EXPERIMENTAL DATA ON HOW PROGRAM BEHAVIOR AFFECTS THE CHOICE OF SCHEDULER PARAMETERS

Juan Rodriguez-Rosell*
Department of Information Processing
The Royal Institute of Technology
Stockholm, Sweden

Summary

A theory of combined scheduling of processor and main memory has begun to emerge in the last few years. It has been proposed that schedulers use the working set concept to avoid thrashing. To our knowledge, no data has been published on experimental measurements of working sets or on the influence that program behavior may have on the choice of quantum size. In this paper empirical data are presented and its influence on the choice of system parameters is discussed.

Introduction

A computer system has different types of resources of which processor and memory are the two most important ones. In paging systems a process can execute having only a sub set of pages in memory. The dynamic address translation mechanism together with the operating system insures that when a page not present in memory is referenced the process will be prevented from further execution until the missing page is brought to core.

Scheduling means deciding which processes are to use the system's resources. In this case the resources that interest us are processor and main memory. Recent research has developed the working set model for program behavior and a theory of combined scheduling of processor and memory has been built around it[1]. Two recent papers deal with the implementation of this scheduling method[2,3], but to our knowledge no information has been published on measured program working set behavior. It is the purpose of this paper to report on working set characteristics of programs and how they might influence the choice of scheduling parameters.

Program working set behavior

The working set, $W(t,\tau)$, of a process is the collection of information referenced by the process in the time span $(t-\tau,t)$. The domain of definition t and τ is time, expressed in some adequate unit, and since we are interested in paging machines we will take the domain of definition of W to be the set of virtual pages allowed to be referenced. Another variable, the working set size, $\omega(t,\tau)$, is simply the number of pages in the working set. Its expected value, $E_t(\omega(t,\tau))$, is denoted by $\bar{\omega}(\tau)$. We assume that it is independent of time. As the process executes, it will reference pages not present in the working set. We call $\lambda(t,\tau)$ the number of pages reentering the working set per unit time. Its expected value can be written as $\bar{\lambda}(\tau)$, a function of τ alone.

In a scheduler based on working sets a process cycles through a running set each time receiving a burst of τ time units until it either blocks or exhausts a quantum q. The pertinent questions are how to choose τ and q. It is necessary to estimate how many pages are expected to be required by a process and how many page faults it will generate. By hypothesis the dispatcher ensures that every member of the running set keeps its working set in memory. The working set is computed at the end of each burst. It is clear that with such a policy m, the expected number of page faults per unit time, is at worst equal to $\bar{\lambda}(\tau)$. In[3] it is reported that the quantum has been chosen arbitrarily whereas[2] does not discuss this problem at all.

Process Efficiency

A process efficiency is defined as the percentage of virtual time a process spends computing

Restricting our attention to quantums of the form

$$q(\tau) = K\tau$$

we expect under demand paging $\bar{\omega}(\tau)$ page faults in the first burst, since the working set has to be demand paged into memory. Thereafter we expect

$$\tau \ \bar{\lambda}(\tau)$$

page faults during each of the remaining $K-1$ bursts. The process efficiency is then

$$\eta_p = \frac{K\tau}{K\tau + T_p(\bar{\omega}(\tau) + (K-1)\tau\bar{\lambda}(\tau))} = \frac{1}{1+f(\tau)}$$

$$f(\tau) = \frac{T_p}{K}\left(\frac{\bar{\omega}(\tau)}{\tau} + (K-1)\bar{\lambda}(\tau)\right)$$

where T_p is the traverse time involved in transferring a page between main memory and auxiliary storage. To maximize η_p is tantamount to minimizing $f(\tau)$.

*Visitor at IBM France Scientific Center
Boulevard de la Chantourne, 38 - La Tronche
France

Deriving we obtain

$$\frac{df(\tau)}{d\tau} = 0 \Longrightarrow K = 1 + \frac{\bar{\omega}(\tau)-\tau\bar{\lambda}(\tau)}{\tau^2 d\bar{\lambda}(\tau)/dt}$$

We have used the fact that $\bar{\lambda}(\tau) = \frac{d\bar{\omega}(\tau)}{d\tau}$. For the proof see [1].

