# RESPONSE TIME IN PARALLEL PROCESSING SYSTEMS WITH CERTAIN SYNCHRONIZATION CONSTRAINTS

Abdelfettah Belghith

November 1986
CSD-860033

UCLA-CSD-860033
November 1986

# RESPONSE TIME IN PARALLEL PROCESSING SYSTEMS WITH CERTAIN SYNCHRONIZATION CONSTRAINTS

by
**Abdelfettah Belghith**
**Leonard Kleinrock**

This research was sponsored by the Defense Advanced Research Projects Agency, Department of Defense.

**Computer Science Department**
**School of Engineering and Applied Science**
**University of California**
**Los Angeles**

## ACKNOWLEDGEMENTS

The support by the Advanced Research Projects Agency of the Department of Defense (Contract number MDA-903-77-C-0272) for this work is greatly appreciated, and we owe a debt of gratitude to the many employees of that contract. We would especially like to thank Lily Chien and Jodi Feiner for their administrative assistance.

Table of Contents

## ABSTRACT OF THE REPORT

Response Time in Parallel Processing
Systems with Certain Synchronization Constraints

by

Abdelfettah Belghith
Leonard Kleinrock
Computer Science Department
School of Engineering and Applied Science
University of California, Los Angeles

We view a multiprocessor system as a set of P cooperating processors, and a computer job as a set of tasks partially ordered by some precedence relationships. For a finite number of processors, we assume that the total system capacity is shared among the jobs proportionate to the number of their ready tasks. This is non-egalitarian Processor Sharing, as compared with the usual egalitarian Processor Sharing technique.

Significant reductions in the expected job sojourn time can be realized by executing a job on a multiprocessor system. This effect is known as the speedup factor, which typically increases with the number of processors used. Along with an increase in the speedup factor, comes a decrease in the efficiency of the processors. While a large speedup factor may appear as a delight to the users, the efficiency of the processors is also extremely important. The inherent, internal parallelism within jobs, namely, a job may need and consequently may hold more than one processor at a time, makes the exact analysis of parallel processing systems so problematic. We formulate an accurate and yet simple parametric approximation of the expected sojourn time, and then investigate the tradeoff between speedup and processor efficiency.

List of Figures

page

# RESPONSE TIME IN PARALLEL PROCESSING SYSTEMS WITH CERTAIN SYNCHRONIZATION CONSTRAINTS

In [Belg85], we studied the number of occupied processors in multiprocessor systems. In particular, we proved that the expected number of busy processors is a function only of the job average arrival rate, the task average processing requirements, and the average number of tasks per job. Although this expected number of busy processors in a multiprocessing system provides some insights as to how much resources can be utilized simultaneously, it does not accurately ascertain the level of parallelism achieved by executing the jobs on a multiprocessor system. This is the aim of the present Report.

First, we introduce and define a new scheduling policy (i.e., a service discipline) based on a non-egalitarian sharing of the processors capacity among the jobs present in the system. Using this scheduling policy, we convert the process graph describing the jobs into an execution graph which identifies the execution stages assumed by any job throughout its life in the system. In Section 2, we consider an infinite number of processors, and we formulate an exact expression, a tight upper bound, and a tight lower bound for the job average response time. In Section 3, we consider the uniprocessor case. We first prove that our scheduling policy forms a complete family of scheduling strategies, in the sense that any response time requirement that can be satisfied at all, can be accomplished by a strategy from the family. Then, we prove a conservation law that puts a linear equality constraint on the set of expected system times of the job execution stages. An accurate and yet very simple approximation for the job average response time is then formulated. In Section 4, we study the job average response time through a multiprocessing system in which all execution stages have the same concurrency degree. This will enable us to formulate a parametric approximation of the job average response time in a multiprocessing system. Simulations are used to validate and prove the excellent accuracy of such an approximation. We conclude the Report by studying the achievable parallelism and the efficiency per processor, and by identifying the optimal operating points at which one should operate the multiprocessor system.

## 1 Model Description

We assume that a job may be modeled as a set of N partially ordered tasks, and is represented by a given directed acyclic graph called hereafter a *Process Graph*, the same for all jobs. The Nodes in the process graph represent the tasks, and the edges represent the precedence relationships among the tasks. The processing time of a task is assumed throughout this chapter to be an exponentially distributed random variable. Different tasks in the process graph have independent and perhaps different mean processing requirements. Tasks in the process graph are identified using alphabetical labels. The task identity set, denoted hereafter by $\Omega$, is the set containing the identities of the N tasks. Let $\tilde{X}_A$, $A \in \Omega$, denote the random variable representing the processing requirement of task A with mean $\frac{1}{\mu_A}$ and a probability density function $b_X(x)=\mu_A e^{-\mu_A x}$ for $x \geq 0$. Jobs arrive to the multiprocessing system according to a Poisson process with an aggregate rate $\lambda$. The job process graph is assumed throughout the chapter, unless stated otherwise, to possess only one starting task and only one terminating task. This property of the process graph is required in the design and analysis of the approximations of the job average response time.

Throughout the Report, we restrict the notion of a scheduling strategy. We consider only strategies which satisfy the following two conditions:

1. They do not explicitly rely on any information about the remaining processing time of any job in the system, and

2. they do not allow processors to be idle when there are jobs waiting to be processed (i.e., work-conserving strategies).

The scheduling strategy (i.e., the service discipline) to be adopted in the case of a finite number of processors will be defined as stated below. For the uniprocessor case however, we shall also consider preemptive and nonpreemptive priority scheduling strategies. In such a case, we consider that the tasks in the process graph are assigned arbitrary but prescribed priority levels. For the infinite number of processors case however, the scheduling strategy vanishes and plays no role.

Let $\tilde{n}$ denote the random variable representing the total number of *ready* (i.e., ready-for-service) tasks from all the jobs present in the system in the steady state. The *Discriminatory Processors Sharing Discipline* for our multiprocessing system is defined as follows:

1. If the total number of ready tasks, $\tilde{n}$, in the system is less than or equal to the number of processors P, then each ready task is allocated one processor; that is each ready task is processed at a rate of 1 second per second.

2. If the total number of ready tasks, $\tilde{n}$, in the system is greater than or equal to the number of processors P, then each ready task is served at a rate of $\frac{P}{\tilde{n}}$ seconds per second. The ready tasks equally share the P processors.

Although at any time the ready tasks share the capacity of the processors in equal proportions, the above defined scheduling discipline divides the total processors capacity in unequal fractions among the jobs present in the system. This is due to the fact that jobs, at any given time, may participate with different numbers of ready tasks. The jobs that possess the largest number of ready tasks will then receive the most preferential treatment at the expense of the others (i.e., at the expense of the jobs having lower numbers of ready tasks). We shall elaborate on this in the sequel, but first let us introduce the notion of an Execution Graph.

An Execution Graph for a given process graph is an acyclic directed graph with nodes representing the state of execution of a job (i.e., the identity of the ready tasks of the job), and edges representing the precedence relationships among the nodes. A node in the execution graph is hereafter called a *stage* of the execution graph. Assuming either an infinite number of processors, or a Processor Sharing service discipline among all the ready tasks, it is not difficult to see that a process graph can always be converted to an execution graph. Consider the process graph given in Figure 1, and having 7 tasks identified by the set $\Omega = \left\{A,B,C,D,E,F,G\right\}$ with A being the starting task and G being the terminating task. Upon the arrival of a job described by such a process graph, its starting task A immediately acquires the ready-for-service status, and thus the execution stage of the job at its arrival instant comprises only the task A. At the completion time of task A, the job forks into two new tasks; namely task B and task C, which immediately assume the ready-for-service status, and consequently the job execution stage at such an instant comprises both tasks B and C. At this point, both tasks B and C are executed at the same rate. If task B finishes first then tasks D and E assume the ready-for-service status, and the new execution stage at the completion time of task B comprises the three ready tasks; namely C,D and E. Otherwise, if task C finishes first then task F would acquire the ready-for-service status, and consequently at the completion time of task C, the job execution stage comprises the ready tasks B and F. Proceeding in this way, the process graph given by Figure 1 results in the execution graph depicted in Figure 2, where the stages are represented by circles and are numbered from 1 to 16. Letters inside the circles denote the identities of the ready tasks comprised in such stages.

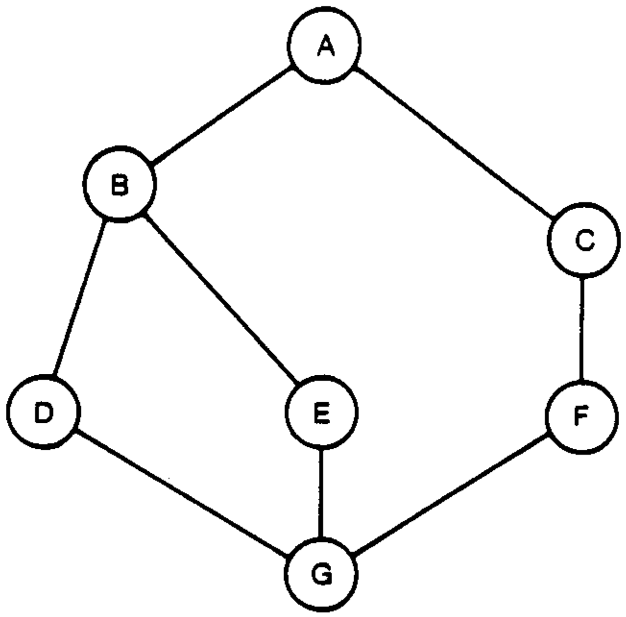


Figure 1: Process Graph with Identity Set $\Omega = \left\{A, B, C, D, E, F, G\right\}$

Generally, a stage in the execution graph represents a specific set of tasks in the job process graph that may be executed in parallel. Formally, let L denote the total number of stages in the execution graph, $\alpha(i)$, i=1,...,L identify the set of ready tasks that are executed concurrently when the job is at execution stage i, and f(i), i=1,...,L be the number of tasks in stage i (i.e., f(i) denotes the cardinality of the set $\alpha(i)$, i=1,...,L) also called hereunder the concurrency of stage i. The task identity set $\Omega$ is then defined as a function of the $\alpha(i)$, i=1,...,L by:

$$\Omega = \bigcup_{i=1}^{L} \alpha(i)$$

From each stage, say stage i, in the execution graph, there are f(i) outgoing edges, each corresponding to the termination of one of the ready tasks being executed in the set $\alpha(i)$. The stages at the end of these edges comprise the set of tasks in $\alpha(i)$ minus the just completed task, plus the new ready tasks, if any, that are activated by the completed task (those which acquire the ready-for-service status upon the completion of the completed task).

To fully describe the job execution graph, we must determine the transition probabilities between the execution stages. Note that the time spent by a job in any given stage is the time needed to finish one of its ready tasks comprised in such a stage, and that upon the completion of the execution of the terminating stage, namely stage L, the job departs the system at

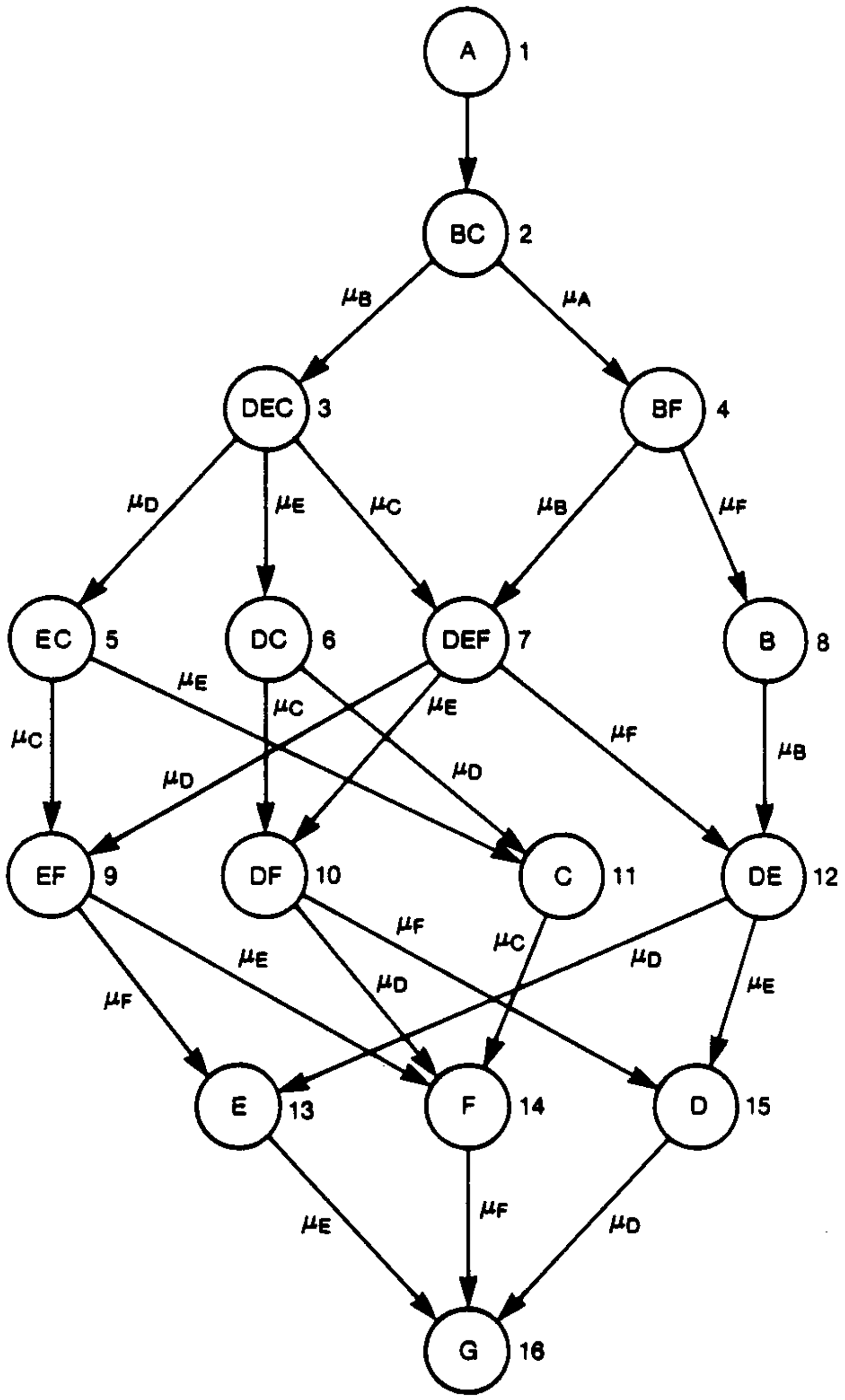


Figure 2: The Execution Graph of the Process Graph of Figure 1

once. An execution stage, other than stage L, comprising one ready task, has only one successor stage and consequently the transition probability is one. For execution stages with more than one ready task, the situation is a bit more complicated. Consider execution stage number 3 in the execution graph of Figure 2. This stage has 3 ready tasks (namely $\alpha(3) = \{C, D, E\}$ and must then have three successor stages, which in fact are stage 5, stage 6, and stage 7 as depicted in Figure 2. The processing times of these tasks are respectively $\tilde{X}_C$, $\tilde{X}_D$, and $\tilde{X}_E$ which are exponentially distributed random variables with respective averages $\frac{1}{\mu_C}$, $\frac{1}{\mu_D}$, and $\frac{1}{\mu_E}$. Let $P_{ij}$, i=1,...,L, j=1,...,L be the transition probabilities between stage i and stage j. Due to the memoryless property (i.e., the Markovian property) of the exponential service time distribution, a task in any stage, say stage number i, has the same mean service time regardless of whether it had being processed earlier in another preceding stage. Moreover, if $\tilde{X}$ and $\tilde{Y}$ are independent and exponentially distributed random variables with respective means $\frac{1}{\mu_X}$ and $\frac{1}{\mu_Y}$, then [*] $P[\tilde{X} \leq \tilde{Y}] = \frac{\mu_X}{\mu_X + \mu_Y}$. Hence:

$$P[\text{ task Z completes first } \mid \text{Z in } \alpha(i)] = \frac{\mu_Z}{\sum\limits_{\text{s.t. task A is in} \alpha(i)} \mu_A} \qquad \text{for all } \ i=1,...,L \tag{1}$$

Using the above equation, and for our example we obtain:

$$P_{35} = \frac{\mu_D}{\mu_C + \mu_D + \mu_E} \qquad P_{36} = \frac{\mu_E}{\mu_C + \mu_D + \mu_E} \qquad P_{37} = \frac{\mu_C}{\mu_C + \mu_D + \mu_E}$$

In the case of the same exponential service time distribution for all the N tasks, equation (1) becomes:

[*] In fact:

$$P[\tilde{X} \leq \tilde{Y}] = \int_{y=0}^{\infty} P[\tilde{X} \leq y \mid y \leq \tilde{Y} < y+dy] dP[\tilde{Y} \leq y]$$

$$= \int_{y=0}^{\infty} \left[1 - e^{-\mu_X y}\right] \mu_Y e^{-\mu_Y y} dy$$

$$= \int_{y=0}^{\infty} \mu_Y e^{-\mu_Y y} dy \;-\; \frac{\mu_Y}{\mu_X + \mu_Y} \int_{y=0}^{\infty} (\mu_X + \mu_Y) e^{-(\mu_X + \mu_Y) y} dy$$

$$= 1 - \frac{\mu_Y}{\mu_X + \mu_Y} = \frac{\mu_X}{\mu_X = \mu_Y}$$

$$P\,[\text{tasks Z completes first} \mid \text{Z in } \alpha(i)] = \frac{1}{f(i)} \qquad \text{for all } i=1,\ldots,L$$

At any time during its sojourn time in the system, a job is fully described by its current execution stage. The global state of the multiprocessing system is thus fully described by the total number of jobs in each stage of the execution graph. It is not difficult to see that the execution graph is a Markovian state transition diagram. Indeed, in Figure 2, we indicated, on each directed edge, the instantaneous average rate of exit from the execution stage along that edge for the case of an infinite number of processors. For stage number 3 for example, the rate of exit to stage number 5 is $\mu_D$, the rate of exit to stage number 6 is $\mu_E$, and the rate of exit to stage number 7 is $\mu_C$.

