I.C.S.'87

INTERNATIONAL CONFERENCE ON SUPERCOMPUTING

Supercomputer architectures, technology, software (operating systems, compilers), theory of parallel processing, parallel numerical analysis, applications of supercomputers to science and engineering

JUNE 8-12, 1987 ATHENS, GREECE

ORGANIZED AND SPONSORED BY THE COMPUTER TECHNOLOGY INSTITUTE (C.T.I.) PATRAS, GREECE

Co-sponsored by: • Ministry of Industry, Energy and Technology of Greece • Ministry of Culture and Sciences of Greece • ACM
In cooperation with • I.F.I.P. WORKING GROUP 2.5 • SIAM Special Group on Supercomputing
• Purdue Center for Vector and Parallel processing • Center for Supercomputing Research and Development, University of Illinois • EATCS

PROGRAM COMMITTEE

D.J. Kuck, *USA* (Chairman)
T.S. Papatheodorou, *Greece* (Co-chairman)
J. Dongarra, *USA*
I. Duff, *England*
E.N. Houstis, *USA - Greece*
P. Gaffney, *Norway*
J. Lenfant, *France*
D. Maritsas, *Greece*
G. Paul, *USA*
C.D. Polychronopoulos, *USA*
J.R. Rice, *USA*
P. Spirakis, *USA - Greece*

INVITED SPEAKERS

Tilak Agerwala, *IBM*
Arvind, *MIT*
Steve Chen, *Cray*
Doug De Groot, *USA*
Jack Dongarra, *Argonne*
Ian Duff, *Harwell*
Geoffrey Fox, *Caltech*
Dennis Gannon, *Indiana*
Ron Gruner, *Alliant*
Ken Kennedy, *Rice*
David Kuck, *Illinois*
Jacques Lenfant, *Rennes*
Yoichi Muraoka, *Waseda*
Christos Papadimitriou, *Stanford*
John Rice, *Purdue*
Ulrich Trottenberg, *Suprenum*
Kenneth Wilson, *Cornell*

Proceedings Springer Verlag

ON THE PROCESSING TIME OF A PARALLEL LINEAR SYSTEM SOLVER

Andreas STAFYLOPATIS

and

Athanasios DRIGAS

National Technical University of Athens

Department of Electrical Engineering

Computer Science Division

157 73 Zographos, Athens, Greece

Abstract

The speed-up obtained by the use of multiprocessor systems is of major importance for numerical applications involving the solution of large dense systems of linear equations. We are interested here in the performance evaluation of an algorithm for the parallel solution of linear systems. The structure of the algorithm's task graph is representative of a class of recently proposed parallel linear system solvers. We develop a probabilistic model for two different parallel execution schemes depending on the synchronization policy adopted. The analytical solution of the model provides the mean algorithm execution time and therefore the speed-up and efficiency obtained with respect to the single processor environment.

This work was supported by the General Secretariat of Research and Technology of Greece, Grant No.9707.

1. Introduction

The growing development of modern multiprocessor technology provides an interesting alternative to the need of high computation speeds for the solution of large numerical applications. In the last decade a considerable number of parallel algorithms have been proposed, which should exploit the advantages of multiprocessor architectures. Some of these algorithms are parallel versions of existing sequential ones, other are completely new algorithms [1], [2], [9]. In all cases, the job is divided into a number of tasks, each executing concurrently with other tasks. This decomposition of a job into a set of tasks with well-defined interdependencies is governed by the logical structure of the parallel algorithm, which is usually represented by a computation graph, or task graph. The precedence relations among tasks imply the existence of synchronization points within the computation, which are the main factor limiting the effective parallelism of programs. An important performance measure concerning the execution of parallel algorithms is the processing time of a job described by a computation graph, which is used to derive the speed-up and the efficiency obtained with respect to the single processor case.

Several models have been proposed so far for the performance evaluation of parallel algorithms. Let us quote, for instance, models concerning numerical computations [3], [5], [7], tree-structured algorithms [6], [8], or general program structures [4]. In most cases, probabilistic models seem best adapted for the description of the issues involved in parallel computation.

In this paper, we consider a probabilistic model for the execution of a parallel algorithm for solving linear systems.

This algorithm is a parallel implementation of the Gauss-Jordan method with partial pivoting [9], described by a task graph of triangular structure with distinct computation levels and particular precedence constraints. The interest of this graph is that its general structure is the same for several direct or iterative linear system solvers involving various triangularisation techniques [9], so that the model can be used for the comparative evaluation of a class of parallel algorithms. We consider two different synchronization policies resulting to two parallel computation schemes. For the first one we obtain the exact solution assuming an arbitrary number of available processors. For the second policy, considering an infinite number of processors, we develop an approximate solution, whose accuracy is validated by simulation results.

In the next Section we describe the numerical algorithm and its parallel implementation. Section 3 concerns the general framework for our modeling approach, whereas in Sections 4 and 5 we present the model for the two synchronization policies considered.

