Operations Research*, Vol.33, No.4, pp. 862–881 (July–August 1985).

[6] Sahner, R. A. and Trivedi, K. S. 'Performance and Reliability Analysis Using Directed Acyclic Graphs', *IEEE Trans. on Software Engineering*, Vol.SE-13, No.10, pp. 1105–1114 (October 1987).

[7] Towsley, D., Rommel, C. G. and Stankovic, J. A. 'The Performance of Processor Sharing for Scheduling Fork-Join Jobs on Multiprocessors', in *High Performance Computer Systems*, E. Gelenbe (ed.), North-Holland, pp. 145–156 (1988).

[8] Nelson, R. and Tantawi, A. 'Approximating Task Response Times in Fork-Join Queues', in *High Performance Computer Systems*, E. Gelenbe (ed.), North-Holland, pp. 157–167 (1988).

[9] Gelenbe, E. and Liu, Z. 'Performance Analysis Approximations for Parallel Processing in Multiprocessor Systems', North-Holland, *Proceedings of the IFIP Working Conference on Parallel Processing*, Pisa, Italy (April 1988).

[10] Gelenbe, E. 'Multiprocessor Performance', John Wiley and Sons, Ltd., New York and Chichester, 1989.

[11] Chu, W.W. and Leung, K.K. 'Module Replication and Assignment for Real-Time Distributed Processing Systems', *Proceedings of the IEEE*, Vol. 75, No. 5 (May 1987).