Starting from the initial stage in the execution graph, there are many paths a job can traverse before reaching the terminating stage. Since we know the transition probabilities between the stages (i.e., the probability of traversing each edge in the execution graph), the probability that a specific path is to be taking can be calculated. As an example, take the path (1,2,3,5,9,13,16) in the execution graph depicted in Figure 2. The probability of taking such a path is:

$$1.\frac{\mu_B}{\mu_B+\mu_C}.\frac{\mu_D}{\mu_C+\mu_D+\mu_E}.\frac{\mu_C}{\mu_C+\mu_E}.\frac{\mu_F}{\mu_E+\mu_F}.1.1$$

and in the case where all tasks have the same mean service time, the probability of taking such a path becomes $\frac{1}{24}$.

Suppose there are M paths from the initial stage to the terminating stage in the execution graph, and let $P(m)$, $m=1,\ldots,M$ represent the probability that a newly arriving job takes path m in the execution graph. Therefore, we may think of our job arrival process as composed of M Poissonian arrival processes, the mth of which has an average rate $\lambda(m) = \lambda P(m)$, m=1,...,M. Moreover, the number of stages in any given path is equal to the number of tasks, N, in the process graph, and consequently a job is a chain of N specific execution stages and is hereafter regarded as requiring service N times. Upon arrival to the multiprocessing system, a job is at its first execution stage. Upon the completion of this first stage, we may consider that the job immediately and instantaneously feeds back its second execution stage. At the completion of its Nth stage, the job departs the system at once. The number of ready tasks a job has at any given time is equal to the concurrency (the number of ready tasks) of the stage the job is in at such a time. During its sojourn time in the system, a job participates with different concurrency levels and hence receives different grades of service. It is for this very reason that our multiprocessing system is hereafter called a *P-dimensional Discriminatory processor sharing With job Feedbacks* and denoted using the P-DPS-WF acronym.

The global state of our P-DPS-WF multiprocessing system is fully described by the vector $S = (n_1, \ldots, n_i, \ldots, n_L)$ where $n_i$, i=1,...,L represents the number of jobs in the system which are in stage i of their execution. We can think of our multiprocessing system as a single node queueing network with L classes [*]. For a finite number of processors, the total capacity of the system is allocated to the different classes according to the discriminatory processor sharing discipline defined earlier. If at any given time, the state of the system is $(n_1, \ldots, n_i, \ldots, n_L)$, then the capacity proportion allocated to class i is:

$$\frac{n_i f(i) P}{\max\left\{P, \sum_{j=1}^{L} n_j f(j)\right\}} \qquad i=1,\ldots,L$$

since a class i job possesses f(i) ready tasks, and $\sum_{j=1}^{L} n_j f(j)$ is the total number of ready tasks in the system. The total number of jobs in the system, on the other hand, is readily given by $n = \sum_{j=1}^{L} n_j$.

Let $P[n_1, \ldots, n_i, \ldots, n_L]$ be the steady state probability density function that the system is in state $(n_1, \ldots, n_i, \ldots, n_L)$. To obtain these probabilities for all the feasible states, one must find a solution to the global balance equations of the system. From the theory of queueing networks, we know that $P[n_1, \ldots, n_i, \ldots, n_L]$ has the *Product Form* and is efficiently computable under the following set of assumptions provided by Baskett, Chandy, Muntz and Palacios [Bask75], and subsequently by Chandy, Howard and Towsley [Chan77] :

1. *Allowable Scheduling Disciplines* : the disciplines allowed are: First Come First Served (FCFS), Processor-Sharing (PS), Last Come First Served Preemptive-Resume (LCFS-PR), and Infinite Servers (IS).

2. *Service Time Distribution* : the service times at an FCFS server must be exponentially distributed with the same mean for all classes. The service times at PS, LCFS-PR, and IS can have a general distribution, perhaps with different mean service times for different classes.

3. *State Dependent Service Rates* : the service rate at an FCFS server can depend only on the total queue length of said server. The service rate for a class at a PS, LCFS-PR, and IS servers may also depend on the queue length for that class, but not on the queue length of other classes. Moreover, the overall service rate of a subnetwork may depend on the total number of customers in the subnetwork.

[*] A job is of class i, i=1,...,L if it is in its ith execution stage; $n_i$, i=1,...,L is also the number of jobs of class i present in the system in steady state.

4. *Interarrival time Distribution*: exogenous arrivals for a given class must be Poisson. In particular no bulk arrivals are permitted

From this set of assumptions, we can see that the third one (and perhaps the fourth one too) cannot be satisfied for our multiprocessing system. The third and fourth assumptions are usually referred to as the *Homogeneity Assumption*, which states that the service rate at each server for a particular class does not depend on the state of the system in any way except for the total queue length and the designated class's queue length at that server. This assumption essentially implies the following:

a. *Single Resource Possession* : a customer may not be present (waiting for service or being served) at more than one server.

b. *No Blocking* : the server's ability to render service is not controlled by any other servers.

c. *Independent Customer Behavior* : there should not be any synchronization requirements. Interaction among customers is limited to queueing effects.

d. *Local Information* : the service rate of any server depends solely on local queue length and not on the state of the rest of the system.

e. *Fair Service* : if service rates differ by class, the service rate for a class depends only on the queue length of that class at that server, and not on the queue lengths of other classes. The servers may not discriminate against customers in a class depending on queue lengths in other classes.

Nevertheless, there are two cases we can identify where the $P[n_1, \ldots, n_i, \ldots, n_L]$ has a *Product Form* solution as given by the following Proposition.

**Proposition 1**

If $P[n_1, \ldots, n_i, \ldots, n_L]$ is the steady state probability density function that the P-DPS-WF multiprocessing system is at state $(n_1, \ldots, n_i, \ldots, n_L)$, then for the two following cases $P[n_1, \ldots, n_i, \ldots, n_L]$ has the *Product Form* solution:

1. Infinite number of processors, and

2. finite number of processors with f(i)=F, i=1,...,L where F is any positive real constant.

**Proof**

We shall prove that for these two cases, the *homogeneity assumption* is satisfied. The proportion of the processors capacity, denoted in this proof by $C_i$, received by class i stages for i=1,...,L when the state of the system is $(n_1, \dots, n_i, \dots, n_L)$, is $C_i = \frac{n_i f(i) P}{\max\left\{P, \sum_{j=1}^{L} n_j f(j)\right\}}$. For the case of an infinite number of processors, the proportion $C_i$ becomes $C_i = n_i f(i)$, i=1,...,L which depends only on the number of stages of class i, and consequently assures the *homogeneity assumption* for *product form* solutions. For the case of constant concurrency degree, the same for all the stages, the proportion of the processors capacity, $C_i$, i=1,...,L, received by class i stages for i=1,...,L when the state of the system is $(n_1, \dots, n_i, \dots, n_L)$, is:

$$C_i = \frac{n_i F P}{\max\left\{P, \sum_{j=1}^{L} n_j F\right\}} = \frac{n_i P}{\max\left\{\frac{P}{F}, \sum_{j=1}^{L} n_j\right\}}$$

the proportion $C_i$, i=1,...,L depends then only on the number of class i stages and the total number of stages in the system; and hence satisfies the *homogeneity assumption* for the system to possess a *product form* solution.

∎

The notion of an execution graph can be extended in a natural way, so that an execution stage i, i=1,...,L may have any arbitrary concurrency degree f(i). Two distinguished case are identified in the following definitions.

**Definition 1**

An Abstracted Execution Chain (AEC) is a chain of N execution stages, each of which may have any arbitrary positive real concurrency degree.

**Definition 2**

A Restricted Abstracted Execution Chain (RAEC) is a chain of N execution stages, each of which may have any arbitrary positive and integer concurrency degree. Moreover, the concurrency degree f(i) of stage i, i=1,...,N is less than or equal to N, and such that $\sum_{i=1}^{N} f(i) \leq \frac{N(N+1)}{2}$. An RAEC is said to be feasible if it actually corresponds to a given process graph with one starting task and oneterminating task.

In the sequel, we shall use abstracted execution chains to show that the discriminatory processor sharing discipline with feedback forms a complete parameterized family of scheduling strategies. Restricted abstracted execution chains, on the other hand, shall be used to formulate upper and lower bounds on the job expected response time , and ascertain the process graphs that provide such upper and lower bounds.

Note that while an execution graph always corresponds to a given process graph, abstracted execution chains and restricted abstracted execution chains may not necessarily correspond to any real process graph description. Let the stages in the AEC and the RAEC be numbered from 1 to N. Three limiting cases of RAECs may be distinguished from definition (2); these are:

**Definition 3**

A Breadth First Execution Chain (BFEC) is an RAEC with f(i)=N-i+1 for i=1,...,N. A Depth First Execution Chain (DFEC) }is an RAEC with f(i)=i for i=1,...,N. An Egalitarian Execution Chain (EEC) is an RAEC with f(i)=1 for all i=1,...,N.

Let ρ denote the total utilization factor of our P-DPS-WF multiprocessing system. We can either express ρ using the process graph description, or its corresponding execution graph description. Using the process graph description, the utilization factor ρ may be expressed as follows:

$$\rho = \frac{\lambda}{P} \sum_{A \in \Omega} \frac{1}{\mu_A} \qquad (2)$$

Using the execution graph description, we also obtain the above expression of the system utilization factor, for a job takes a given execution path in the execution graph, and in such a path every task appears exactly once. From [Klei75] we know that the stability of the system is maintained as long as ρ is less than unity.

## 2 The Infinite Number of Processors Case

We consider an infinite number of processors. We shall first develop an expression for the job average response time, where jobs may be represented by any given arbitrary process graph comprising N tasks. Then, and by the use of restricted abstracted execution chains, we formulate a tight upper bound and a tight lower bound on the average response time and provide the process graphs, among all possible process graphs comprising N tasks, which achieve these upper and lower bounds.

We now proceed to determine the average input (i.e., arrival) rate to each stage in the job execution graph. Recall that the average arrival rate of jobs to the system is $\lambda$, that the number of levels in the execution graph is equal to the number N of tasks in the process graph, and that the number of stages in the execution graph is L, with $L \geq N$. Let $\lambda_i$, i=1,...,L represent the average arrival rate of class i jobs (i.e., the average input rate to stage i). We readily have $\lambda = \lambda_1 = \lambda_L$, and using the transition probabilities between the stages yields:

$$\lambda_i = \sum_{j=1}^{i-1} P_{ij} \lambda_j \quad i=2,...,L \tag{3}$$

The sum of the average input rate to all execution stages in any given level of the execution graph is $\lambda$. Since there are exactly N levels in the execution graph, we must have:

$$\sum_{i=1}^{L} \lambda_i = N\lambda$$

An execution stage may belong to several paths in the execution graph. The probability that stage i is visited during a job execution is then given by $\frac{\lambda_i}{\lambda}$. On the other hand, the expected time a job spends in stage i is given by $\frac{1}{\sum_{k \in \alpha(i)} \mu_k}$. Consequently, the average response time, denoted by $T_\infty$, of a job with an arbitrary process graph comprising N tasks is given by:

$$T_\infty = \sum_{i=1}^{L} \frac{\lambda_i}{\lambda \sum_{k \in \alpha(i)} \mu_k} \tag{4}$$

For the case of the same average service time, say $\frac{1}{\mu}$, for all the N tasks, equation (4) reduces to:

$$T_\infty = \frac{1}{\lambda\mu} \sum_{i=1}^{L} \frac{\lambda_i}{f(i)} \tag{5}$$

Let $T_{\infty,UB}$ and $T_{\infty,LB}$ represent, respectively, the upper bound and the lower bound on the expected response time of jobs with an arbitrary process graph comprising N tasks, and in a system with an infinite number of processors. The following two Theorems provide the exact values of $T_{\infty,UB}$ and $T_{\infty,LB}$.

**Theorem 1**

The upper bound $T_{\infty,UB}$ on the expected response time of jobs having any given arbitrary process graph comprising N tasks is given by:

$$T_{\infty,UB} = \sum_{i=1}^{N} \frac{1}{\mu_i}$$

**Proof**

From equation (4), we obtain the following nonlinear program to solve:

$$\text{Maximize} \quad A = \sum_{i=1}^{L} \frac{\lambda_i}{\sum_{k \in \alpha(i)} \mu_k} \qquad \text{Subject to:} \quad \begin{cases} 1. \ \sum_{i=1}^{L} \lambda_i = \lambda N \\ 2. \ f(i) \geq 1 \,, \forall i = 1, \dots, L \\ 3. \ \lambda_i \leq \lambda \\ 4. \ L \geq N \end{cases}$$

Consider the ith term in the expression of A. Maximizing $\frac{\lambda_i}{\sum_{k \in \alpha(i)} \mu_k}$ is the same as maximizing $\lambda_i$ and minimizing the sum $\sum_{k \in \alpha(i)} \mu_k$, for all i=1,...,L. The maximum value of $\lambda_i$, i=1,...,L is $\lambda$. The minimum of the sum $\sum_{k \in \alpha(i)} \mu_k$, i=1,...,L is obtained when f(i)=1. On the other hand, if f(i)=1, i=1,...,L then L=N. All the four constraints are also satisfied, and A becomes $A = \lambda \sum_{i=1}^{N} \frac{1}{\mu_i}$.

IIII

It is rather interesting to note that among all possible process graphs comprising N tasks, the process graph PG(N,N) is the one that maximizes the expected job response time; such an average response time is exactly $T_{\infty,UB}$. Also in the special case of the same average service time, say $\frac{1}{\mu}$, for all the N tasks, we have $T_{\infty,UB} = \frac{N}{\mu}$. Although in general it is hard to infer the total number of stages in an execution graph obtained from an arbitrary process graph, the next Lemma identifies the process graphs PG(N,r) comprising N tasks and r, r=1,...,N levels which result in the smallest, respectively the largest, number of stages in their corresponding execution graphs EG(X,N).

**Lemma 1**

1. Among all possible process graphs comprising N tasks (i.e., PG(N,r) for r=1,...,N), the process graph PG(N,N) gives the execution graph with the smallest number of stages; this number of stages is equal to N.

2. Among all possible process graphs comprising N tasks (i.e., PG(N,r) for r=1,...,N), the process graph PG(N,1) gives the execution graph with the largest number of stages; this number of stages is equal to $(2^N-1)$.