2. The parallel algorithm

Let us consider the solution of linear algebraic equations of the form

$$Au = b \qquad (1)$$

where A is a real nonsingular dense matrix of order n, u is the n-dimensional vector of unknowns and b is a given n-dimensional vector. We consider the parallel implementation of the Gauss-Jordan method with partial pivoting for solving (1). The Gauss-Jordan

algorithm is a variation of the classical Gauss-elimination method in that it reduces A into a diagonal matrix instead of an upper triangular, so that the solution vector u is obtained immediately. In order to illustrate the decomposition of the algorithm into a set of tasks, we quote the sequential program for the diagonalisation of the matrix A using the Gauss-Jordan method [9]:

```
Program GAUSSJORDAN(A(n,n))
for k:=1 to n do
    Find ℓ such that                                   ⎤
    |A(ℓ,k)|=max(|A(k,k)|,...,|A(n,k)|)                |
    PIV(k):=ℓ {pivot row}                              |
    A(PIV(k),k)⟷A(k,k)                                 |
    c:=1/A(k,k)                                        |  T_k^k
    for i:=1 to n do                                   |
    Skip the value i=k                                 |
    A(i,k):=A(i,k)xc                                   ⎦

    for j:=k+1 to n do                                 ⎤
    A(PIV(k),j)⟷A(k,j)                                 |
    for i:=1 to n do                                   |  T_k^j,  j>k
      Skip the value i=k                               |
      A(i,j):=A(i,j)-A(i,k)xA(k,j)                     ⎦
```

(2)

As shown in the program, we consider a task to be the code segment which works on a particular column j for a particular value of k. We denote a task by T_k^j, $1 \leq k \leq j \leq n$. Taking into account the precedence constraints imposed by the sequential program, one can verify that the set

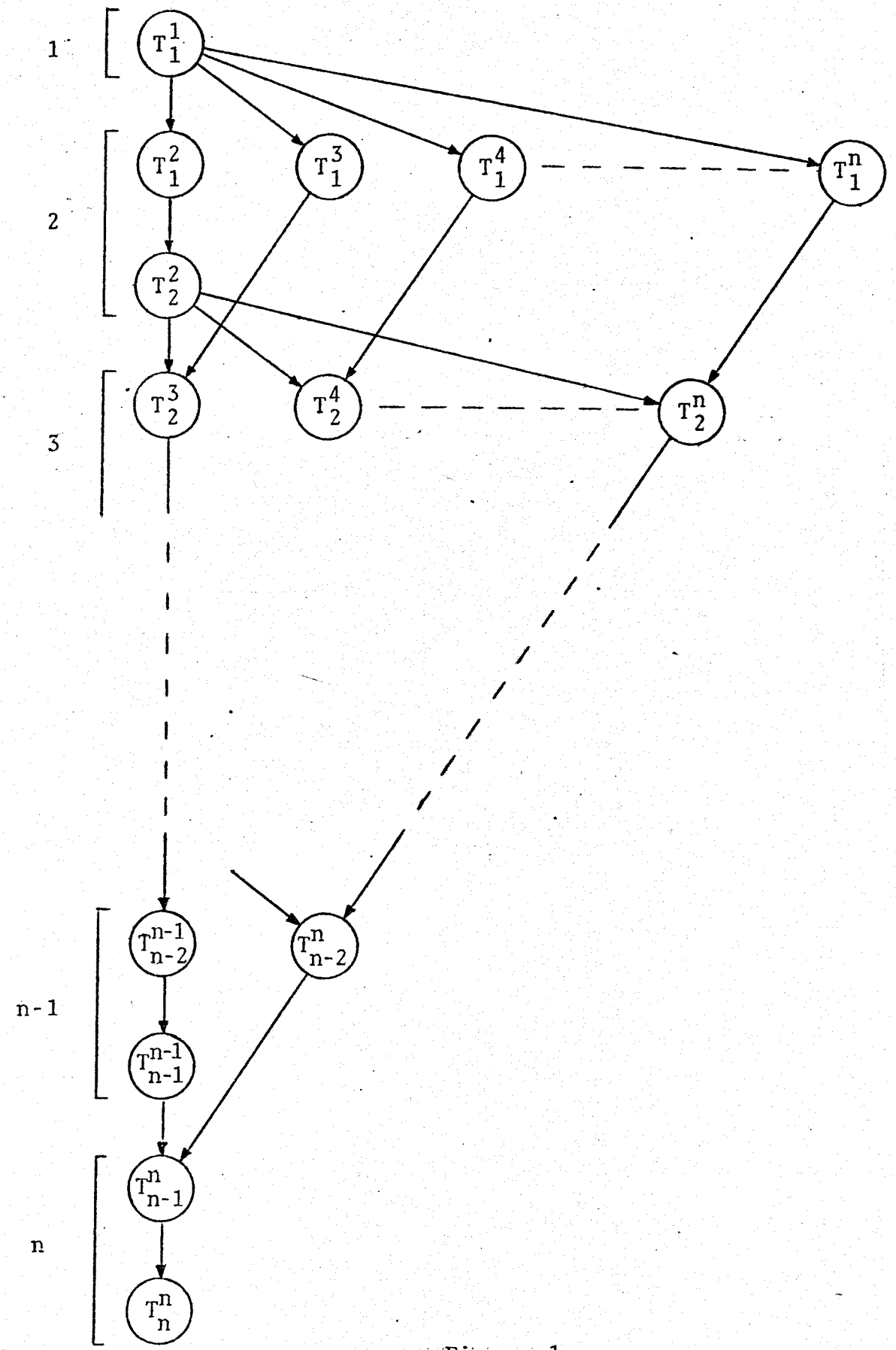

Level
1
2
3
n-1
n
T_1^1
T_1^2
T_1^3
T_1^4
T_1^n
T_2^2
T_2^3
T_2^4
T_2^n
T_{n-2}^{n-1}
T_{n-2}^n
T_{n-1}^{n-1}
T_{n-1}^n
T_n^n