By hypothesis K>1 which means

$$\bar{\omega}(\tau) - \tau\,\bar{\lambda}\,(\tau) < 0$$

because $\frac{d\bar{\lambda}(\tau)}{d\tau} < 0$.

The shape of $\bar{\lambda}(\tau)$ strongly suggests that in a certain range it may be approximated by

$$a\,e^{-b\tau}$$

solving for $\bar{\omega}(\tau)$ in $\frac{d\bar{\omega}(\tau)}{d\tau} = \bar{\lambda}(\tau)$ we find

$$\bar{\omega} = \bar{\omega}_0 + \frac{1}{b}(\bar{\lambda}_0-\bar{\lambda}).$$

We write $\bar{\omega}$ for $\bar{\omega}(\tau)$ and $\bar{\omega}_0$ for $\bar{\omega}(\tau_0)$ for simplicity of notation. The same applies to $\bar{\lambda}(\tau)$ and

$$\bar{\omega}_0 + \frac{\bar{\lambda}_0}{b} < \bar{\lambda}\,(\frac{1}{b} + \tau) = a\,e^{-b\tau}(\frac{1}{b} + \tau) \Longrightarrow$$

$$\frac{\bar{\lambda}_0+b\bar{\omega}_0}{a} < (1 + b\tau)e^{-b\tau} < 1.$$

The experimental values for $\bar{\lambda}_0$, $\bar{\omega}_0$, a, b indicate that

$$\frac{\bar{\lambda}_0+b\bar{\omega}_0}{a} > 1.$$

It seems than in the range where the approxi mation is valid no maximum can be attained. It is however possible to obtain an upper bound on the efficiency, because from

$$f(\tau) = \frac{1}{\eta_p} - 1$$

we can deduce

$$K = \frac{\bar{\omega}-\tau\bar{\lambda}}{\tau(\alpha-\bar{\lambda})}$$

$$\alpha = \frac{1}{T_p}\,(\frac{1}{\eta_p} - 1)$$

since $\bar{\omega} - \tau\bar{\lambda} > 0$

we find

$$\alpha > \bar{\lambda} \Longrightarrow$$

$$\eta_p < \frac{1}{1+\bar{\lambda}T_p}$$

The quantity on the right-hand side of the inequality can be considered to be the process steady-state efficiency (called duty factor in [1]). This quantity is a strict upper bound on the attainable process efficiency. We can come arbitrarily close to it, however. Since λ is a decreasing function of τ the relative maximum for the process steady-state efficiency corresponds to choosing the largest possible τ. Set now

$$\eta_p = \frac{\rho}{1+\bar{\lambda}T_p} \qquad \rho < 1$$

as target efficiency. This gives

$$K = \frac{T_p(\bar{\omega}-\tau\bar{\lambda})}{\tau(1+\bar{\lambda}T_p)} \qquad \frac{\rho}{1-\rho}$$

As ρ comes closer to 1 the process efficiency approaches the steady state efficiency and K must be increased accordingly.

System Efficiency

The question arises of whether we should try to maximize the total system efficiency (defined as the percentage of real time the CPU is busy with problem programs) rather than a process total efficiency.

When a program executes in a paging machine it will demand I/O operations from the paging channel and the file units. If the program is to execute V virtual time units it will require a total time of

$$T = V + Vm_p\,T_p + Vm_{i0}\,T_{i0}$$

The process efficiency will in this case be

$$\eta_T = \frac{V}{T} = \frac{1}{1+m_pT_p+m_{i0}T_{i0}}$$

The quantities m_p and T_p are respectively the number of page faults per unit virtual time and the transfer time of a page fault. A complete discussion of these quantities is found in [1]. Similarly, m_{i0} and T_{i0} are the number of I/O operations per unit virtual time demanded by the process and the time necessary to process one such request. The discussion of the factors affecting these two parameters parallels that of m_p and T_p. It is interesting, however, to notice that m_{i0} depends only on the process being considered.