**Proof**

First, we prove statement one. Since the execution graph obtained from a PG(N,r) has exactly N levels, and since the minimum number of execution stages per level is one, it follows that the minimum number of stages in an execution graph is also equal to N. We now proceed to prove the second statement. It is not difficult to see that the process graph PG(N,1) gives the execution graph with the largest number of paths; and hence the largest number of stages. On the other hand, the sets $\alpha(i)$, i=1,...,L are the *Power set* of the set of tasks, without the empty set. The proof follows since the cardinality of the *Power set* is $2^N$.

∎

**Theorem 2**

The lower bound $T_{\infty,LB}$ on the expected response time of jobs having any given arbitrary process graph comprising N tasks, and such that the tasks have the same average service time, $\frac{1}{\mu}$, is given by:

$$T_{\infty,LB} = \frac{1}{\mu}\sum_{i=1}^{N}\frac{1}{i}$$

**Proof**

From equation (5), we obtain the following nonlinear program to solve:

$$\text{Minimize} \quad A = \sum_{i=1}^{L}\frac{\lambda_i}{f(i)} \qquad \text{Subject to:} \quad \begin{cases} 1. \ \sum_{i=1}^{L}\lambda_i = \lambda N \\ 2. \ f(i) \geq 1, \ \forall i=1,\ldots,L \\ 3. \ \lambda_i \leq \lambda \\ 4. \ L \geq N \end{cases}$$

Since the execution graph has N levels, we can write:

$$A = \sum_{j=1}^{N} \left\{ \sum_{i \in level j} \frac{\lambda_i}{f(i)} \right\}$$

Now, let us proceed to minimize level by level under the stated set of constraints. Since for all levels j, j=1,...,L we have $\sum_{i \in level j} \lambda_i = \lambda$, our minimization problem is equivalent to maximizing the number of stages per level and for all levels; hence maximizing the total number of stages in the execution graph. From Lemma 1, we already know that the process graph that gives the execution graph with the largest number of stages is the PG(N,1). On the other hand, the process graph PG(N,1) is the process graph where a job arrives as a bulk of N concurrent tasks.

The service time of such a bulk of N parallel tasks is $\max_{A \in \Omega}\left\{\tilde{X}_A\right\}$, and consequently:

$$T_{\infty,LB} = \int_0^{\infty} \left[1-(1-e^{-\mu t})^N\right] dt = \frac{1}{\mu} \sum_{i=1}^{N} \frac{1}{i}$$

∎

## 3 The Uniprocessor Case

In the case of a uniprocessor system, we may consider the nodes in the process graph as having assigned priority levels. This system can then be studied by means of the M/G/1 queueing system with job feedback [*]. However, we are mostly interested in the study of the discriminatory processor sharing service discipline.

Very few studies of the M/G/1 queueing system with the discriminatory processor sharing discipline have appeared in the literature and none to our best knowledge if we also have job feedback. Kleinrock [Klei67] was the first to introduce such a strategy for a single processor system with M job classes and no feedback, and provided an expression for the steady state expected response time of a class k job whose required service time is t. Under a different set of assumptions, and using a different analysis method, O'Donovan [O'Do74] obtained the same expression. More recently, Fayolle, Iasnogorodshi, and Mitrani [Fayo78] presented another solution to the same problem. Their analysis method follows O'Donovan's approach in deriving a system of integro-differential equations for the steady state expected response time of a class k job whose required service time is t. The system of equations was solved, for general distributions of the required service times, by the method of Wiener-Hopf. In the case of exponentially distributed required service times, the authors in [Fayo78] provided a system of linear equations for the unconditional steady state average response times. In this section, we first prove that the 1-DPS-WF family of scheduling strategies forms a complete family in the sense that any

[*] A comprehensive treatment of the M/G/1 queueing system with job feedback and non-preemptive, work-conserving priority scheduling is giving in Chapter 6 in [Belg86].

response time requirement that can be satisfied at all, can be achieved by a strategy from the family. Then, we proceed to investigate the job average response time in the 1-DPS-WF ayatem, and present an accurate and yet very simple approximation. Finally, we develop and prove a conservation law that puts a linear equality constraint on the set of expected system times of the different stages representing the job execution using the discriminatory processor sharing discipline with feedback.

## 3.1 Completeness of the 1-DPS-WF family of Scheduling Strategies

we now proceed to prove that the 1-DPS-WF with jobs represented by an AEC forms a complete parameterized family of scheduling strategies. A performance requirement stated in terms of the average system times of the different stage types, is said to be achievable if, given the loading conditions on the system (i.e., given the $\rho_i$, i=1,...,N), there exists a scheduling strategy which satisfies it. A family of scheduling strategies is said to be complete if every achievable performance requirement can be satisfied by a strategy from the family. Let the performance of the system, given the $\rho_i$'s, i=1,...,N, be measured by the vector $T = \left[ T_1, \ldots, T_i, \ldots, T_N \right]$. If, for a given scheduling strategy S, the value of the performance vector is T, we say that S achieves T and denote it by $S \Rightarrow T$. A given performance vector T is said to be achievable, if there exists a scheduling strategy S such that $S \Rightarrow T$ ( S need not be unique). Denote the set of all achievable performance vectors by H; we have:

$$H = \left\{ T \;\middle|\; \exists S : S \Rightarrow T \right\}$$

It is obvious that not all performance vectors (e.g., T=(0,0,...,0) ) are achievable. Let $\Phi$ be a family of scheduling strategies, and let $H_\Phi$ represent the set of all performance vectors that can be achieved using strategies from the set $\Phi$. That is :

$$H_\Phi = \left\{ T \;\middle|\; \exists S \,;\, S \in \Phi \text{ and } S \Rightarrow T \right\}$$

We say that the family $\Phi$ is complete if $H_\Phi = H$. In other words, the family $\Phi$ is complete if any performance vector T which can be achieved at all, can be achieved by a strategy from the family $\Phi$. Note that no finite or denumerable family of scheduling strategies can be complete.

Let $P_1$, $P_2$, ..., $P_{MP}$ be the performance vectors of the $MP$ preemptive priority disciplines which can be operated with the N stages. These $MP$ vectors are the vertices or "corners" of the set H. Moreover, the set H is a convex hull defined by these vertices (and is an (N-1) dimensional set because it lies on the hyperplane defined by the generalized conservation law defined and stated in Chapter 6 in [Belg86] ). In other words, the boundary of the convex hull H consists of the performance vectors which correspond to strategies giving one or more stages preemptive priority over the remaining ones. These corners of the set H represent then the extremes of the system performance (best for some stages and worst for others).

Returning to our 1-DPS-WF family of strategies, it is rather easy to see that for any given strategy in such a family (i.e., any given vector $( f(1), \ldots, f(i), \ldots, f(N) )$ ), multiplying all the f(i), i=1,...,N by the same constant does not change the strategy. Therefore one of the f(i)'s can be arbitrary fixed; let f(N)=1. We have now an (N-1) dimensional parameter set G defined by:

$$G = \left\{ (f(i),...,f(i),...,f(N-1)) \;\middle|\; f(i)>0\,;\, i=1,...,N-1 \right\}$$

Each point in the set G uniquely determines a 1-DPS-WF strategy, and consequently a performance vector T. Moreover, it is rather easy to see that this correspondence is one-to-one and continuous. Let $\Psi$ denote such a family of strategies. Since the parameter set G is open, it follows that the set $H_\Psi$ of performance vectors achievable by strategies from the family $\Psi$ is also open and therefore $H_\Psi \neq H$; that is, $\Psi$ is not complete. In fact, the performance vector of any preemptive priority discipline cannot be achieved by a 1-DPS-WF strategy because the latter would not allow a stage in the system to be completely deprived of service. However, the family $\Psi$ is *almost complete* in the sense given by the following Theorem:

**Theorem 3**

The set of performance vectors achievable by strategies from the family $\Psi$, $H_\Psi$, is equal to the set H of all achievable performance vectors without its boundary. If a performance vector T is an inside point of H, then it can be achieved by a strategy from $\Psi$; and if T is on the boundary of H, then it can be approximated as closely as desired by strategies from $\Psi$.

**Proof**

Consider the parameter subsets defined by:

$$G_{L,U} = \left\{ (f(1),...,f(i),...,f(N-1)) \;\middle|\; L \le f(i) \le U\,;\, i=1,...,N-1 \right\}$$

where L and U are positive real numbers, and let $\Psi_{L,U}$ denote the family of 1-DPS-WF strategies defined over $G_{L,U}$. We then have:

$$G = \lim_{L\to 0,\, U\to\infty} G_{L,U} \qquad \text{and} \qquad H_\Psi = \lim_{L\to 0,\, U\to\infty} H_{\Psi_{L,U}}$$

The boundary of the set $G_{L,U}$ consists of those points (f(1),...,f(i),...,f(N-1)) for which f(i)=L for at least one i and/or f(j)=U for at least one j, i=1,...,N-1 and j=1,...,N-1. Let $B_{L,U}$ be the set of performance vectors which corresponds to these boundary points in $G_{L,U}$. Now, since the set $G_{L,U}$ is compact and there is a one-to-one correspondence between the set $G_{L,U}$ and the set $\Psi_{L,U}$, then the set $H_{\Psi_{L,U}}$ consists of the set $B_{L,U}$ and all performance vectors inside it.

Moreover, the set $B_{L,U}$ is a closed and continuous surface since it is the image of a closed and continuous surface by a continuous mapping. Let T be any arbitrary performance vector such that it is an inside point of H. Because $B_{L,U}$ is continuous, then there must exist a sufficiently small L, and a sufficiently large U such that the performance vector T is an inside point of the set $B_{L,U}$. This means that T belongs to $H_{\Psi_{L,U}}$ and hence T belongs to $H_{\Psi}$.

IIII

The above Theorem states the special fact that if a performance vector T is on the boundary of the convex hull H, then it can be approximated as closely as desired by strategies from the family $\Psi$. In particular, the preemptive priority ordering $(1>2> \cdots >N)$ can be achieved by considering the point in the set G defined by f(N)=1, f(i) $\rightarrow\infty$ i=1,...,N-1, and such that $\frac{f(i)}{f(i+1)}\rightarrow\infty$, i=1,...,N-1. Likewise, the preemptive priority ordering $(1<2< \cdots <N)$ is achieved by considering the point in the set G defined by f(1)=1, f(i) $\rightarrow\infty$ i=2,...,N, and such that $\frac{f(i+1)}{f(i)}\rightarrow\infty$, i=1,...,N.

### 3.2 The Job Average Response Time

We now proceed to investigate the job average response time in the 1-DPS-WF system. Jobs are described according to a given process graph with N tasks. The process graph, in the sequel, is assumed to have only one *starting task* and only one *terminating task*; Figure 1 is an example.

We shall first determine, among all possible RAECs with N stages, the RAEC that provides the lowest job average response time, and the RAEC that results in the highest job average response time. We then discuss and present a rather accurate approximation of the job average response time. Simulations are used to back up and validate the accuracy of the approximation.

**Theorem 4**

Among all possible RAECs with N stages, the BFEC is the RAEC which provides the largest job average response time, and the DFEC is the RAEC which provides the lowest job average response time.

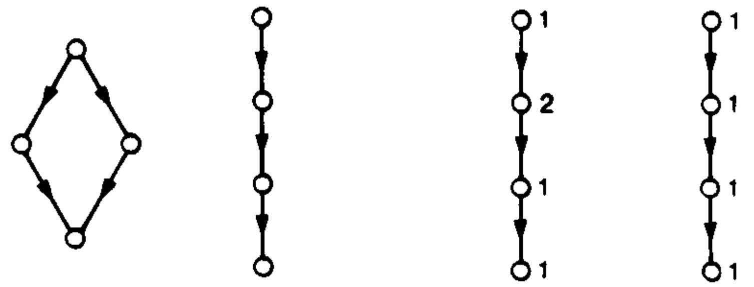
**Proof**

Among all the RAECs with N stages, the RAEC which results in the earliest job completions is the one that implicitly allocates the highest priority to the jobs having the least remaining processing time; this is the DFEC. The RAEC which results in the furthest job completions on the other hand, is the one that implicitly allocates the highest priority to the jobs having the largest remaining processing time; this is the BFEC. The proof is complete since the unfinished work in the system, at any time, is independent of the way the server capacity is shared among the different jobs present in the system (i.e., independent of the RAEC representing the jobs).

IIII

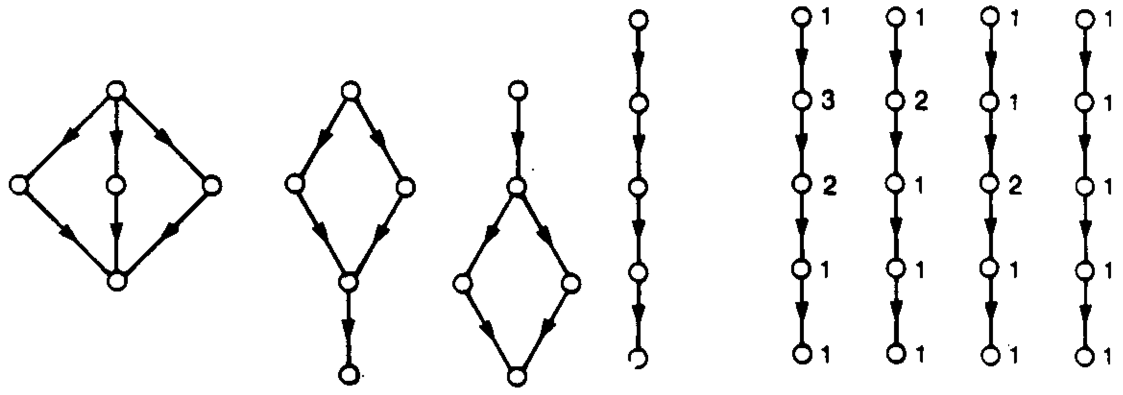
Although the BFEC and the DFEC do not correspond to any given process graph, they present, by means of Theorem 4, respectively upper and lower bounds on the job average response time. Many RAECs, among all possible RAECs with N stages, do not actually correspond to a given process graph. Indeed for N=3, the only feasible RAEC is the one with f(1)=f(2)=f(3)=1; for the number of process graphs with N tasks is one, the process graph represented by a chain of three tasks. There is also a unique process graph for N=2, the one corresponding to the RAEC with f(1)=f(2)=1. Among all the RAECs with N=4 stages, only two are feasible, for there are only two possible process graphs having N=4 tasks, and each one of them corresponds to a unique feasible RAEC. These two possible process graphs with N=4 are depicted in Figure 3:(a) and their correspondent RAECS are depicted in Figure 3:(b). For N=5, we obtain 4 possible process graphs, each of which corresponds to a unique RAEC. Figure 4:(a) represents these 4 possible process graphs, and Figure 4:(b) depicts their corresponding RAECs.

From the above, we observe that for any given N, there are only very few feasible RAECs. Moreover, the feasible RAECs are by no means extreme cases. In fact, if there exists a stage i, i=1,...,L in the execution graph such that $f(i) \geq 2$, then the stages immediately before the last in the execution graph must have a concurrency degree of one. In other words, any path in the execution graph has the inherent property that f(1)=f(L-1)=f(L)=1 for any arbitrary given process graph comprising N tasks. This inherent property of the execution graph assures that the job average response time should be much closer to the average response time given by the EEC with N stages, then to the average response time obtained by using either the BFEC or the DFEC with the same number of stages. The average response time resulting from the EEC may be considered somehow as a median among the average response times given by any feasible RAEC. For large values of N (i.e., $N \geq 5$), the resulting execution graph may have many execution paths. These paths are obviously feasible RAECs with N stages. Due to the inherent property of the execution graph, namely that f(1)=f(L-1)=f(L)=1, we may suggest that the majority of the RAECs representing the execution paths provide somehow slightly larger values of the average response time than the one resulting by using the EEC with the same number of stages. Consequently, the EEC represents an optimistic and good approximation of the average response time of jobs having any arbitrary given process graph.