Figure 1

$$\{T_k^{k+1}, T_k^{k+2}, \ldots, T_k^n\} \quad , \quad 1 \leq k < n$$

is composed of mutually noninterfering, which could be executed in parallel. This decomposition results to the maximally parallel computation graph of Fig.1, where vertices represent tasks and directed edges represent precedence relations. The computation graph is characterized by the processing times of the tasks; as a first approach, we can consider a deterministic evaluation for the processing times. The execution of T_k^k requires n-k comparisons, 1 division and n-1 multiplications; assuming that one comparison or one arithmetic operation consitutes one time step, then it follows that T_k^k requires 2n-k time steps. Similarly the execution of T_k^j needs n-1 multiplications and n-1 subtractions, namely, 2n-2 time steps. Thus, the processing times of the tasks are given by:

$$w(T_k^j) = \begin{cases} 2n-k & \text{if } k=j \\ 2n-2 & \text{if } k<j \end{cases} \qquad (3)$$

For the complete solution of (1) we still need n divisions which can also be carried out in parallel.

Using the above deterministic values for the task processing times, an optimal schedule is developed in [9] requiring n/2 processors. A similar parallel implementation applied to other linear system solving algorithms results to computation graphs of the same general structure. In all these cases, an optimal schedule on O(n) processors is obtained, thus exhibiting an advantage over previously developed methods which required a much larger number of processors [9].

3. General framework

Let us consider the execution of the task graph of Fig.1 on an asynchronous multiprocessor system of the MIMD type. We shall assume that processors communicate through shared memory and that the communication cost for cooperating tasks is not a dominant factor in the algorithm's execution.

The task graph is made of n horizontal levels defined as follows level k includes the tasks T_{k-1}^{j}, j=k,k+1,...,n plus the task T_k^k (Fig.1) Clearly level 1 includes only T_1^1.

The processing times of all tasks T_k^j, $1 \leq k < j \leq n$, are independent identically distributed random variables following an exponential distribution with parameter λ. On the other hand, the processing time of task T_k^k ($1 \leq k \leq n$), is an exponentially distributed random variable with parameter μ_k, where these random variables are mutually independent. We shall assume that the mean processing times of the tasks are given by the expressions (3) obtained in the previous. Section under deterministic assumptions concerning the execution times of arithmetic operations. Thus:

$$\left.\begin{aligned} 1/\mu &= 2n-k \\ 1/\lambda &= 2n-2 \end{aligned}\right\} \qquad (4)$$

The above probabilistic assumptions express the fact that, although a task requires in general a constant amount of service, its processing time depends upon the current state of the system due to overhead associated with resource sharing.

We are given a set of m processors, with m<n. Since the maximal parallelism of the computation graph is n-1 (Fig.1) we will refer to the case m=n-1 as the case of infinite available processors. Tasks are executed following the precedence relations represented by the structure of the computation graph. We shall consider two

different parallel execution schemes depending on the synchronization policy adopted:

1. Level-by-level policy. The tasks of a given level k are executed in parallel by the available processors. If some processor terminates the processing of a task and there still exists a task of the same level with no processor assigned to it, then the free processor is reassigned to one such task. At the begining of the processing of level k there is always a processor assigned to task T^k_{k-1}; also, since the tasks T^k_{k-1} and T^k_k must be executed sequentially, we assume that the some processor is assigned to both of them. Processors are synchronized at the end of execution of each level, that is, no task of level k+1 may be executed if there are non-executed tasks of level k, even if there exist free processors.
2. Anticipatory policy. At any instant of the algorithm processing no processor remains unassigned as long as there exists a task ready for execution with no processor assigned to it. A task is ready when all precedence constraints regarding it are satisfied. The assumptions introduced for the first policy concerning the tasks T^k_{k-1} and T^k_k hold for this policy too. When a processor terminates the processing of a task and there exist more than one ready tasks, then a task is chosen among those of minimum level (lowest-level-first policy).

The above synchronization policies are introduced in [8], where tree-structured programs are studied, and are referred to as non-anticipatory and anticipatory respectively.