As long as the quantities m_p, T_p, m_{i0}, T_{i0} are independant of the number N of processes in main memory the total system efficiency will grow linearly with N, i.e.,

$$\eta_{sys} = \frac{N}{1+m_pT_p+m_{i0}T_{i0}}$$

If this is the case the times T_p and T_{i0} are composed almost exclusively of seek time and/ or rotational delay, plus data transfer time. However, when N is increased both thrashing (i.e. an increase in m_p) and the effect of queueing (i.e. an increase in T_p) will add up to make $m_pT_p >> 1 + m_{i0}T_{i0}$. This is confirmed experimentally, since in artificially induced thrashing it is observed that the number of I/O operations per second of real time is very low (as corresponds to an extremely low observed CPU activity) whereas the channel is paging at nearly its maximum rate.

If there are M pages available in main memory N is given by

$$N = \frac{M}{\bar{\omega}+m_p\tau}$$

where m_p is to be replaced by the average number of page faults per unit time, i.e.

$$m_p = \frac{\bar{\omega}+(K-1)\tau\bar{\lambda}}{K\tau}$$

It is possible to form now $\frac{\partial\eta_{sys}}{\partial\tau}$ and $\frac{\partial\eta_{sys}}{\partial K}$, set them to 0 and solve the two non-linear, simultaneous equations. The expressions involved are complicated and an easier course is to plot η_{sys} as a function of τ for different values of the parameter K. It shows sudden decreases in efficiency whenever the expected multiprogramming level diminishes.

System Operating Point

Measures of processor and memory utiliza tion are, respectively, system efficiency, η_{sys} (defined above) and memory efficiency, η_{mem}, defined as the percentage of memory occupied by the processes executing in the system. The operating point of the system is a point in the (η_{mem},η_{sys}) plane.

The path of the operating point of the system depends on several factors. The processes executing and the behavior of the operating system affect it. It oscillates rapidly with time but experimental evidence shows that the average operating point of a whole session will vary little from that of other sessions.

Computer systems often operate near memory saturation. In other words, η_{mem} is often about 90 %. In a well balanced system η_{sys} will be correspondly high. However, in a surprinsingly large number of cases η_{sys} is very low, about 20 %. In an effort to use the idle CPU unwise operating systems try to increase the level of multiprogramming. Thus thrashing sets in.

With a working set policy the number N of processes in main memory is not fixed. Rather it varies depending on the instantaneous demands of the processes being executed. We would like to know the probability of finding the system in a certain operating point.

Let the function $G(\theta,\tau)$ be defined as follows

$$G(\theta,\tau) = \text{probability } (\omega(t,\tau) \leq \theta)$$

The function $G(\theta,\tau)$ is called the page demand function. It depends only on the process being considered and not on the choice of paging algorithm. The function accounts for the fact that working set sizes are rather different from their expected values. The derivative $\frac{dG}{d\theta}(\theta,\tau)$ indicates the most probable page demand value.

Under a working set dispatching policy a process is guaranteed an amount of memory equal to its working set size. Moreover, it will demand the pages reentering its working set. A process has the right to claim

$$r(\tau) = \bar{\omega}(\tau) + \tau\bar{\lambda}(\tau)$$

pages. When a process demands more than $r(\tau)$ pages it is said to be page avid. The probability of a process being avid is $1 - G(r(\tau),\tau)$ If we have identical processes the expected multiprogramming level is

$$N = \text{entier } (M/r(\tau))$$

Corresponding to this expected multiprogramming level we will have an expected system efficiency given by

$$\bar{\eta}_{sys} = \frac{N}{1+m_pT_p+m_{i0}T_{i0}}$$

and an expected memory efficiency, $\bar{\eta}_{mem}$,

$$\bar{\eta}_{mem} = \frac{Nr(\tau)}{M}$$

The probability that the system efficiency decreases below η_{sys} depends on no program being page avid, for when a program is page avid it may well demand so many pages that another program is forced to leave the running set in order to accommodate the page avid program's sudden demand for pages. The probability of no process being page avid is of course

$$1 - (G(r(\tau), \tau))^N$$

and the probability that some process is page avid is an upper bound on the probability that the system efficiency is smaller than its expected value.