(a): Process Graphs (b): Execution Graphs

Figure 3: Process Graphs and their Corresponding Execution Graphs for N=4



(a): Process Graphs (b): Execution Graphs

Figure 4: Process Graphs and their Corresponding Execution Graphs for N=5

From Proposition 1, we know that the steady state probabilities $P[n_1, \ldots, n_i, \ldots, n_L]$ that the uniprocessor system is in state $(n_1, \ldots, n_i, \ldots, n_L)$ have a *Product Form* solution when the jobs are represented in the system by their EEC. The uniprocessor system can be seen as a single node BCMP [Bask75, Chan77] queueing network with N classes. A job in the system is of class i, i=1,...,N if it is at its ith execution stage. Since the service time of task i, i=1,...,N is assumed to be exponentially distributed with an average $\frac{1}{\mu_i}$, and since $\lambda_i = \lambda$ for i=1,...,N, it follows that [*]:

$$P[n_1, \ldots, n_i, \ldots, n_N] = P[0,\ldots,0,\ldots,0](\sum_{i=1}^{N} n_i)! \prod_{i=1}^{N} \frac{(\rho_i)^{n_i}}{n_i!} \qquad n_i \geq 0 \quad i=1,\ldots,N \tag{5}$$

where $\rho_i = \frac{\lambda}{\mu_i}$, i=1,...,N. Let $n = \sum_{i=1}^{N} n_i$, and P[n] be the steady state probability that there are n jobs in the system. From equation (5), we obtain:

$$P[n] = P[0]\,\rho^n \qquad n \geq 0$$

where $\rho = \sum_{i=1}^{N} \rho_i = \sum_{i=1}^{N} \frac{\lambda}{\mu_i}$, and since $\sum_{i=1}^{N} P[n] = 1$, we get:

$$P[n] = (1-\rho)\rho^n \qquad n \geq 0 \tag{6}$$

and consequently, if we let $\bar{n}$ denote the average number of jobs in the system in the steady state, we get from equation (6) and using the fact that $\bar{n} = \sum_{i=0}^{\infty} nP[n]$ :

$$\bar{n} = \frac{\rho}{1-\rho}$$

Since $\lambda = \frac{\rho}{\sum_{i=1}^{N} \frac{1}{\mu_i}}$, and using Little's result; namely that $\bar{n} = \lambda T(1)$, where T(1) denotes the average response time, we have:

$$T(1) = \frac{\sum_{i=1}^{N} \frac{1}{\mu_i}}{1-\rho} \tag{7}$$

[*] For the sake of clarity and since we are working on the EEC, we are using here numbers instead of letters to identify the different tasks.

Equation (7) represents an approximation (an optimistic approximation) of the job average response time of jobs having any arbitrary given process graph comprising N tasks. To validate this approximation, we simulated the 1-DPS-WF system using the process graph description depicted in Figure 1. The method used to estimate the extent of the simulation transient state is *the method of independent replications* [Lave83], and the method used to estimate the statistic T(1) for the 1-DPS-WF system in the steady state is *the method of batch means* [Lave83]. Figure 5 depicts the job average response time given by equation (7) along with the simulation results represented by the confidence intervals. The confidence intervals are depicted in Figure 5 as vertical bars, are obtained from the simulation output via the t-distribution, and are at the 90% level.

From Figure 5, we observe that equation (7) represents a very accurate approximation of the job average response time in the 1-DPS-WF system and for jobs having the process graph depicted in Figure 1. Over all permissible ranges of the system utilization factor, the approximation is well within the 90% confidence intervals. It is also rather interesting to notice the narrowness of these confidence intervals. The optimistic character of the approximation may be observed on Figure 5 for moderate values of the system utilization factor (the average response time curve intersects the confidence intervals at their lower parts). For either small or high values of the utilization factor, such an optimistic behavior is much less noticeable.

## 3.3 The Conservation Law

In this section, we develop a conservation law that puts a linear equality constraint on the set of average system times of the different stages in the abstracted execution chain that represents the job execution in the uniprocessor system using the discriminatory processor sharing discipline with feedback. In [Klei76], Kleinrock established the conservation law for the M/G/1 queueing system and for any non-preemptive work-conserving queueing discipline. In a similar fashion, we shall use the fact that the unfinished work in the system is invariant to the sharing of the sever capacity, provided that the sharing discipline does not explicitly rely on information about the remaining processing time of any job in the system. In [Belg86], the authors provided a generalization of the conservation law for any M/G/1 queueing system with job feedback, and any non-preemptive work-conserving priority scheduling of the different stages constituting the job.

Let $T_i$, i=1,...N represent the average time spent in the uniprocessor system by stage i from the time of its arrival to the system until its completion. For stage i, i=1,...,N, let $\frac{1}{\mu_i}$ be the stage average processing time, and $\rho_i = \frac{\lambda_i}{\mu_i}$ be the uniprocessor utilization due to stages of type i. We consider that all jobs have the same AEC, and hence $\lambda_i = \lambda$, i=1,...,N, and $\rho_i = \frac{\lambda}{\mu_i}$,

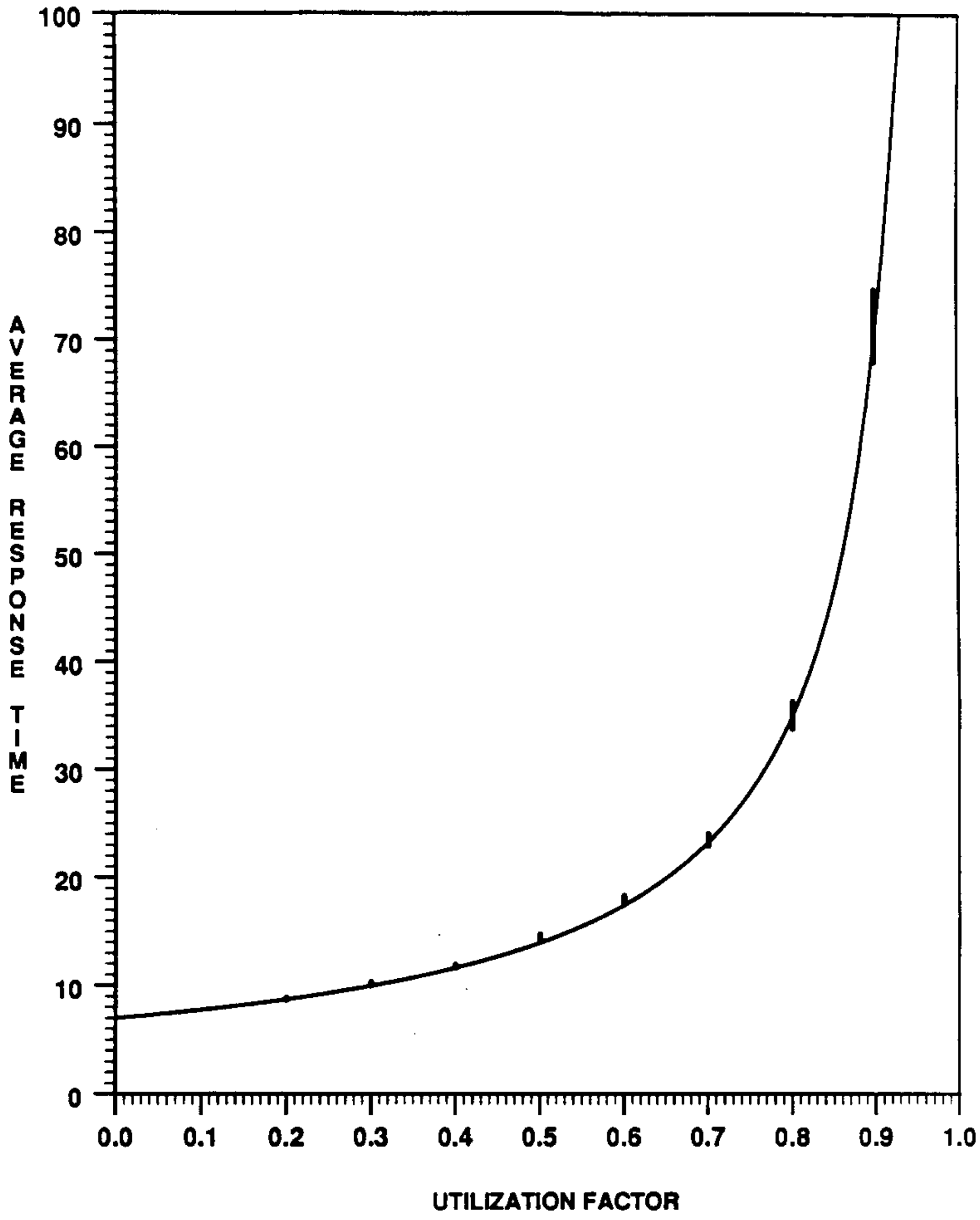


Figure 5: Average Response Time in a 1-DPS-WF System

i=1,...,N. Recall that the f(i), i=1,...,N are arbitrary and continuous positive real values.

**Theorem 5 : The 1-DPS-WF Conservation Law**

For an exponential work-conserving uniprocessor system using the discriminatory processor sharing discipline with jobs described by an abstracted execution chain, it must be, for any choice of the concurrency degrees f(i), i=1,...,N, that:

$$\sum_{i=1}^{N}\left[\sum_{j=i}^{N}\rho_j\right]T_i = \frac{\rho}{1-\rho}\,\frac{\sum_{j=1}^{N}\frac{j}{\mu_j}}{N}$$

where $\rho = \sum_{i=1}^{N}\rho_i < 1$.

**Proof**

let us first prove that $\sum_{i=1}^{N}\left[\sum_{j=i}^{N}\rho_j\right]T_i = \text{constant}$. Let $\overline{U}$ denote the average unfinished work in the system, and $P[n_1,\ldots,n_i,\ldots,n_N]$ denote the steady state probability that there are $n_i$, i=1,...,N stages of type i in the system. A stage of type i found in the system participates, in the unfinished work, by its remaining service time plus the service time of its fed back stages; namely stages i+1,...,N. Therefore, if the state of the system is $(n_1,\ldots,n_i,\ldots,n_N)$, and by using the Markovian property of the exponential service time distributions, the unfinished work in the system, given such a state, is $\sum_{i=1}^{N} n_i \sum_{j=i}^{N}\frac{1}{\mu_j}$. Consequently, the average unfinished work in the system, $\overline{U}$, is given by:

$$\overline{U} = \sum_{n_1=0}^{\infty}\cdots\sum_{n_i=0}^{\infty}\cdots\sum_{n_N=0}^{\infty} P[n_1,\ldots,n_i,\ldots,n_N]\left[\sum_{i=1}^{N} n_i \sum_{j=i}^{N}\frac{1}{\mu_j}\right]$$

which amounts to:

$$\overline{U} = \sum_{i=1}^{N}\overline{n_i}\sum_{j=i}^{N}\frac{1}{\mu_j}$$

where $\overline{n_i}$ is the average number of jobs in execution stage i in the system in steady state. Using Little's result [Litt61], namely that $\overline{n_i} = \lambda_i T_i$, i=1,...,N and recalling that the unfinished work in the system is invariant to the sharing discipline of the server capacity among the stages present in the system, completes our first proof. Now let us take the special case of an EEC chain to represent the job description. For this case, we have:

$$T_i = \frac{\sum_{j=1}^{N} \frac{1}{\mu_j}}{1-\rho} \frac{1}{N} \qquad i=1,\ldots,N$$

Using these values completes the proof.

∎

In the sequel, we shall develop an approximation for the job average response time in the P-DPS-WF multiprocessing system based on the EEC approximation. But first, we shall investigate the P-DPS-WF multiprocessing system where all the stages in the execution graph have the same concurrency degree. Results from such a system will also be used in the P-DPS-WF average response time approximation.

## 4 The P-DPS-WF Multiprocessing System with the Same Concurrency Degree for all Stages

In this section, we study the P-DPS-WF multiprocessing system where jobs are represented by a given execution graph with stages having the same concurrency degree, denoted hereafter by F (i.e., f(i)=F, i=1,...,L). A job is therefore described by a chain of N execution stages with the same concurrency degree F. Let the state of the system be $S=(n_1,\ldots,n_i,\ldots,n_N)$ where $n_i$, i=1,...,N represents the number of jobs which are present in the system and are at their execution stage number i. We shall refer to $n_i$, i=1,...,N as the number of class i jobs. Let $\rho_i$, i=1,...,N denote the utilization factor of the multiprocessing system due to class i jobs, and $\rho$ denote the total utilization factor of the system. Hence we have $\rho = \sum_{i=1}^{N} \rho_i$. To evaluate the $\rho_i$'s, i=1,...,N, recall [Klei75] that the utilization factor is in a fundamental sense the ratio of the rate at which *work* enters the system to the maximum rate (i.e., capacity) at which the system can perform this work. The work an arriving customer brings to the system equals the number of seconds of service he requires. If $\frac{1}{\mu_i}$, i=1,...,N represents the average service demand of a class i job, we then have $\rho_i = \frac{\lambda}{\mu_i}$, i=1,...,N independently of the value of F. Therefore, we have:

$$\rho = \frac{\lambda}{P} \sum_{i=1}^{N} \frac{1}{\mu_i} \tag{8}$$

Since from Proposition 1, we know that the multiprocessing system under investigation has a *product form* solution, we can use the $M \Rightarrow M$ conditions [Munt73, Bask75, Chan77] to determine the steady state probabilities $P[n_1, \ldots, n_i, \ldots, n_N]$. If the system state is $(n_1, \ldots, n_i, \ldots, n_N)$, then the departure rate of class i jobs, denoted hereafter by $d_i$, is given by:

$$d_i = \frac{n_i F P}{\max \left\{ P, \sum_{j=1}^{N} n_j F \right\}} \mu_i \quad i=1,\ldots,N$$

since $\dfrac{n_i F P}{\max \left\{ P, \sum_{j=1}^{N} n_j F \right\}}$ is the proportion of the processors capacity allocated to class i jobs. The above equation may be rewritten as:

$$d_i = \frac{n_i P \mu_i}{\max \left\{ \frac{P}{F}, \sum_{j=1}^{N} n_j \right\}} \quad i=1,\ldots,N \tag{9}$$

Now using the $M \Rightarrow M$ conditions [Munt73], we have:

$$\frac{P[n_1, \ldots, n_i+1, \ldots, n_N] \dfrac{(n_i+1) P \mu_i}{\max \left\{ \frac{P}{F}, 1 + \sum_{j=1}^{N} n_j \right\}}}{P[n_1, \ldots, n_i, \ldots, n_N]} = \lambda_i$$

for all i=1,...,N and for all the feasible states. Let $\chi$ denote the set of all the feasible states; that is $\chi = \left\{ (n_1, \ldots, n_i, \ldots, n_N) \,\middle|\, \forall i=1,\ldots,N \;\; n_i \geq 0 \right\}$. The above equation yields:

$$P[n_1, \ldots, n_i+1, \ldots, n_N] = \frac{\lambda_i}{\mu_i P} \frac{1}{n_i+1} \max \left\{ \frac{P}{F}, 1 + \sum_{j=1}^{N} n_j \right\} . P[n_1, \ldots, n_i, \ldots, n_N] \tag{10}$$

$$\forall i=1,\ldots,N \text{ and } \forall (n_1, \ldots, n_i, \ldots, n_N) \in \chi$$

Using equation (10) repetitively and starting from $P[1,0,\ldots,0]$, we can express the probability of any feasible state as a function of the probability $P[0,\ldots,0]$ of the system being empty. We obtain:

$$P[n_1, \ldots, n_i, \ldots, n_N] = P[0,\ldots,0] \prod_{i=1}^{N} \left\{ \left[ \frac{\lambda_i}{\mu_i P} \right]^{n_i} \frac{1}{n_i!} \prod_{j=1}^{n_i} \max \left\{ \frac{P}{F}, \sum_{k=1}^{i-1} n_k + j \right\} \right\} \tag{11}$$

$$\forall (n_1, \ldots, n_i, \ldots, n_N) \in \chi$$

Now since we have:

$$\prod_{i=1}^{N}\left\{\prod_{j=1}^{n_i}\max\left\{\frac{P}{F},\ \sum_{k=1}^{i-1} n_k+j\right\}\right\}=\prod_{j=1}^{\sum_{i=1}^{N} n_i}\max\left\{\frac{P}{F},\ j\right\}$$

and by letting $n=\sum_{i=1}^{N} n_i$, equation (11) yields:

$$P[n_1,\ldots,n_i,\ldots,n_N]=P[0,\ldots,0]\ \frac{\prod_{j=1}^{n}\max\left\{\frac{P}{F},\ j\right\}}{P^n}\prod_{i=1}^{N}\left[\frac{\lambda_i}{\mu_i}\right]^{n_i}\frac{1}{n_i!} \qquad (12)$$

$$\forall (n_1,\ldots,n_i,\ldots,n_N)\in\chi$$

Equation (12) gives the probability of any feasible state as a function of the probability $P[0,\ldots,0]$ of the system being empty. In the sequel, we develop a closed form expression for the probability P[n] of having a total of n jobs in the system in steady state. Let $m=\lfloor \frac{P}{F} \rfloor$ denoting the largest integer less or equal to $\frac{P}{F}$. We distinguish two cases depending on the value of m.