4. Modeling of the level-by-level policy

We are interested in evaluating the mean processing time of the computation graph under the assumptions of the previous section. We

suppose that the m parallel processors are executing an infinity of such graphs, in the sense that as soon as the processing of task T_n^n of a graph is terminated, then the processing of task T_1^1 of another graph is immediately assumed (we will consider here that the time required to perform the n divisions at the end of the graph's execution is negligible). The system's behaviour can be described by a finite state-space Markov process whose states are defined as follows:

- The system is in state $(1',0)$ when the task T_1^1 is being exectuted.
- The system is in state (k, ℓ) $2 \leq k \leq n$, $0 \leq \ell \leq n-k$, when tha task T_{k-1}^k is being executed and there are ℓ non-executed tasks (including the ones being executed) among T_{k-1}^j, $k+1 \leq j \leq n$.
- The system is in state (k',ℓ) $2 \leq k \leq n$, $0 \leq \ell \leq n-k$, when the task T_k^k is being executed and there are ℓ non-executed tasks (including the ones being executed) among T_{k-1}^j, $k+1 \leq j \leq n$.
- The system is in state (k'',ℓ) $2 \leq k \leq n-1$, $1 \leq \ell \leq n-k$, when the execution of task T_k^k is terminated and there are ℓ non-executed tasks (including the ones being executed) among T_{k-1}^j, $k+1 \leq j \leq n$.

Synchronization at the end of execution of level k corresponds to exit from one of the states $(k',0)$ or $(k'',1)$.

The process is a finite state-space irreducible Markov process, so there exists a steady-state distribution that we will denote by π.

Let us define the following parameters for each level k:

$$\left.\begin{aligned} a_k &= \min(m-1, n-k) \\ b_k &= \min(m, n-k) \end{aligned}\right\} \qquad (5)$$

which correspond to the number of available processors for the execution of tasks T_{k-1}^j, $k+1 \leq j \leq n$, depending on the fact that one of the tasks T_{k-1}^k and T_k^k is being executed or not respectively.

The steady-state probabilities must satisfy the following system of linear equations:

$$\mu_1 \pi_{(1',0)} = \mu_n \pi_{(n',0)} \qquad (6)$$

$$
\left.
\begin{aligned}
&(a_k+1)\lambda\pi_{(k,n-k)} = \mu_{k-1}\pi_{((k-1)',0)} + \lambda\pi_{((k-1)'',1)} \\
&(a_k+1)\lambda\pi_{(k,\ell)} = \alpha_k\lambda\pi_{(k,\ell+1)} \quad , \quad a_k \le \ell \le n-k-1 \\
&(\ell+1)\lambda\pi_{(k,\ell)} = (\ell+1)\lambda\pi_{(k,\ell+1)} \quad , \quad 0 \le \ell \le a_{k-1}
\end{aligned}
\right\} \quad 2 \le k \le n \qquad (7)
$$

$$
\left.
\begin{aligned}
&(\mu_k + a_k\lambda)\pi_{(k',n-k)} = \lambda\pi_{(k,n-k)} \\
&(\mu_k + a_k\lambda)\pi_{(k',\ell)} = \lambda\pi_{(k,\ell)} + a_k\lambda\pi_{(k',\ell+1)} \quad , \quad a_k \le \ell \le n-k-1 \\
&(\mu_k + \ell\lambda)\pi_{(k',\ell)} = \lambda\pi_{(k,\ell)} + (\ell+1)\lambda\pi_{(k',\ell+1)} \quad , \quad 0 \le \ell \le a_k - 1
\end{aligned}
\right\} \quad 2 \le k \le n \qquad (8)
$$

$$
\left.
\begin{aligned}
&b_k\lambda\pi_{(k'',n-k)} = \mu_k\pi_{(k',n-k)} \\
&b_k\lambda\pi_{(k'',\ell)} = \mu_k\pi_{(k',\ell)} + b_k\lambda\pi_{(k'',\ell+1)} \quad , \quad b_k \le \ell \le n-k-1 \\
&\ell\lambda\pi_{(k'',\ell)} = \mu_k\pi_{(k',\ell)} + (\ell+1)\lambda\pi_{(k'',\ell+1)} \quad , \quad 1 \le \ell \le b_k - 1
\end{aligned}
\right\} \quad 2 \le k \le n-1 \qquad (9)
$$

as well as the normalizing condition that steady-state probabilities for all states must sum to 1.

From the solution of the above linear system we can readily obtain the mean parallel processing time of the graph T_p. In fact, T_p is given by the mean recurrence time of state $(1',0)$, since the process visits this state once for every graph execution. Hence:

$$T_p = [\mu_1\pi_{(1',0)}]^{-1} \qquad (10)$$

In order to derive the speed-up due to the parallel processing we need the mean sequential processing time T_s of the algorithm, which can be obtained by summing the mean processing times of all tasks:

$$T_s = \frac{1}{\mu_1} + \sum_{k=2}^{n} \left[(n-k+1)\,\frac{1}{\lambda} + \frac{1}{\mu_k} \right] \qquad (11)$$

So the speed-up and the efficiency of the parallel scheme can be expressed as:

$$S_p = T_s / T_p \qquad (12)$$

$$E_f = S_p / m \qquad (13)$$

We have plotted in Figure 2 and 3 some numerical results illustrating the behaviour of the parallel system. The curves of

Fig.2 represent the speed-up and the efficiency obtained for a problem of fixed size as a function of the number of available processors. The same quantities are shown in Fig.3 as a function of the problem size for selected values of the number of available processors. We first notice that the speed gain grows almost linearly as the size of the problem increases but is limited with respect to the number of processors used. This is due, on one hand, to the fact that the maximal parallelism of the algorithm is inherently limited and, on the other hand to synchronization delays which are particularly effective under the level-by-level synchronization policy. Another important remark is that the speed-up is almost stabilized for m>n/2, which validates the optimal scheduling developed in [9] using n/2 processors.