$$P_r(\eta_{sys} < \bar{\eta}_{sys}) \leq 1 - (G(r(\tau),\tau))^N$$

Analysis of results

In order to investigate the dynamic behavior of programs a fully interpretive simulator designed to monitor any IBM SYSTEM/360 program has been used. A set of routines gathers information on which pages a program references at every instruction. The pages are then annotated in a boolean matrix kept in core. When the programs request an I/O operation the pages it references are also noted down by a program that examines the channel program. All data reduction is done after the monitor relinquishes control so that there will be no interference with the monitored programs I/O functions. The quantities measured include W(t,1) and the number of I/O operations the program requests together with the units involved.

To conduct the experiments the programs have been loaded in a virtual machine running in a 360/67 under control of CP67[4]. The system at Grenoble includes 128 pages, each page having 4096 8-bit bytes. Paging is done on a single drum, which CP uses on a first-come-first served basis. Each page is assigned a fixed location in the drum (or in disk should the number of pages in the drum be insufficient) which is never changed during the life of the process. After the data is collected other programs treat them and display pertinent results on a CRT.

We have tested the Fortran compiler the Assembler and the PL/I compiler, which are all heavily used components of the system Using virtual machines has given us the advantages of a bare machine while still keeping on-line capabilities. This has provided us with a powerful and versatile tool hard to match conventional systems.

Figures 1, 2 and 3 show pictures of $\bar{\omega}$, $\bar{\lambda}$ and $\omega(t,\tau)$ for τ = 25000 instructions. The program monitored was in this case the assembler. The program to be assembled was about 300 instructions long and it was assembled in roughly two million instructions. The range

(500 000 - 2 500 000) instructions

has been found enough to compile or assemble typical assembler or Fortran modules.

In figure 3 it can be seen that the working set size stays close to 20 pages during the life of the process. The working set size jumps suddenly, attaining values close to 40 pages and even depassing this value. In this case it has been determined that the peaks correspond to the loading of the different phases of the assembler. The peaks are short lived, rarely lasting more than 10000 instructions, as it could be seen in a plot of $\omega(t,\tau)$ with τ fixed at 10000 instructions. Software has already been modified in Grenoble to decrease the working set size of programs. Virtual access methods are presently being implemented. It is expected that this technique will completely eliminate the peaks.

Figures 4 and 5 show graphs of $\bar{\lambda}$ and $\bar{\omega}$ as a function of τ together with their approximations. The agreement with the experimental values in the range (5000,50000) instructions is good. The approximation used in this particular case has been

$$\bar{\lambda} = \frac{0.45}{10^3 \text{ instructions}} e^{-\frac{0.4x\tau}{10^4 \text{instructions}}}$$

Other experiments indicate that $\bar{\lambda}$ is approximated by

$$\bar{\lambda} = a_1\ a_2\ e^{-b\tau}$$

where a_2 and b are practically the same for all programs examined. Their values have been determined as

$$a_2 = 0.9$$

$$b = \frac{0.4}{10^4} \cdot \frac{1}{\text{instructions}}$$

The constant a_1 depends on the type of program being considered. For example, it has been found that

$$a_1 = \frac{0.5}{10^3} \cdot \frac{1}{\text{instructions}} \quad \text{for assembler}$$

$$a_1 = \frac{0.35}{10^3} \cdot \frac{1}{\text{instructions}} \quad \text{for Fortran}$$

Figure 6 shows $\bar{\eta}_{sys}$ as a function of τ. The curves for K=1 and K=20 are nearly parallel after τ = 20000 instructions. Before this value of τ the curves exhibit large variations caused by changes in multiprogramming level. For values of K between 10 and 50 the curves remain very close to the curve for K=20. This suggests that after τ = 20000 instructions and K=10 the expected systems efficiency should be about 45 per cent a value rather insensitive to the choice of K. On the other hand, the response time to a particular transaction depends quite certainly on the choice of K, which can then be viewed as the parameter insuring equity in the distribution of CPU time.