**Case of $n\leq m$**

Since $\prod_{j=1}^{n}\max\left\{\frac{P}{F},\ j\right\}=\left[\frac{P}{F}\right]^n$, equation (12) gives:

$$P[n_1,\ldots,n_i,\ldots,n_N]=P[0,\ldots,0]\ \frac{1}{F^n}\prod_{i=1}^{N}\left[\frac{\lambda_i}{\mu_i}\right]^{n_i}\frac{1}{n_i!}$$

$$\forall (n_1,\ldots,n_i,\ldots,n_N)\in\chi \text{ and } n\leq m$$

Consequently, we have:

$$P[n]=\frac{P[0,\ldots,0]}{F^n}\sum_{s.t.\ \sum_{i=1}^{N} n_i=n}\left\{\prod_{i=1}^{N}\left[\frac{\lambda_i}{\mu_i}\right]^{n_i}\frac{1}{n_i!}\right\}$$

$$=\frac{P[0,\ldots,0]}{F^n n!}\left[\sum_{i=1}^{N}\frac{\lambda_i}{\mu_i}\right]^n$$

since P[0,...,0]=P[0], and $\sum_{i=1}^{N}\frac{\lambda_i}{\mu_i}=P\rho$, the above equation yields:

$$P[n] = P[0]\,\frac{\left[\frac{P\rho}{F}\right]^n}{n!} \qquad \forall\ n \le m \tag{13}$$

**Case of $n > m$**

Since $\prod_{j=1}^{n} \max\left\{\frac{P}{F}, j\right\} = \left[\frac{P}{F}\right]^m \frac{n!}{m!}$, equation (12) yields:

$$P[n_1, \ldots, n_i, \ldots, n_N] = P[0,\ldots,0]\left[\frac{P}{F}\right]^m \frac{n!}{m!}\frac{1}{P^n}\prod_{i=1}^{N}\left[\frac{\lambda_i}{\mu_i}\right]^{n_i}\frac{1}{n_i!}$$

$$\forall (n_1, \ldots, n_i, \ldots, n_N) \in \chi \text{ and } n > m$$

Consequently and after some algebra, we obtain:

$$P[n] = P[0]\,\frac{\left[\frac{P}{F}\right]^m}{m!}\rho^n \qquad \forall\ n > m \tag{14}$$

We now proceed to evaluate the steady state probability P[0] of the system being empty. Using the fact that $\sum_{n=0}^{\infty} P[n] = 1$, and equations (13) and (14), we have:

$$P[0]^{-1} = \sum_{n=0}^{m}\left[\frac{P\rho}{F}\right]^n \frac{1}{n!} + \sum_{n=m+1}^{\infty}\frac{\left[\frac{P}{F}\right]^m}{m!}\rho^n$$

which after some algebra yields:

$$P[0]^{-1} = \sum_{n=0}^{m-1}\left[\frac{P\rho}{F}\right]^n \frac{1}{n!} + \frac{\left[\frac{P\rho}{F}\right]^m}{m!}\,\frac{1}{1-\rho} \tag{15}$$

Equations (13), (14) and (15) provide explicit expressions of the steady state probability P[n] of having n jobs in the system. Let $\bar{n}$ denote the steady state average number of jobs in the system. Using equations (13), (14), and (15), we have:

$$\bar{n} = \sum_{n=0}^{\infty} nP[n]$$

$$= \sum_{n=0}^{m} nP[0]\,\frac{\left[\frac{P\rho}{F}\right]^n}{n!} + \sum_{n=m+1}^{\infty} nP[0]\,\frac{\left[\frac{P}{F}\right]^m}{m!}\rho^n$$

which after some algebra, yields:

$$\bar{n} = \frac{P\rho}{F} + P[0]\frac{\left[\frac{P\rho}{F}\right]^m}{m!}\frac{\rho}{(1-\rho)^2} + P[0]\frac{\left[\frac{P\rho}{F}\right]^m}{m!}(m-\frac{P}{F})\frac{\rho}{1-\rho} \qquad (16)$$

Equation (16) along with the expression of the probability P[0] given by equation (15), explicitly defines the steady state average number of jobs in the P-DPS-WF system where jobs are represented by any given execution graph with stages having the same concurrency degree F. The job average response time, T(P), in such a P-DPS-WF system can then be obtained by using Little's result; namely that $T(P) = \frac{\bar{n}}{\lambda}$, where λ is the aggregate rate of the job Poisson arrival process. Using equation (8) and equation (16), we thus have:

$$T(P) = \frac{\sum_{i=1}^{N}\frac{1}{\mu_i}}{F} + P[0]\frac{\left[\frac{P\rho}{F}\right]^m}{m!}\frac{\sum_{i=1}^{N}\frac{1}{\mu_i}}{P(1-\rho)^2} + P[0]\frac{\left[\frac{P\rho}{F}\right]^m}{m!}(m-\frac{P}{F})\frac{\sum_{i=1}^{N}\frac{1}{\mu_i}}{P(1-\rho)} \qquad (17)$$

**Example**

Let us take the case of F=1; this is the usual Processor Sharing scheduling strategy. We have m=P, and equation (16) becomes:

$$\bar{n} = P\rho + P[0]\frac{\left[\frac{P\rho}{F}\right]^P}{P!}\frac{\rho}{(1-\rho)^2}$$

where, by using equation (15), P[0] is given by:

$$P[0]^{-1} = \sum_{n=0}^{P-1}\frac{(P\rho)^n}{n!} + \frac{(P\rho)^P}{P!}\frac{1}{1-\rho}$$

This is the known solution of the average number of jobs in the M/M/P FCFS queueing system and the M/G/P Processor Sharing queueing system.

$\blacksquare$

**Limiting Behavior of T(P)**

It is of interest to determine the limiting values $T_0(F,P)$ and $T_1(F,P)$ of the job average response time as the utilization factor ρ approaches respectively zero and one, for any real positive value of the concurrency degree F. As ρ approaches zero, the job average response time approaches the total job average processing time through an empty system. Since the average processing time of execution stage i, i=1,...,N is $\frac{\max\left\{P,F\right\}}{FP}\frac{1}{\mu_i}$, we obtain :

$$T_0(F,P) = \frac{\max\left\{P,F\right\}}{PF} \sum_{i=1}^{N} \frac{1}{\mu_i} \tag{18}$$

On the other hand, the limiting value $T_1(F,P)$ is specified in the following Theorem.

**Theorem 6**

The limiting value $T_1(F,P)$ of the job average response time through a P-DPS-WF system with a constant concurrency degree F, the same for all the stages, is independent of F and is equal to the average response time in an M/M/1 queueing system having the same utilization factor ρ; that is:

$$T_1(F,P) = \frac{\rho}{1-\rho} \frac{1}{\lambda}$$

**Proof**

Let us prove that as ρ approaches one, the average number of jobs in the system as given by equation (16) approaches the average number of jobs in an M/M/1 queueing system having the same utilization factor. From equation (15), we have:

$$\lim_{\rho \to 1} P[0] = \frac{m!}{\left[\frac{P\rho}{F}\right]^m}(1-\rho)$$

Replacing P[0] by this limiting behavior in the expression for $\bar{n}$ as given by equation (16) yields:

$$\frac{\frac{\rho}{1-\rho}}{\bar{n}} = 1$$

which completes the proof.

∎

Let us now return to equation (9) providing the departure rate of class i jobs, i=1,...,L when the system state is $(n_1, \ldots, n_i, \ldots, n_L)$. We can rewrite equation (9) as follows:

$$d_i = \frac{n_i \frac{P}{F}}{\max\left\{\frac{P}{F}, \sum_{j=1}^{L} n_j\right\}} F\mu_i \qquad i=1,\ldots,N \tag{19}$$

We recognize this as the rate of departure of class i jobs from a processor sharing system comprising $\frac{P}{F}$ processors, and where the average service demand of class i jobs is $\frac{1}{F\mu_i}$. This amounts then to a decrease in both the number of processors and the average service demand of class i jobs, i=1,...,N when $F>1$, and to an increase in both the number of processors and the average service time demand of class i jobs, i=1,...,N when $F<1$.

Figure 6 depicts the job average response time given by equation (17), as a function of the system total utilization factor $\rho$, and for various values of the constant concurrency degree F. Assume, for example, that we start with P=20 processors, N=1 and $\mu = 0.05$. Therefore, for F=1, we have the usual processor sharing multiprocessor system whose job average response time is depicted by the curve that intersects the y-axis at the value 20. For F=2, we obtain the curve that intersects the y-axis at the value 10. This is the curve of the job average response time in a processor sharing system comprising 10 processors and where the job average service demand is 0.1. For F=20, we obtain the curve that intersects the y-axis at the value 1. This is the curve of the job average response time in a processor sharing uniprocessor system, where the job average service demand is 1. Consequently, we may conclude that it is much better to have a system comprising less processors with shorter job service demands than a system comprising more processors but with larger job service demands.

The other average response time curves in Figure 6, are obtained using values of F smaller than unity. For $F = \frac{1}{2}$ for example, we obtain the curve that intersects the y-axis at the value 40. This is then the job average response time in a processor sharing system comprising 40 processors, and where the job average service demand is 2.

## 5 Average Response Time in the P-DPS-WF System

We now proceed to investigate the job average response time in a P-DPS-WF multiprocessor system. Jobs are represented by a given arbitrary process graph with N tasks and comprising only one starting task and one terminating task. We shall present and analyze a rather accurate approximation for the job average response time. This approximation is based on the EEC and the P-DPS-WF systems with constant concurrency degree, the same for all execution stages. Simulations are used to validate the approximation. First, we shall deduce the exact value of the average response time of jobs through an empty P-DPS-WF system (i.e., $\rho = 0$), and then we provide a generalization of Theorem 4.

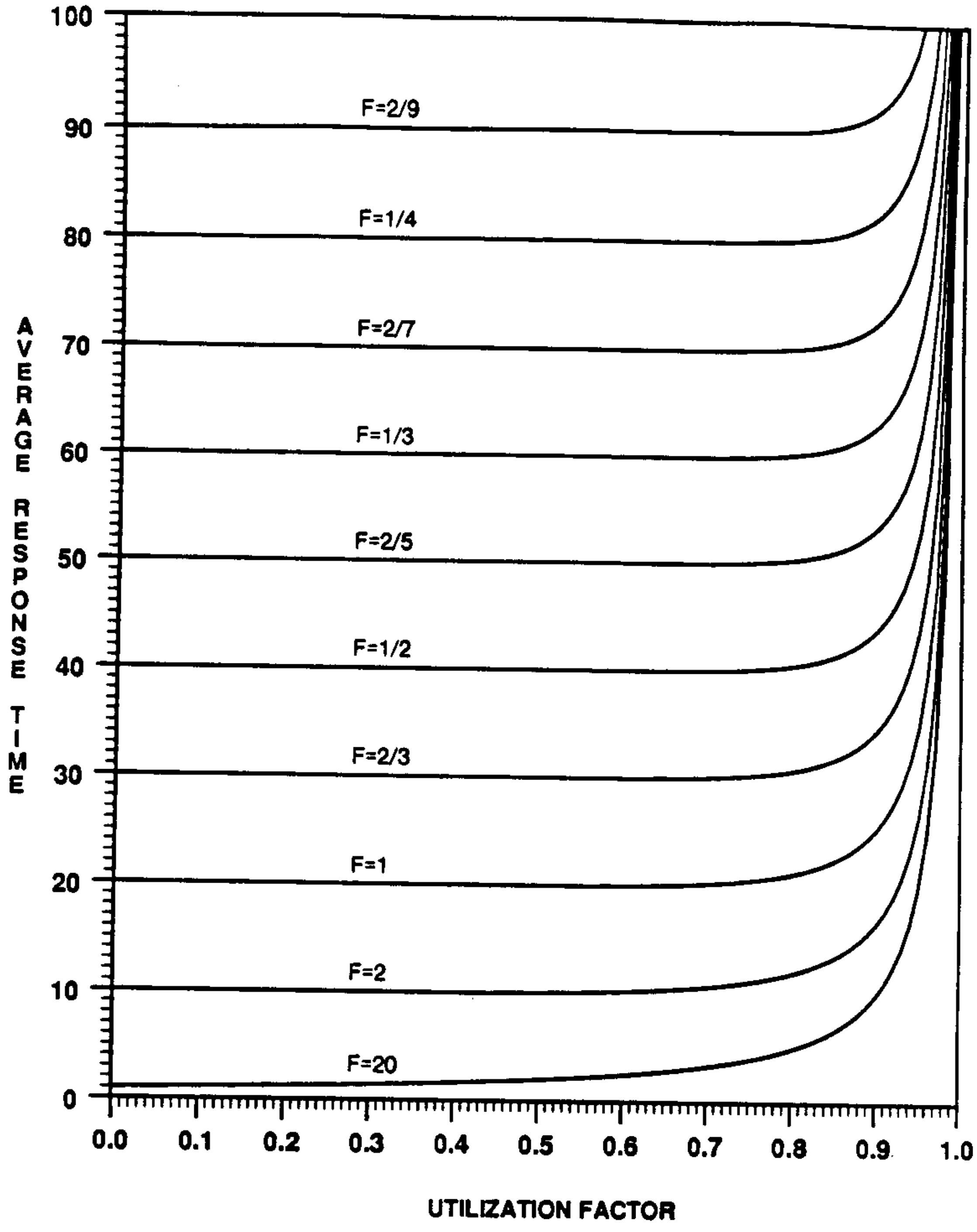


Figure 6: Average Response Time in a P-DPS-WF Multiprocessor System with a Constant Concurrency Degree

## 5.1 Job Average Response Time Through an Empty P-DPS-WF System

The job average response time through an empty system, denoted hereafter by $T_0(P)$, is equal to the average processing time needed by a job when it is alone in the system. Since if $\tilde{X}$ and $\tilde{Y}$ are independent exponentially distributed random variables with respective means $\frac{1}{\mu_X}$ and $\frac{1}{\mu_Y}$, and if $\tilde{Z}$ is the random variable defined by $\tilde{Z} = \min(\tilde{X}, \tilde{Y})$ then $P\{\tilde{Z} \le z\} = 1 - e^{-(\mu_X + \mu_Y)z}$, $z \ge 0$. It follows that the average of the random variable $\tilde{Z}$ is $\bar{Z} = \frac{1}{\mu_X + \mu_Y}$. Consequently, since whenever the first task among the set $\alpha(i)$ completes service, execution stage i terminates, then the average service demand brought by stage i to the system is given by $\frac{1}{\sum_{A \in \alpha(i)} \mu_A}$, i=1,...,L. On the other hand, the processing rate (i.e., capacity) at which this service demand is serviced depends on whether f(i) is greater than P. If f(i) is less or equal to P then each task in the set $\alpha(i)$ is processed at a rate of one second per second. If f(i) is greater than P, however, each task in the set $\alpha(i)$ is processed at a rate of $\frac{P}{f(i)}$ seconds per second. If $S_0(i)$ denotes the average processing time of stage i in an empty system, we have:

$$S_0(i) = \frac{\max\left\{P, f(i)\right\}}{P} \frac{1}{\sum_{A \in \alpha(i)} \mu_A} \qquad i=1,\dots,L \tag{20}$$

On the other hand, since the probability that stage i is visited during the execution of a job is given by $\frac{\lambda_i}{\lambda}$, i=1,...,L. We thus obtain:

$$T_0(P) = \sum_{i=1}^{L} \frac{\lambda_i}{\lambda} S_0(i) \tag{21}$$

Equation (21) along with equation (20), provides the exact value of the total average processing time needed to complete a job through an empty P-DPS-WF system. If $P = \infty$, equation (21) reduces to equation (4). In the case of $\mu_A = \mu$ for all $A \in \Omega$, equation (20) reduces to:

$$S_0(i) = \frac{\max\left\{P, f(i)\right\}}{P f(i) \mu} \qquad i=1,\dots,L$$

and equation (21) becomes:

$$T_0(P) = \frac{1}{\lambda P \mu} \sum_{i=1}^{L} \frac{\lambda_i \max \left\{ P, f(i) \right\}}{f(i)}$$

## 5.2 Approximation of the Job Average Response Time

We now proceed to determine which RAEC, among all possible RAECs with N stages, provides the upper (respectively the lower) bound on the job average response time.