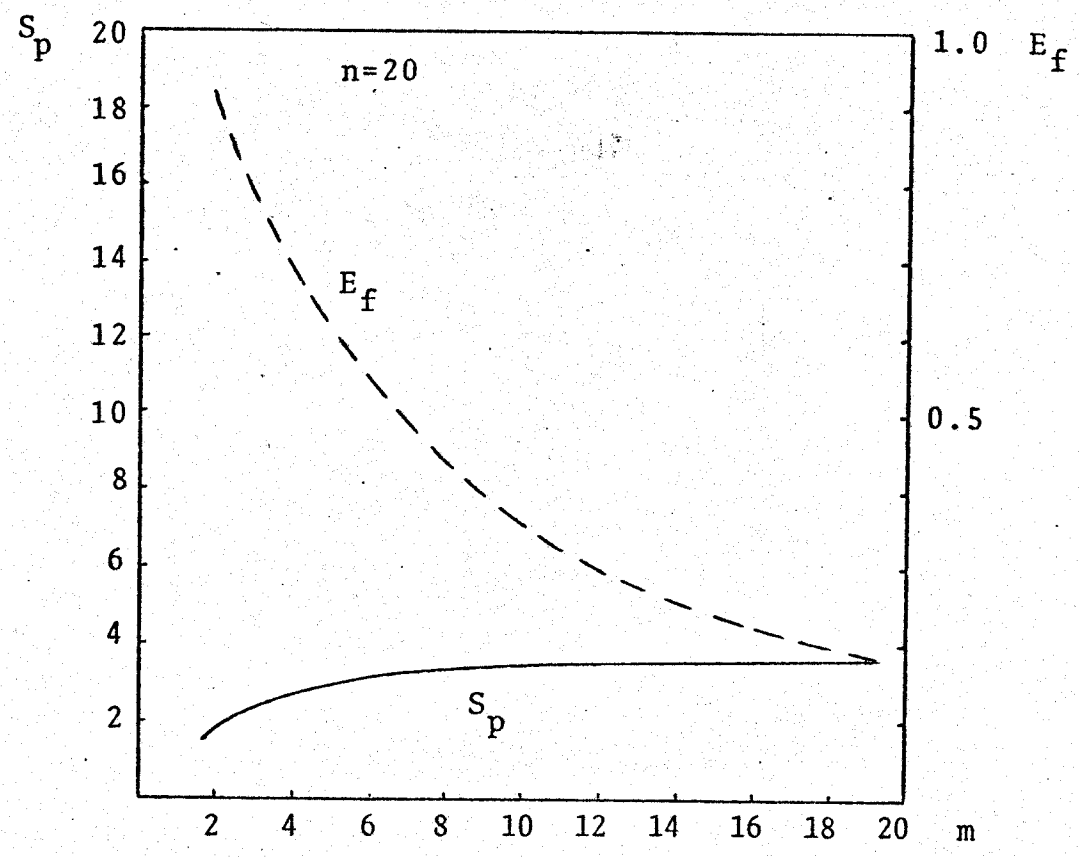
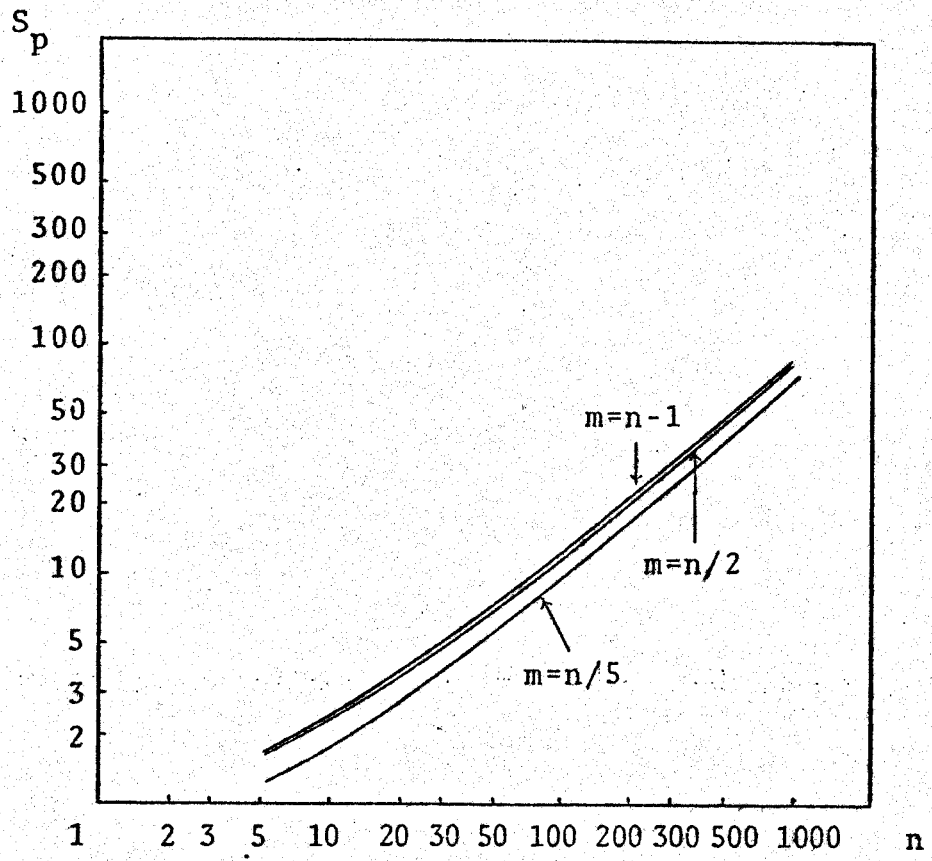