When τ is small the probability of some program being page avid is rather high, since we have 4 or 5 processes in memory. As τ increases, however, we reach a more stable operating point in which increasing τ does not significantly alter the expected efficiency, but it increases the probability of dropping further in the multiprogramming level. For this reason τ should be chosen such that it keeps the system in the stable region. Its choice depends on the available system resources. In the case depicted in figure 6, τ should be at least 20000 instructions (i.e. 30 msec in a 360/67). Then K can be chosen to improve efficiency somewhat. Its value should be greater than 10.

We are at present implementing a dispatcher based on working sets to run CP67. We have chosen τ = 30 msec and K=35. There is of course, no reason why K and τ cannot be time-dependent, nor any reason why they should be the same for all processes in the system. Our research is directed towards letting K and τ be time-varying, process-dependent parameters, as well as development of software measuring tools to evaluate system response.

Figure 7 shows the expected operating points of the system which depend on K and τ Only the stable region ($20000 \leq \tau \leq 50000$) is depicted. The path of the system as τ is increased is shown by the arrows.

Figure 8 is a plot of $G(\theta,\tau)$ and $\frac{dG}{d\theta}(\theta,\tau)$. It is interesting to notice that the most probable value is smaller than the average working set size. However, should the program become page avid, it will demand vastly more than its right, as indicated by the hump around 38 pages. Other programs show similar graphs. $G(\omega,\tau)$ is usually between 60 % and 70 %.

Figure 9 shows $G(r,\tau)$ and $1 - (G(r,\tau))^N$. G depends only on the program being considered but G^N depends on the multiprogramming level, i.e. on the available resources. In our case there are 85 pages available for user processes, giving very low multiprogramming levels. In the range of interest, multiprogramming is never done on more than 5 processes, 3 being the most common value. Experimental data being gathered by a Spy machine indicates that the mean multiprogramming level in a whole session is about 3 in our system.

Acknowledgements

I am deeply grateful to Mr. Max Peltier and Mr. Claude Hans, both of IBM France Scientific Center, for their guidance and criticism. Also M. Jacolin, S. Schuman and J.P. Dupuy, all members of the Scientific Center, have given their time and advice unsparingly. Thanks are due to the anonymous referees, who provided helpful comments.

This work has been supported by a research grant from IBM Svenska A.B.

Bibliography

1) Denning, P J : Resource allocation in multiprocess computer systems.
Tech. Rep. MAC-TR-50, MIT project MAC, Cambridge, Mass 1968

2) Weizer, N and Oppenheimer G : Virtual memory management in a paging environment.
Proc. AFIPS 1969, SJCC.

3) Doherty, W J : Scheduling TSS/360 for responsiveness.
Proc. AFIPS 1970, FJCC.

4) CP Program Logic Manual Form GY20-0590-0
IBM Corporation. Technical Publications Department