**Theorem 7**

Among all possible RAECs with N execution stages, the DFEC is the RAEC which provides the lowest job average response time. Moreover, there must exists a value $\rho^*$ of the system utilization factor such that for all $\rho \in [0,\rho^*]$, the EEC is the RAEC which provides the largest job average response time, and for $\rho \in [\rho^*,1)$, the BFEC is the RAEC which provides the largest job average response time.

**Proof**

The proof is based on a sample path representation of the job arrival and departure patterns. For a given sample of the job arrival process, the RAEC which results in the earliest job departures is the DFEC, for it implicitly and dynamically allocates the highest priority to the jobs nearest to completion. This proves the first statement. For $\rho=0$, the DFEC and the BFEC provide the same average response time which is, on the other hand, smaller than the one provided by the EEC. From Theorem 4, we know that the BFEC is the worst RAEC in the 1-DPS-WF system. Since for $\rho \rightarrow 1$, the P-DPS-WF system becomes congested and thus behaves as a 1-DPS-WF system with a total capacity of P seconds per second, it follows that for a sufficiently high value of the utilization factor, the BFEC results in the largest job average response time.

IIII

Although the BFEC and the DFEC do not actually correspond to any given process graph, they present, by means of the above Theorem, and for certain ranges of the utilization factor, respectively upper and lower bounds on the job average response time. Moreover, given the process graph description, Theorem 7 states that there exists a certain value $\rho^*$ such that for all $\rho \in [0,\rho^*]$, the job average response time lies between the one given by the EEC and the one given by the DFEC; and for all $\rho \in [\rho^*,1)$, the job average response time lies between the one given by the BFEC and the one given by the DFEC.

Recall that among all possible RAECs with N execution stages, there are only a few feasible ones. Moreover, the inherent property of the execution graph, namely that f(1)=f(L-1)=f(L)=1, assures that the job average response time for high values of the system utilization factor is much closer to the average response time given by the EEC then to the one obtained by either using the BFEC or the DFEC. The approximation of the average response time in a P-DPS-WF system must satisfy the following:

1. For $\rho = 0$, the approximation should provide the same value as the one given by equation (21). Moreover, for small values of the utilization factor, the approximation should result in a curve lying between the curves provided respectively by the EEC and the DFEC.

2. For higher values of the utilization factor (i.e., $\rho > \rho^*$), the approximation should result in a curve that is very close to the curve provided by the EEC.

The above two requirements can be satisfied by using the P-DPS-WF system job average response time given by equation (17) with the proper value for the concurrency degree F. Therefore, we have a parametric approximation that depends only on the parameter F. For any given process graph, we start by computing the value of the job average response time $T_0(P)$ through an empty P-DPS-WF using equation (21). Now, since the concurrency degree F is less or equal to P, the number of processors in the system, by equating the just computed value of $T_0(P)$ to $T_0(F,P)$ given by equation (18), gives the value of F:

$$F = \frac{\sum_{i=1}^{N} \frac{1}{\mu_i}}{T_0(P)} = \frac{T_0(1,P)}{T_0(P)}$$

where $T_0(P)$ is given by equation (21). Notice that for P=1, the value of F is one and, hence our approximation results in the one presented earlier for the uniprocessor case.

We simulated the P-DPS-WF system using the process graph description depicted in Figure 1, and for the values P=2,3,4,5 and 16. The task average service time is equal to unity and the same for all the seven tasks. The *Method of Independent Replications* [Lave83] is used to estimate the extent of the simulation transient state, and the *Method of Batch means* [Lave83] is used to estimate the job average response time T(P) in the steady state. Figure 7 depicts the job average response time as given by equation (17), along with the simulation results represented by the confidence intervals. The average response time in an empty system using equation (21) is 5.208333 for P=2 and 5.055555 for $P \geq 3$. From these values, we get F=1.344 for P=2 and F=1.3846 for $P \geq 3$. The confidence intervals are depicted on Figure 7 as vertical bars, are obtained from the simulation using the t-distribution, and are of 90% level.

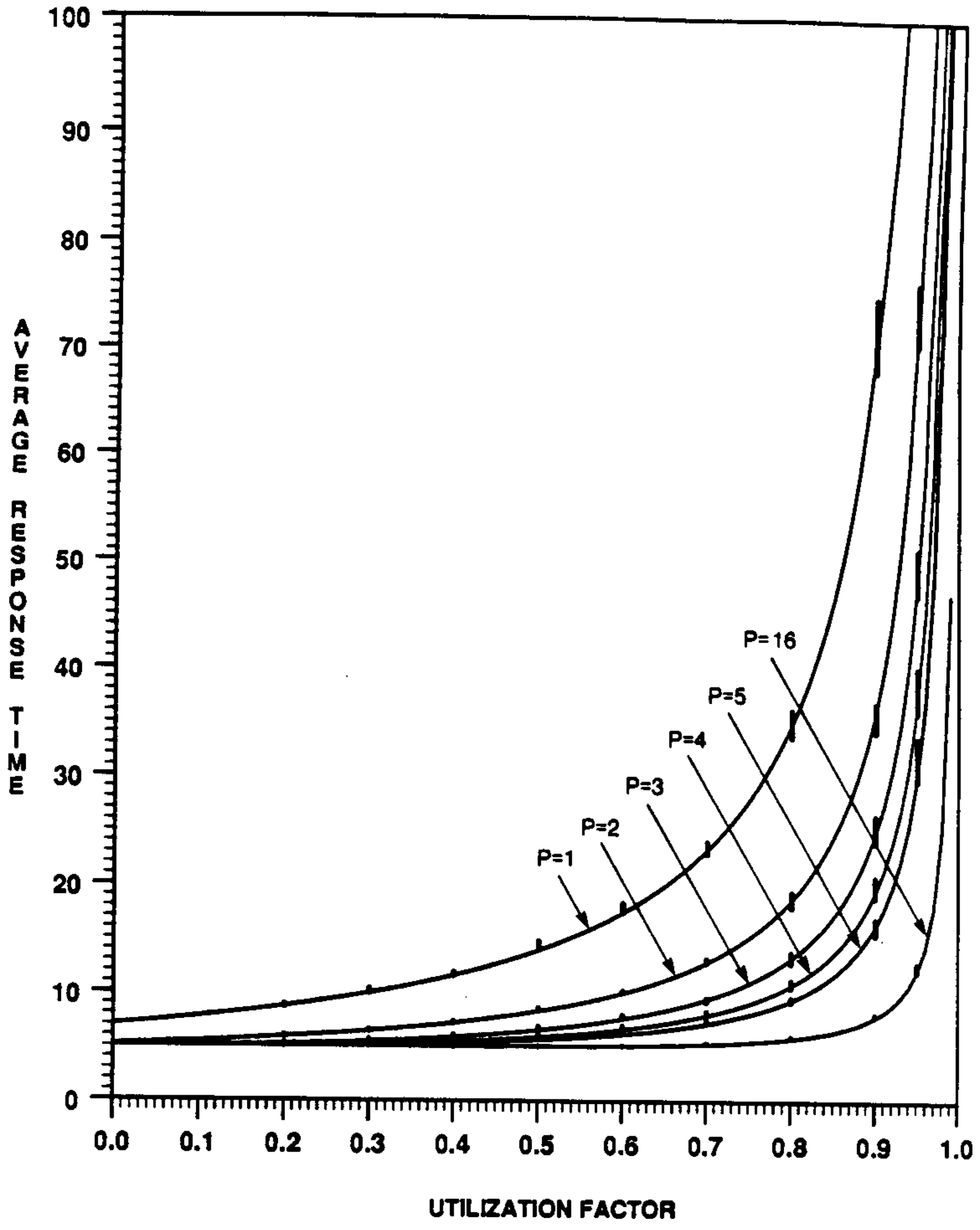


Figure 7: Average Response Time in a P-DPS-WF Multiprocessor System

Over all the permissible range of the utilization factor (i.e., $\rho \in [0,1)$), the approximation is well within the 90% confidence intervals. It is rather interesting to notice the narrowness of these confidence intervals even for very high values of $\rho$. The optimistic character on the other hand, is less noticeable than in the uniprocessor case.

## 5.3 Achievable Parallelism in a P-DPS-WF System

Significant reductions in the job average response time can be realized by executing a job, described by a given process graph, on a multiprocessor system. This effect is known as the *Speedup factor* (see below), which typically increases with the number of processors composing the multiprocessing system. Along with an increase in the speedup factor, comes a decrease in the efficiency of the processors. As more processors are used, the total amount of processors idle time increases also. While a large speedup factor may appear as a delight for the users, the efficiency of the processors is also very important. It is rather easy to get an efficiency of one, but this system is extremely slow (e.g., the job average response time is too large). This tradeoff is investigated below by the use of the *Power* function as defined in [Klei79, Gail83]. First, we shall investigate the achievable parallelism attained by a multiprocessor system with P processors, and then we shall return to investigate the above tradeoff.

Customerly, the speedup factor, denoted by $\sigma$, of a parallel processing system is defined as the ratio of the job total processing time through an empty uniprocessor system to the job total processing time through a empty multiprocessor system [Kung84, Hwan84]. This is the same definition as the concurrency degree used previously. It is not difficult to realize that the speedup factor (in the special case of exponentially and identically distributed task service requirements) is bounded above by $\frac{N}{\ln N}$. Indeed for the process graph with N concurrent tasks, and for the case where the service time per task is exponentially distributed with mean $\frac{1}{\mu}$, the same for all tasks, Theorem 2 readily gives the lower bound on the average response time using an infinite number of processors. For large N (i.e., N >> 1 ), $T_{\infty,LB} \cong \frac{\ln N + \Phi}{\mu}$ where $\Phi$ is the Euler's constant (i.e., $\Phi = 0.57721...$), and the job processing time in a uniprocessor system is $\frac{N}{\mu}$.

Although the speedup factor, as defined above, represents a very useful measure in determining the process graph structural parallelism (i.e., the inherent parallelism within the job process graph), it does not portray the achievable parallelism obtainable by using a multiprocessor system, for it does not incorporate any measure of the queueing effects. It is therefore more interesting for our purposes to redefine the speedup factor as a function of the number P of processors used, the system utilization factor $\rho$, and the scheduling strategy adopted. For a given scheduling strategy S, we therefore define the speedup factor to be:

$$\sigma(S,P,\rho) = \frac{T_1(S,\rho)}{T_P(S,\rho)} \tag{22}$$

where $T_1(S,\rho)$ and $T_P(S,\rho)$ represents the job average response times respectively through a uniprocessor system and a multiprocessor system, for the same scheduling strategy and for the same utilization factor $\rho$.

The question naturally arises as to which centralized system we are in fact comparing our multiprocessor system. It is not hard to see that indeed we are comparing the multiprocessor system with P processors, to the centralized system composed of P individual noninteracting uniprocessor subsystems, where the average arrival rate of jobs to each is $\frac{\lambda}{P}$. This indeed constitutes an interesting comparison, for it ascertain the gain achieved by interconnecting the P individual processors. Moreover, the super uniprocessor (i.e., a uniprocessor system having the same capacity as the P-processor system) system is always superior to the multiprocessing system.

For our discriminatory processor sharing scheduling strategy, equation (22) becomes:

$$\sigma(DPS-WF,P,\rho) = \frac{T_1(DPS-WF,\rho)}{T_P(DPS-WF,\rho)} \tag{23}$$

Using our approximation, we then obtain the speedup function for our process graph of Figure 1. Figure 8 depicts the P-DPS-WF system speedup as a function of the utilization factor $\rho$, and for various values of the number of processors P. At $\rho = 0$, we obtain the customary definition of the speedup factor, that is $\sigma(DPS-WF,1,0) = 1$, $\sigma(DPS-WF,2,0) = 1.344$ and $\sigma(DPS-WF,P,0) = 1.3846$. for all $P \geq 3$. For $\rho = 1$, we observe that the speedup factor reaches the value P, as stated through Theorem 7. Therefore, we may conclude that the speedup factor for a P-DPS-WF system ranges between its lowest value obtained at $\rho = 0$ and which is equal to the value of the concurrency degree F used, to its highest value P obtained at $\rho = 1$. Moreover, all intermediary values of the speedup factor can be obtained by a proper choice of the value of the utilization factor. In Figure 8, we also depicted the speedup factor achieved by an infinite number of processors. This speedup factor forms then, for any value of $\rho$, the upper bound on the achievable parallelism.

Figure 9 shows the achievable parallelism (i.e., speedup factor) as a function of the number of processors for different values of $\rho$. We depict in heavy marks the two limiting cases; namely the case of $\rho = 0$ and the case of $\rho = 1$. When $\rho = 0$, the lowest curve shows that a substantial increase in the speedup factor is only obtained when we move from P=1 to P=2 and then to P=3. For higher values of P, the speedup factor is the same as that achieved by an infinite number of processors. For $\rho = 1$ on the other hand, we have the other heavy marked curve which states that the speedup factor is equal to the number P of processors used. As $\rho$

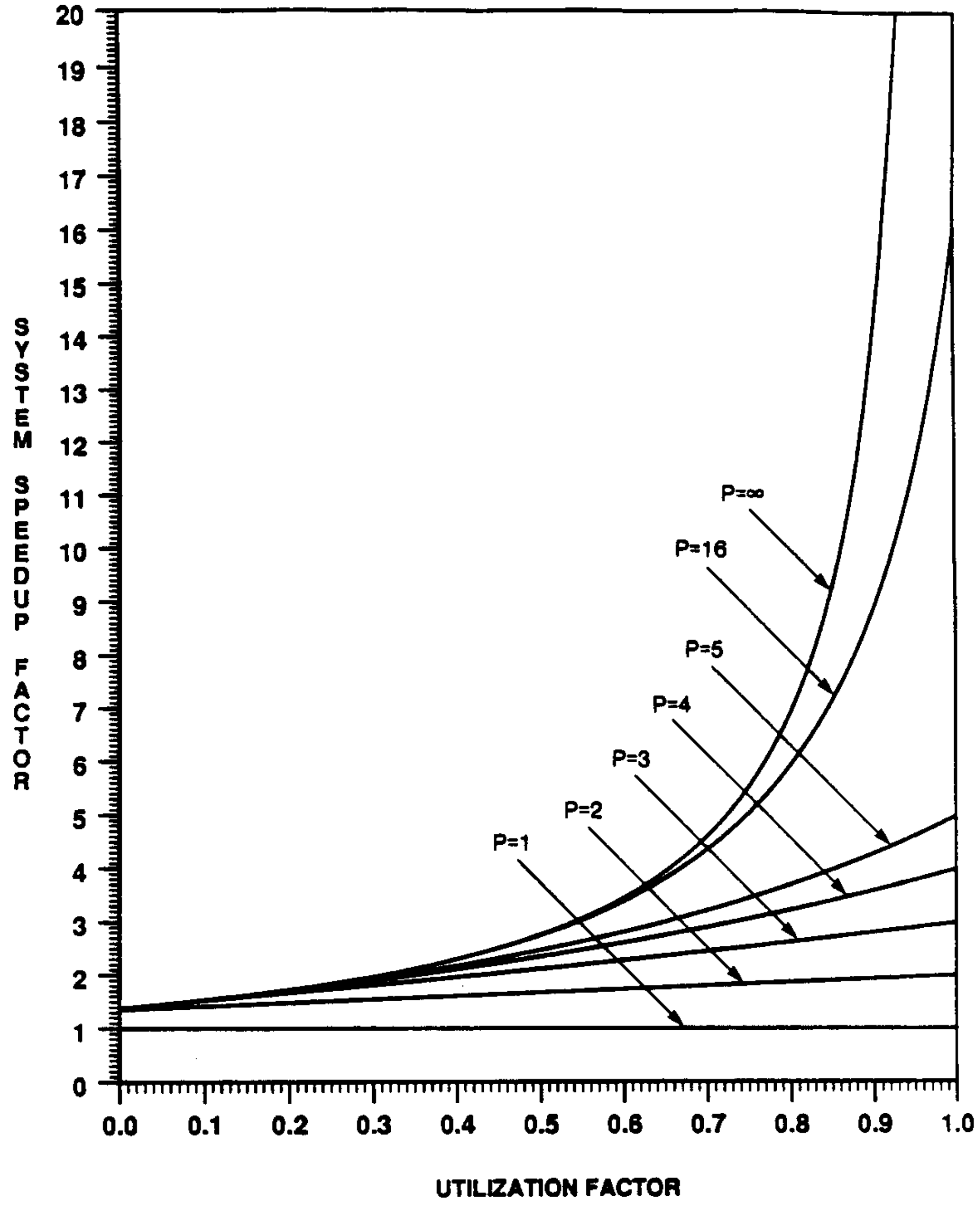


Figure 8: Achievable Parallelism in a P-DPS-WF Multiprocessor System

increases from zero to one, we obtain the other curves. In the next section, we shall determine the optimum operating value of $\rho$ and then deduce the achievable parallelism obtained at such a level.