Figure 2


S_p
1000
500
300
200
100
50
30
20
10
5
3
2
m=n-1
m=n/2
m=n/5
1
2
3
5
10
20
30
50
100
200
300
500
1000
n

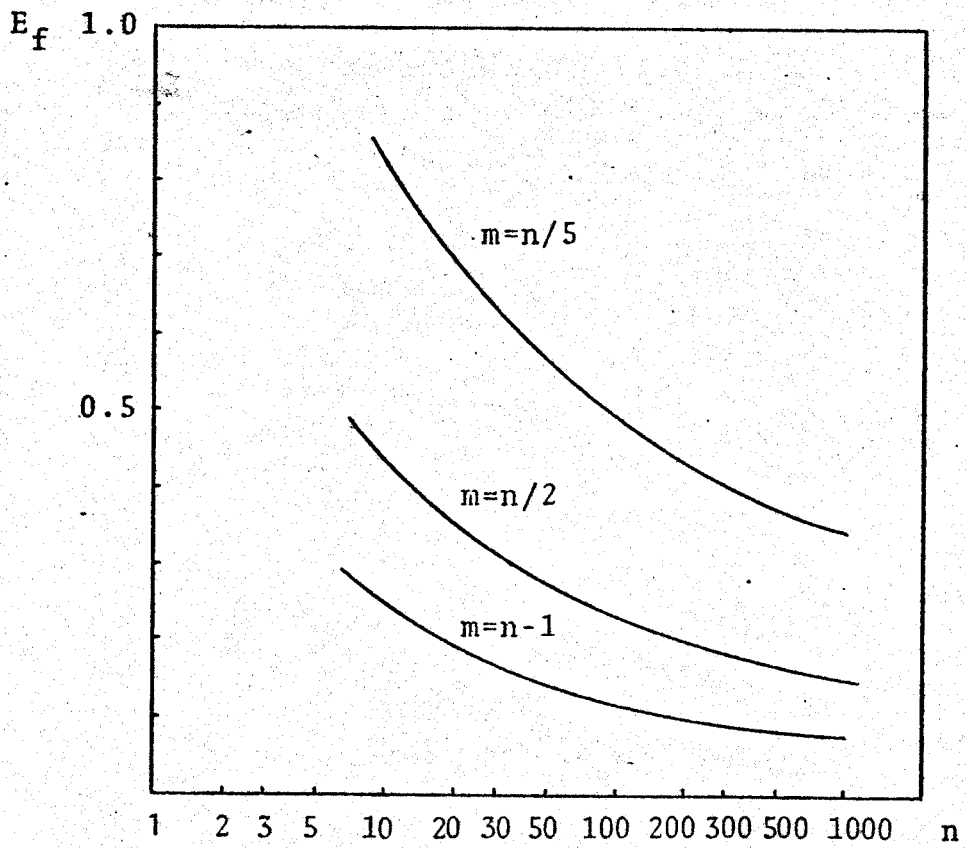

E_f
1.0
0.5
m=n/5
m=n/2
m=n-1
1
2
3
5
10
20
30
50
100
200
300
500
1000
n

Figure 3

5. Modeling of the anticipatory policy

As in the previous Section we want to evaluate the mean processing time of the computation graph under the assumptions of Section 3. Unfortunately, in the case of the anticipatory policy, a complete state description is impossible to handle due to the size of the state-space. Even in the limiting cases of two or infinitely many processors the model does not have an analytically tractable solution. So, we are going to develop a heuristic approach for the infinite number of processors case, based upon the principle of process decoupling by using a "markovian approximation" [3]. According to this method we express the interaction between concurrent processes through coupling parameters. These parameters are evaluated by separately solving parts of the whole problem and taking into account only the first order effects of coupling between processes. The results obtained using this approach will be validated by simulation results.

Our heuristic method is based upon the following considerations:

- The parallel execution of the computation graph (Fig.1) using an infinite number of processors can be viewed as the execution of n-1 concurrent processes. The first of these processes, which will be referred to as the "main" process, concerns the execution of the tasks $T_1^1, T_1^2, T_2^2, T_2^3, \ldots, T_{n-1}^n, T_n^n$. In fact, these tasks constitute the longest path of the graph and the time required for the execution of this path determines the processing time of the algorithm. Besides the main process, we consider the n-2 processes concerning the diagonal paths of the graph $\{T_i^j,\ 1 \leq i \leq j-2\}$ for $3 \leq j \leq n$. These processes will be called "secondary" processes, and will be numbered from 2 to n-1.
- The secondary processes do not directly interact with each other, but each one of them interacts with the main process. In fact, a secondary process k can be blocked at level i ($2 \leq i < k$) due to waiting for completion of the task T_i^i. On the other hand the main