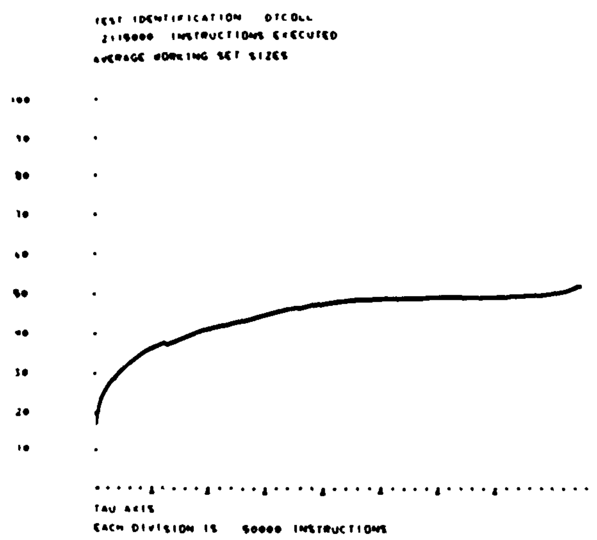

Figure 1

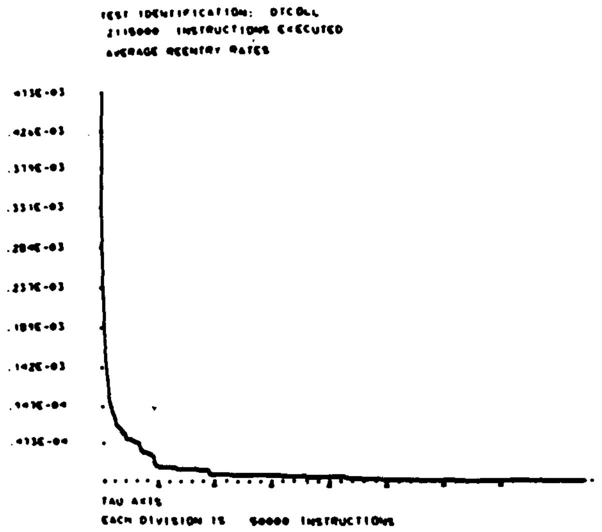

Figure 2

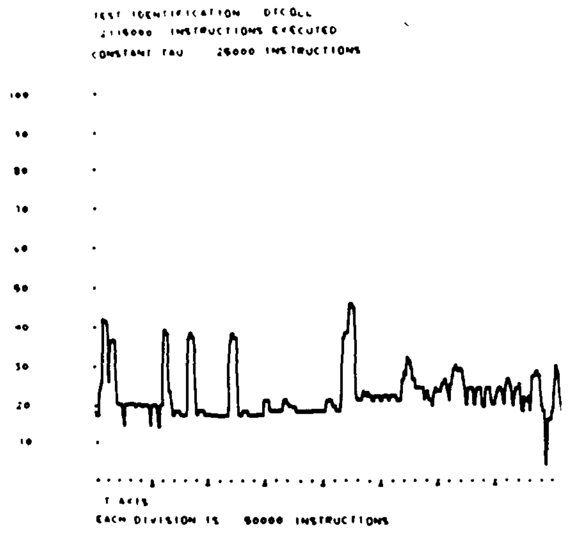

Figure 3