The efficiency per processor in a multiprocessor system with P processors is the ratio $\frac{\sigma}{P}$. As mentioned earlier, it is also of interest to quantify the efficiency of each processor. Figure 10 depicts the efficiency per processor as a function of the utilization factor $\rho$, and for the process graph of Figure 1, and for P=1,2,3,4,5,16. As expected, we notice that for small values of $\rho$, the efficiency of each processor is very low, and poorer as P gets larger. As the system utilization factor grows towards one, the efficiency per processor grows rather rapidly and at $\rho = 1$ reaches its maximum value of one.

Recall from our definition of the speedup factor, that we are indeed comparing a P-individual-noninteracting uniprocessors system architecture to a multiprocessing system architecture. Figures 8 and 9 show how much gain can be achieved by using our P-DPS-WF system architecture compared to the P individual noninteracting 1-DPS-WF systems architecture. We observe from these figures that the parallel processing architecture is superior (achieves a lower job average response time) to the centralized architecture over all permissible values of $\rho$. Indeed, this superiority approaches its maximum when the utilization factor approaches one. Nevertheless, at $\rho = 1$, the job average response time in the P-DPS-WF system, while P times less than that in the centralized architecture, is too large to be of any use. The question naturally arises as to which value of the utilization factor should we use for the P-DPS-WF system, and consequently how much parallelism is achieved at this utilization factor point.

Our interest is the tradeoff between throughput and response time involved in choosing a particular system operating point. As the input traffic offered to our multiprocessing system increases, the job average response time increases; see Figure 7. On the other hand, since we are operating within the system stability condition; namely $\rho<1$, then the throughput of the system is equal to its input rate. Hence, the job average response time and the throughput of our multiprocessing system are both increasing functions of the input traffic. A performance measure incorporating throughput and delay into a single function is the notion of *Power* introduced in [Gies78]. It is simply defined as:

$$PW = \frac{\rho}{T_P(\rho)}$$

where $\rho$ is the system utilization factor and $T_P(\rho)$ is the job average response time through the multiprocessing system with P processors. The two contrasting objectives of maximizing throughput and minimizing delay are combined into this single objective function. Other measures of power have appeared in the literature [Yosh77, Klei79].

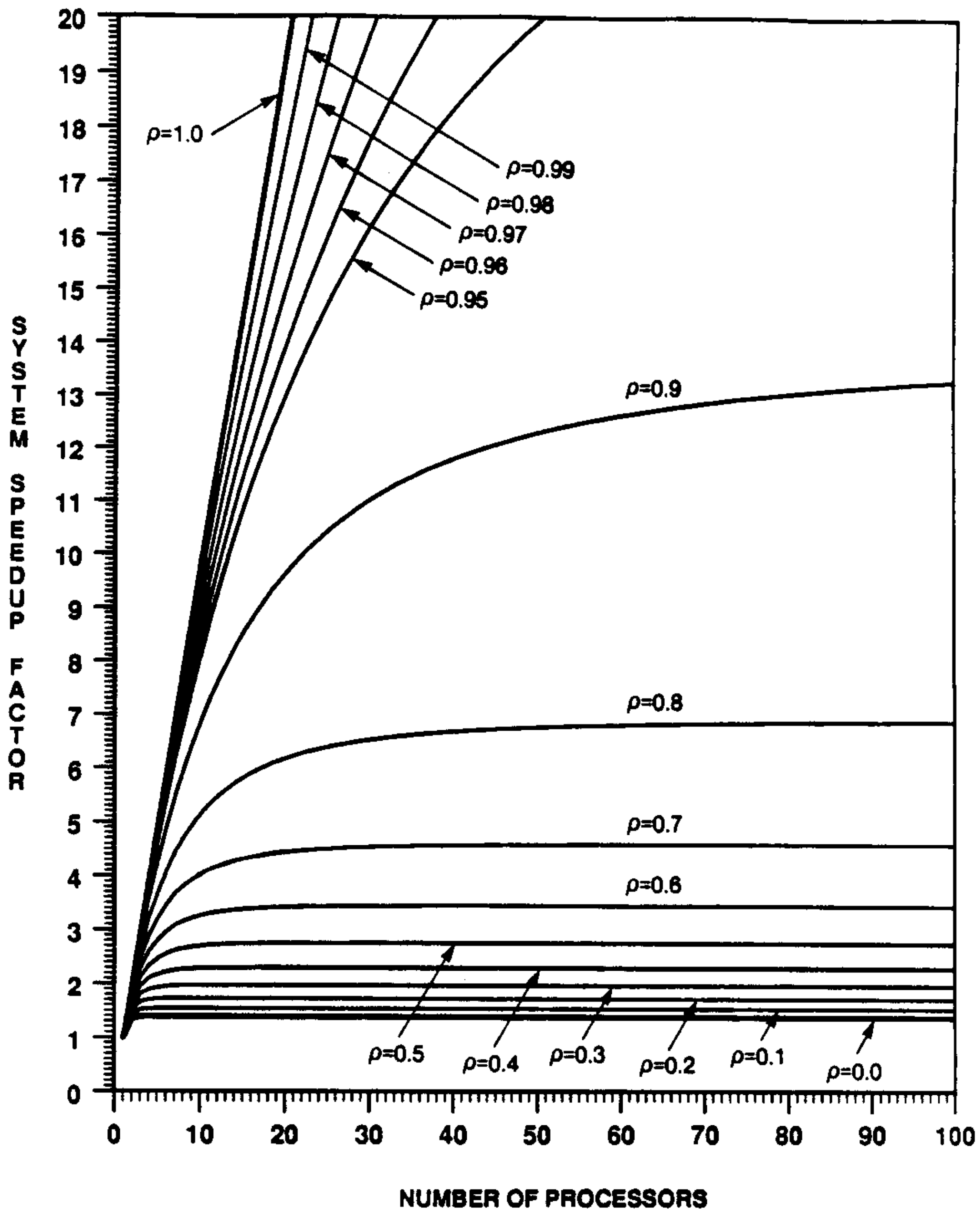


Figure 9: Achievable Parallelism versus the Number of Processors

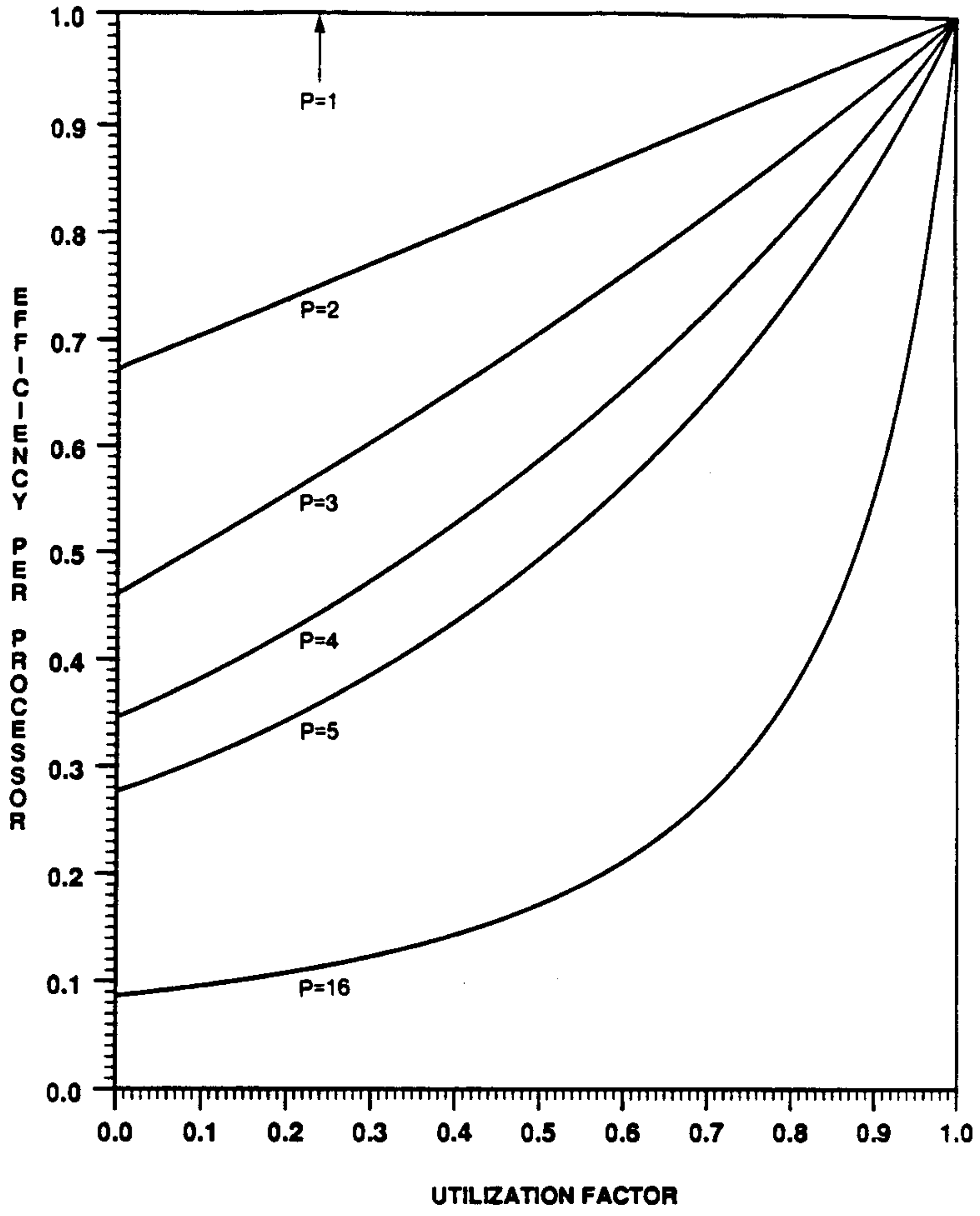


Figure 10: Efficiency per Processor in a P-DPS-WF Multiprocessor System

Note from Figure 7 that for small values of ρ, a significant increase in the traffic input (i.e., in the throughput) can be obtained with only a slight increase in the job average response time, motivating us to increase the input traffic in this region. Conversely, for large values of ρ, a large decrease in the job average response time will occur if the input traffic rate is decreased only slightly, motivating us to decrease the input traffic rate in this region. From Figure 7, it is not difficult to see that an appropriate operating point for our multiprocessing system would be in the vicinity of the *Knee* of the average response time versus the utilization factor curve. The knee is defined as the point on the curve such that a line through the origin to this point is tangent to the curve. Kleinrock [Klei78a] demonstrated the usefulness of this knee criterion by observing that the value of ρ which maximizes power occurs exactly at the knee. Kleinrock further extended the above argument by noting that the job average response time curve need not be a convex function of ρ. In the cases where more than one tangent line can be found (i.e., more than one knee occurs), maximum power will occur for that tangent line which makes the smallest angle with the horizental axis. Although it is known that the throughput-response time curves are convex, non convex curves may occur in the case of multiple access protocols which adapt to increasing load. Such a type of behavior was already observed for the URN scheme of Yemini and Kleinrock [Klei78b].

Using the process graph description of Figure 1, and our approximation of the job average response time through a P-DPS-WF system, we obtain the power profile depicted in Figure 11. In this figure, we plot the power of the multiprocessor system as a function of the utilization factor ρ for various values of P. Since for any permissible value of ρ, the minimum average response time is obtained by using an infinite number of processors, it follows that this case provides the upper bound of the power measure. This is then represented on Figure 11 by the straight line defined by $PW = \frac{\rho}{T_\infty} = \frac{\rho}{5.055555}$. It is rather interesting to notice that the optimal operating point grows with P. Figure 12, depicts the relationship between the number of processors P, and the optimal corresponding operating point. For a given value of the utilization factor ρ, Figure 12 gives the number of processors to be used in the P-DPS-WF multiprocessor system which behaves optimally at such utilization level. For a given value of P, on the other hand, Figure 12 provides the optimal operating point ρ. Notice that in the range $\rho \in [0,0.5)$ there exists no system that behaves optimally (i.e., at its maximum power), and that for high values of ρ (i.e., $\rho \geq 0.8$), a slight increase in ρ amounts to a large increase in P.

We now contrast our multiprocessing system to the centralized architecture system. Two alternatives may be considered. First, we compare the P-DPS-WF multiprocessor system to the centralized architecture system operating both at their respective optimal operating points. In this case, the achievable parallelism is represented by the lower curve in Figure 13. The short dashed line represents the asymptotic behavior of the achievable parallelism as P gets very large. Since the optimal operating point for the centralized architecture is at $\rho = 0.5$ which gives $T(1)=14.0$ for our process graph of Figure 1, and since the average response time through