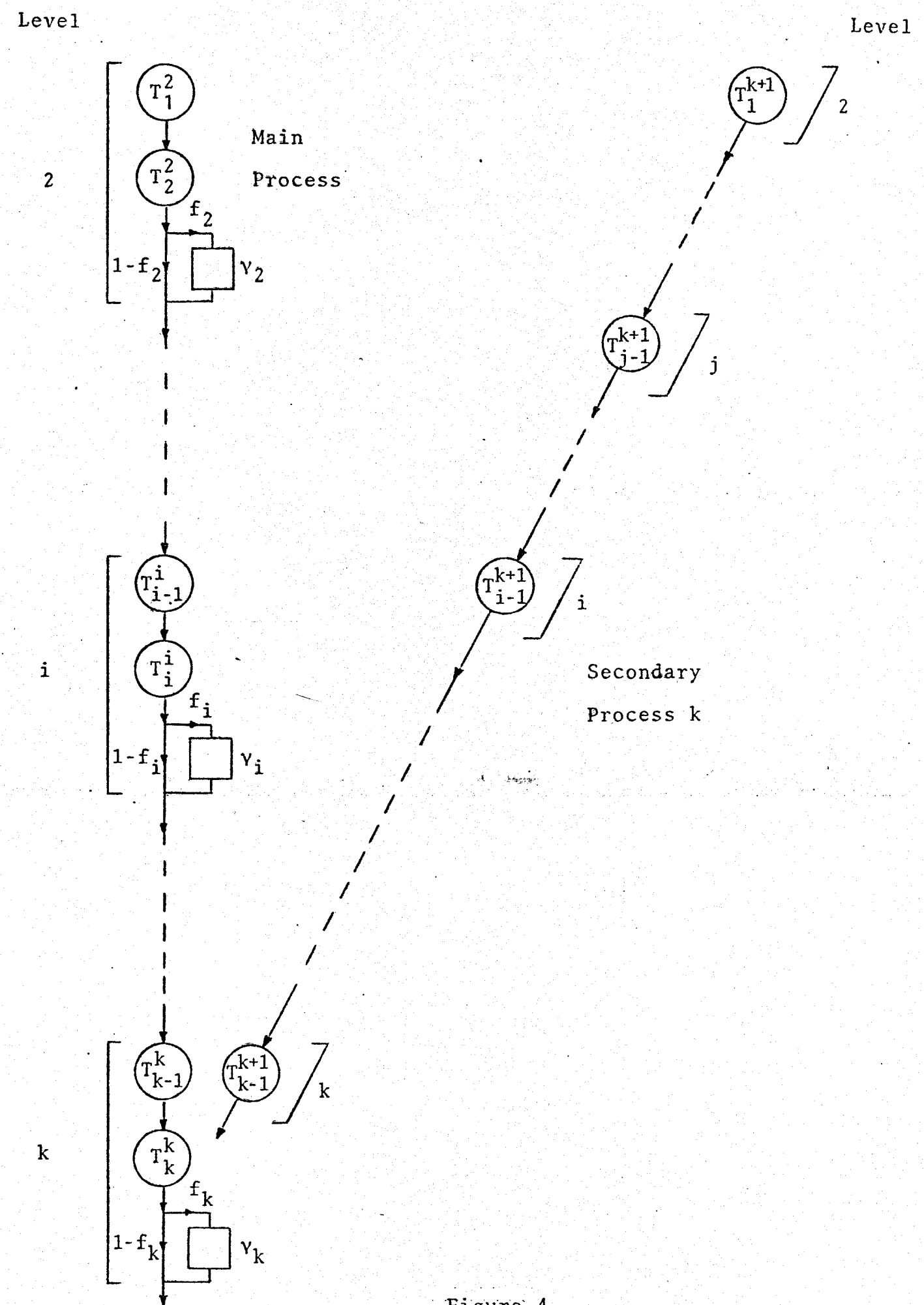

Level
Level
Main
Process
Secondary
Process k
2
i
k
j
T_1^2
T_2^2
f_2
$1-f_2$
ν_2
T_{i-1}^i
T_i^i
f_i
$1-f_i$
ν_i
T_{k-1}^k
T_k^k
f_k
$1-f_k$
ν_k
T_1^{k+1}
T_{j-1}^{k+1}
T_{i-1}^{k+1}
T_{k-1}^{k+1}

Figure 4

process can proceed to level k+1, only if the execution of secondary process k has terminated.

- In order to obtain the mean processing time of the graph, we are going to consider the behaviour of the main process taking into account its interaction with each one of the secondary processes studied separately. A secondary process k can produce delay to the main process at the end of execution of level k. This delay will be expressed by means of a branching probability f_k towards a waiting state with mean residence time $1/\gamma_k$ (Fig.4). The coupling parameters f_k, γ_k can be obtained by solving a markovian model describing the evolution of these two processes independently of the remaining secondary processes. Since this model concerns the behaviour of the main process up to level k, it can be successively solved for $2 \leq k \leq n-1$, each time yielding the corresponding coupling parameters. It should be noticed, that the solution of the model for a particular value k takes into account the effect of the secondary processes i, $2 \leq i \leq k-1$ through the coupling parameters f_i, γ_i that have already been determined.