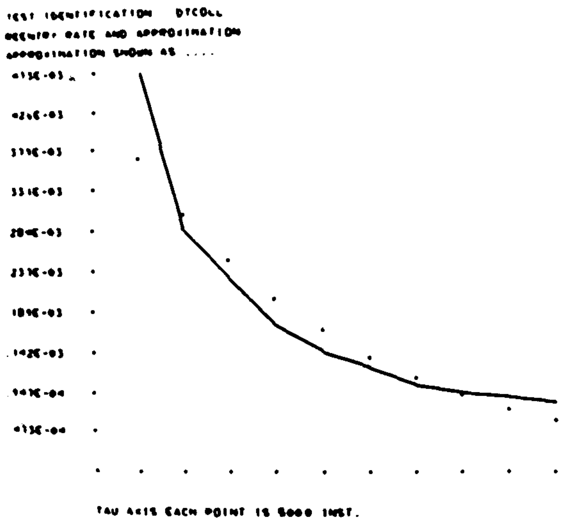

Figure 4

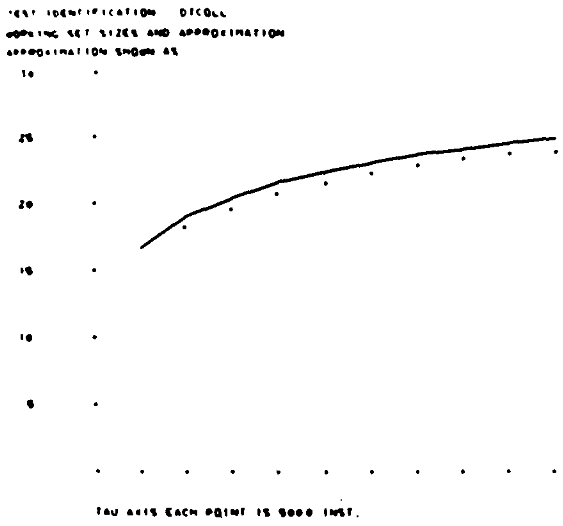

Figure 5

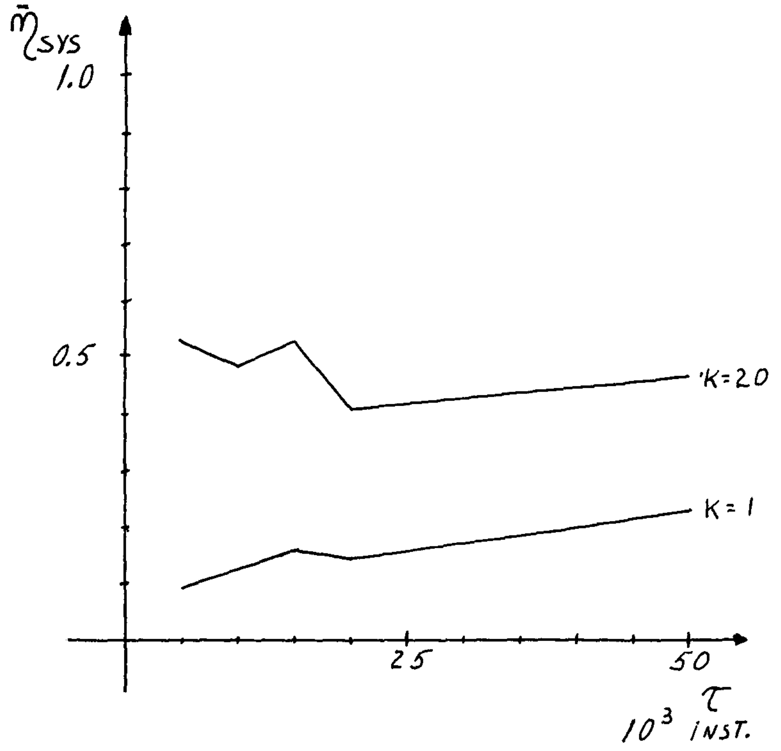

Figure 6