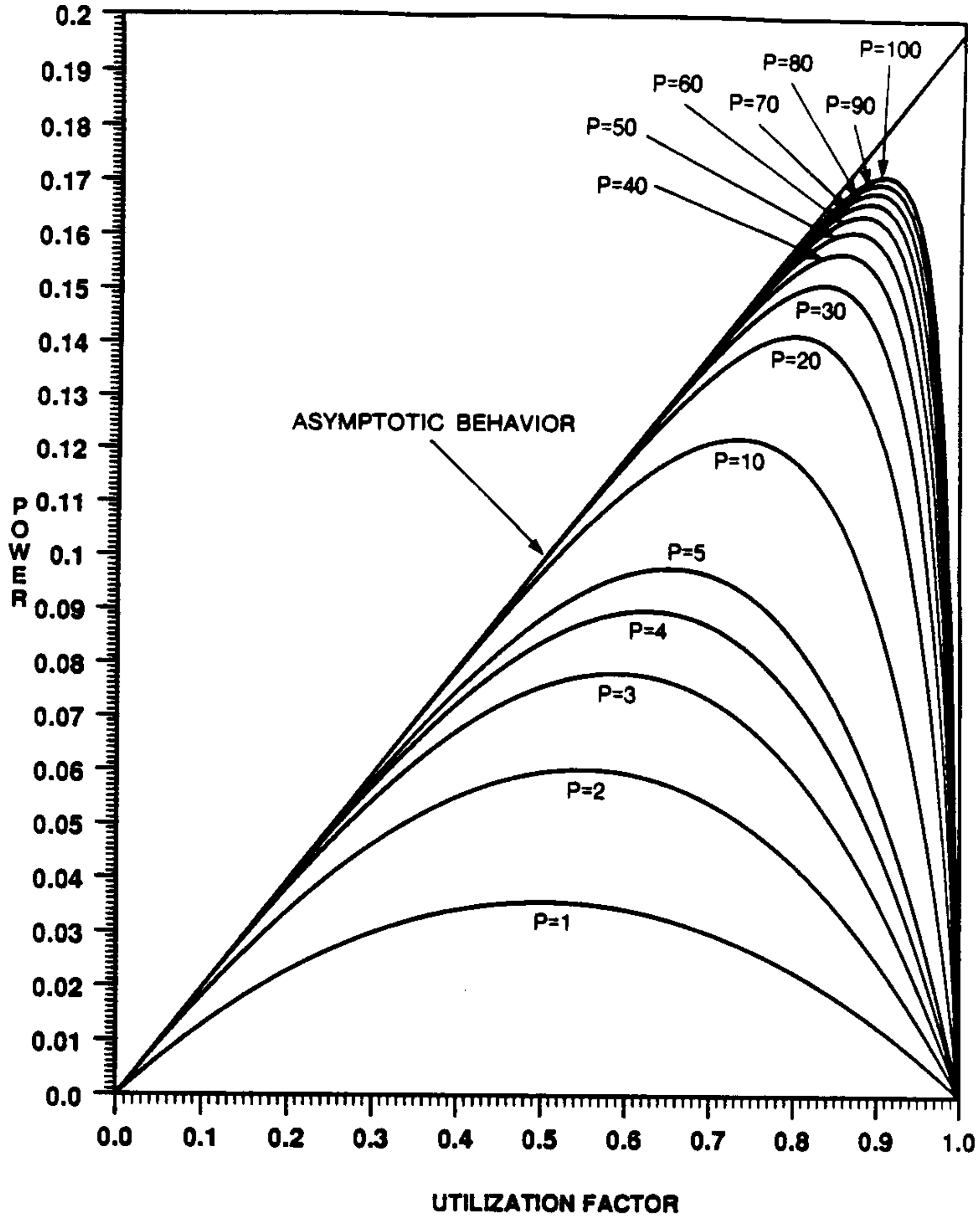


Figure 11: Power Function in a P-DPS-WF Multiprocessor System

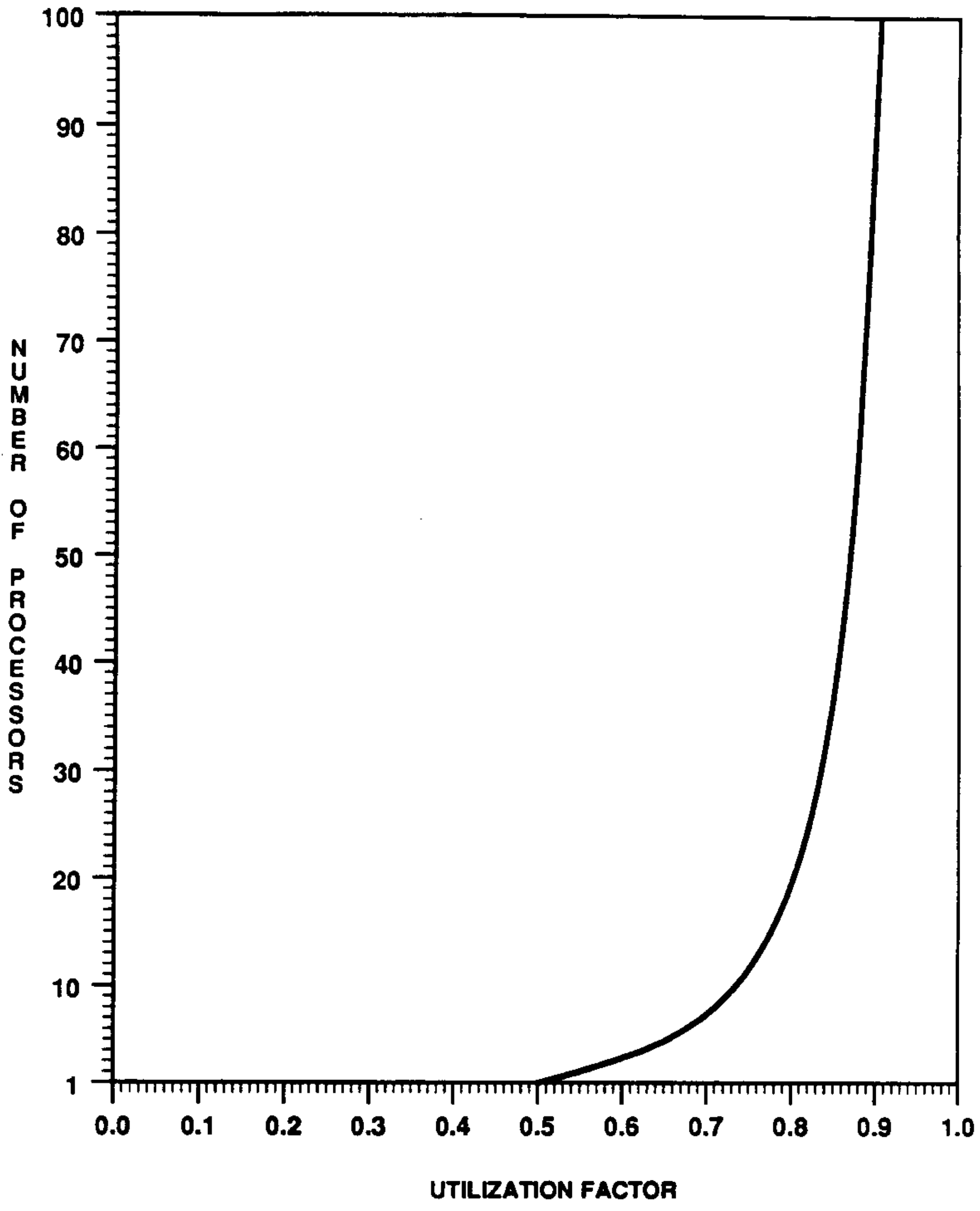


Figure 12: Maximum Power in a P-DPS-WF Multiprocessor System

an infinite number of processors system is 5.055555, it follows that the asymptotic value is equal to 2.76923. It is also interesting to notice that the achievable parallelism, as a function of P, reaches rather quickly its ultimate asymptotic value. This is mainly due to the fact that after a certain value of P, the P-DPS-WF system behaves as an infinite number of processors system. Second, we compare the P-DPS-WF multiprocessing system to the centralized architecture system operating both at the same value of the utilization factor; the one defined by the operating point of the multiprocessing system. For this case, we obtain the upper curve of Figure 13. Although the achievable parallelism of the P-DPS-WF system is less than P, it is monotonically increasing with the number of processors used. This later comparison is more interesting and practical, for it compares the two architecture under the same load conditions.

## 6 Conclusion

Significant reductions in the job average response time can be realized by executing a job, described by a given process graph, on a multiprocessor system. In this Report, we introduced a new scheduling strategy termed the *Discriminatory Processor Sharing With job Feedback*, which is proved to form a complete family of scheduling strategies in the uniprocessor case. The study of the job average response time assumed a preliminary conversion of the process graph into an execution graph describing the execution stages of a job during its life in the system. By exploiting the execution graph properties, we formulated and proved upper and lower bounds on the job average response time. Accurate and yet very simple approximation for the job average response time are developed and used to quantify the achievable parallelism attained by multiprocessor systems. To portray the level of parallelism achieved, we defined the speed up factor as a function of both the scheduling strategy and the utilization factor of the system.

In this Report, we have assumed that the cost of exchanging data between processors is free and is achieved instantaneously. In reality, there is always some delay incurred when communicating between processors. The communication delays are rather crucial components which must be considered in conjunction with the processing delays to better ascertain the parallelism achieved by multiprocessor systems, and to properly compare multiprocessor systems to multicomputer systems. A simple model to account for such communication delays would incorporate, in the job process graph, some communication nodes representing both the communication times involved between parallel tasks, and the data transfers required between consecutive tasks. Investigations of this communication issue are being considered.

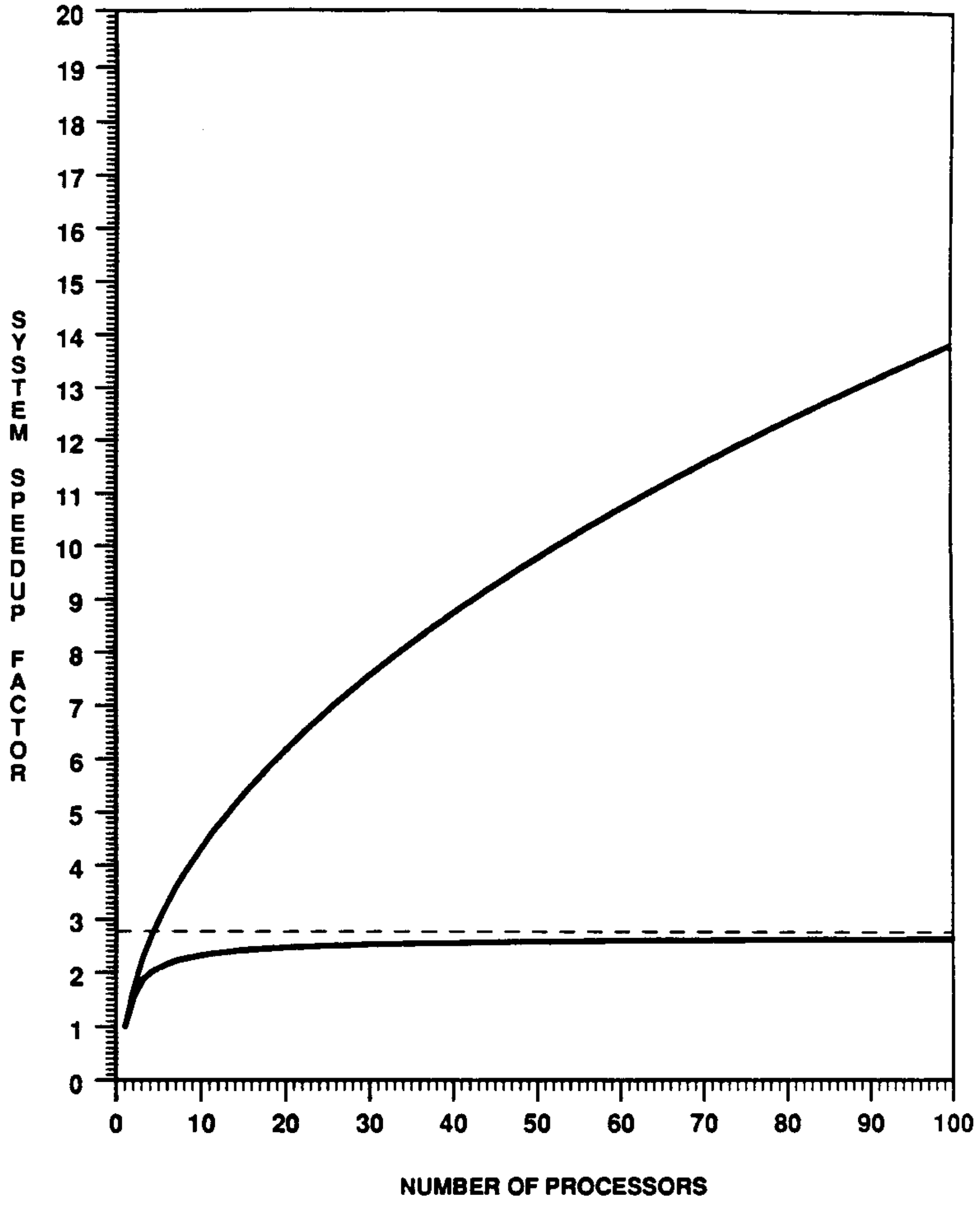


Figure 13: Achievable Parallelism at Optimal Operating Points

We have seen that a job may need and consequently may hold more than one processor at a time. Consider the process graph of Figure 1. Let us assume that no more than one processor may work on a given task. When task A is being processed, the job can proceed at a maximum rate of 1 second per second. When task A is completed, tasks B and C may begin, and the job can proceed at a maximum rate of 2 seconds per second, assuming a multiprocessor system comprising two or more processors, etc. Consequently, the maximum rate at which a job can absorb work depends upon where it is in its processing cycle (i.e., which tasks are ready), and on the number of processors composing the multiprocessor system. This forms a new class of queueing systems where the maximum rate at which a job can proceed varies with elapsed time, as compared to the classic queueing model, which assumes that the maximum rate at which a job can absorb work is constant. In this Report, we adopted the discriminatory processor sharing of the processors' total capacity among the jobs present in the system, and thus we solved (exactly in the infinite number of processors case, and approximately in the finite processors case) for the job average response time in such a system. Further research is also undoubtedly needed in this direction.

## References

[Bask75] F. Baskett, K.M. Chandy, R.R. Muntz, and F. Palacios-Gomez, ''Open, Closed, and Mixed Networks of Queues with Different Classes of Customers,'' *JACM*, Vol. 22, No. 2, April 1975.

[Belg85] A. Belghith and L. Kleinrock, ''Analysis of the Number of Occupied Processors in a Multiprocessing system,'' *Technical Report No. UCLA-CSD-850027, School of Engineering and Applied Science, Computer Science department, University of California, Los Angeles*, August, 1985.

[Belg86] A. Belghith, ''Response Time and Parallelism in Multiprocessing Systems with Certain Synchronization Constraints,'' *Ph.D. Dissertation, School of Engineering and Applied Science, Computer Science Department, University of California, Los Angeles*, December 1986.

[Chan77] K.M. Chandy, J.H. Howard, and D.F. Towsley, ''Product Form and Local Balance in Queueing Networks,'' *JACM* , Vol. 24, No. 2, April 1977, pp. 250-263.

[Fayo78] G. Fayolle, I. Iasnogorodski, and I Mitrani, ''On the Sharing of a Processor Among Many Job Classes,'' *IRIA-Laboria, Domaine de Voluceau, 78150 Le Chesnay, France*, 1978.

[Gail83] R. Gail, ''On the Optimization of Computer Network Power,'' *Ph.D. Dissertation, School of Engineering and Applied Science, Computer Science Department, University of California, Los Angeles*, September 1983.

[Gies78] A. Giessler, J. Hanle, A. Konig, and E. Pade, ''Free Buffer Allocation - An Investigation by Simulation,'' *Computer Networks*, Vol. 1 (3), July 1978, pp. 191-204.

[Hwan84] K. Hwang and F.A. Briggs, *Computer Architecture and Parallel Processing*: McGraw Hill ( Series in Computer Organization and Architecture), 1984.

[Klei67] L. Kleinrock, "Time-Shared Systems : A Theoretical Treatment," *Journal of the Association of Computer Machinery (JACM),* Vol. 14, 1967, pp. 242-261.

[Klei75] L. Kleinrock, *Queueing Systems, Vol 1: Theory,* New York: Wiley-Interscience, 1975.

[Klei76] L. Kleinrock, *Queueing Systems, Vol 2: Computer Applications,* New York: Wiley-Interscience, 1976.

[Klei78a] L. Kleinrock, "On Flow Control in Computer Networks," *Conference Records, International Conference on Communications,* Vol. 2, June 1978, pp. 27.2.1-27.2.5.

[Klei78b] L. Kleinrock and Y. Yemini, "On Optimal Adaptive Scheme for Multiple Access Broadcast Communication," *Conference Records, International Conference on Communications,* June 1978, pp. 7.2.1-7.2.5.

[Klei79] L. Kleinrock, "Power and Deterministic Rules of Thumb for Probabilistic Problems in Computer Communications," *Conference Record, International Conference on Communications,* June 1979, pp. 43.1.1-43.1.10.

[Kung84] K.C. Kung, "Concurrency in Parallel Processing Systems," *Ph.D. Dissertation, School of Engineering and Applied Science, Computer Science Department, University of California, Los Angeles,* 1984.

[Lave83] S.S. Lavenberg, *Computer Performance Modeling Handbook,* New York: Academic Press, 1983.

[Litt61] J. Little, "A Proof of the Queueing Formula $L = \lambda W$," *Operations Research,* Vol. 9, No. 2, March 1961, pp. 383-387.

[Munt73] R.R. Muntz, "Poisson Departure Processes and Queueing Networks," *Proceedings of the 7th Annual Princeton Conference on Information Science, Princeton University,* 1973, pp. 435-440.

[O'Do74] T.M. O'Donovan, "Direct Solutions of M/G/1 Processor-Sharing Models," *Operations Research,* Vol. 22, 1974, pp. 570-575.

[Yosh77] Y. Yoshioka, T. Nakamura, and R. Sato, "An Optimum Solution of the Queueing System," *Electronics and Communications in Japan,* Vol. 60 (B8), August 1977, pp. 590-591.