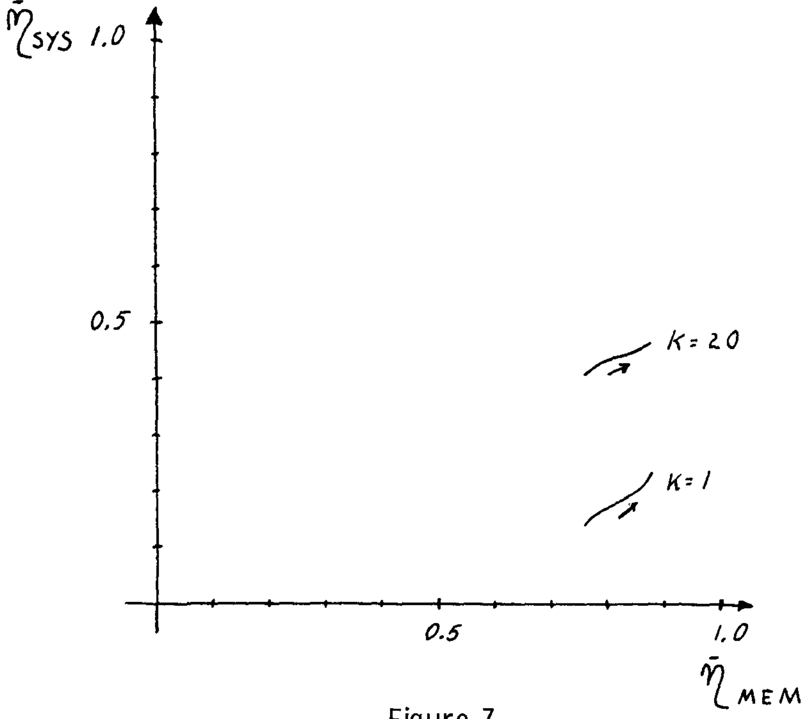

Figure 7

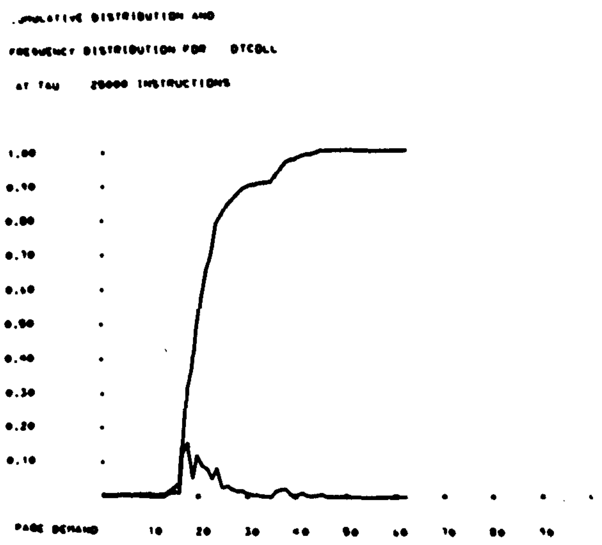

Figure 8

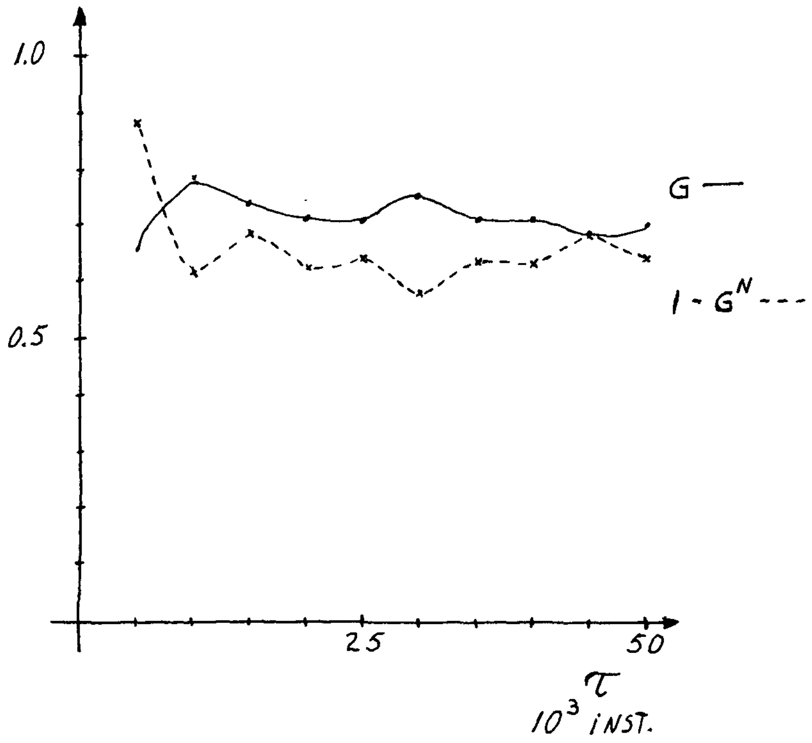

Figure 9