We proceed now to the description of the model concerning the interaction of the main process with secondary process k ($2 \leq k \leq n-1$). For each value k the model represents the execution of a part of the main process, namely from level 2 to level k. After succesively solving the model for all values of k we can obtain the mean processing time for the part of the graph composed of levels 2 to n-1. The contribution of levels 1 and n to the total mean processing time can be added straightforwardly, since the execution of these levels concerns only the main process.

We consider that the part of the graph corresponding to the secondary process k is executed infinitely many times. Our main simplifying assumption is that the waiting states of the main process

at levels i, $2 \leq i \leq k-1$ are markovian. Their behaviour is characterized by the parameters f_i, γ_i, which are known since they have already been computed while solving the model for the corresponding secondary process i. We are led to a finite state-space Markov process with states defined as follows (Fig.4).

- The main process is in state i (resp. i'), $2 \leq i \leq k$, when the task T^i_{i-1} (resp. T^i_i) is being executed.
- The main process is in state i'', $2 \leq i \leq k$, when it is waiting for the corresponding secondary process i to terminate execution.
- The secondary process k is in State j, $2 \leq j \leq k$, when the task T^{k+1}_{j-1} is being executed.
- The secondary process k is in state j'', $2 \leq j \leq k$, when it is waiting for completion of the task T^j_j of the main process.
- According to the above definitions the possible states of the Markov process are:

$$\left.\begin{array}{l}(i,j),(i',j),(i'',j),\\ (i,i''),(i',i'')\end{array}\right\} \quad 2 \leq i \leq k$$

$$(i'',i+1),(i'',(i+1)''), \qquad 2 \leq i \leq k-1$$

- Exit from one of the states (k',k'') or (k'',k) implies the end of execution of the corresponding part of the graph.

The Markov process is irreducible with finite state-space so there exists a steady-state distribution, that will be denoted by π and satisfies the following system of linear equations ($2 \leq k \leq n-1$):

$$\left.\begin{array}{l} 2\lambda\pi_{(2,2)} = \lambda\pi_{(k'',k)} + \mu_k \pi_{(k',k'')} \\ \lambda\pi_{(2,2'')} = \lambda\pi_{(2,2)} \end{array}\right\} \tag{14}$$

$$\left.\begin{array}{l} 2\lambda\pi_{(i,2)} = (1-f_{i-1})\mu_{i-1}\pi_{((i-1)',2)} + \gamma_{i-1}\pi_{((i-1)'',2)} \\ 2\lambda\pi_{(i,j)} = (1-f_{i-1})\mu_{i-1}\pi_{((i-1)',j)} + \gamma_{i-1}\pi_{((i-1)'',j)} + \lambda\pi_{(i,j-1)}, \ 2<j<i \\ 2\lambda\pi_{(i,i)} = \gamma_{i-1}\pi_{((i-1)'',i)} + \lambda\pi_{(i,i-1)} + (1-f_{i-1})\mu_{i-1}\pi_{((i-1)',(i-1)'')} \\ \lambda\pi_{(i,i'')} = \gamma_{i-1}\pi_{((i-1)'',i'')} + \lambda\pi_{(i,i)} \end{array}\right\} \ 2<i\leq k \tag{15}$$

$$
\left.
\begin{aligned}
&(\mu_i+\lambda)\pi_{(i',2)} = \lambda\pi_{(i,2)} \\
&(\mu_i+\lambda)\pi_{(i',j)} = \lambda\pi_{(i,j)} + \lambda\pi_{(i',j-1)} \quad , \quad 2<j\leq i \\
&\mu_i\pi_{(i'',i'')} = \lambda\pi_{(i,i'')} + \lambda\pi_{(i',i)}
\end{aligned}
\right\} \quad 2\leq i\leq k \qquad (16)
$$

$$
\left.
\begin{aligned}
&(\gamma_i+\lambda)\pi_{(i'',2)} = f_i\mu_i\pi_{(i',2)} \\
&(\gamma_i+\lambda)\pi_{(i'',j)} = f_i\mu_i\pi_{(i',j)} + \lambda\pi_{(i'',j-1)} \quad , \quad 2\leq j\leq i \\
&(\gamma_i+\lambda)\pi_{(i'',i+1)} = f_i\mu_i\pi_{(i',i'')} + \lambda\pi_{(i'',i)} \\
&\gamma_i\pi_{(i'',(i+1)'')} = \lambda\pi_{(i'',i+1)}
\end{aligned}
\right\} \quad 2\leq i<k \qquad (17)
$$

$$
\left.
\begin{aligned}
&\lambda\pi_{(k'',2)} = \mu_k\pi_{(k',2)} \\
&\lambda\pi_{(k'',j)} = \mu_k\pi_{(k',j)} + \lambda\pi_{(k'',j-1)} \quad , \quad 2<j\leq k
\end{aligned}
\right\} \qquad (18)
$$

plus the normalizing condition that the steady-state probabilities must sum to 1.

From the solution of the above linear system we obtain the coupling parameters f_k, γ_k as follows:

- The branching probability f_k can be derived as the asymptotic relative frequency with which the main process visits the waiting state k'' during an execution of the corresponding part of the job.

  The mean departure rate from state k'' (and hence the rate of visiting this state) is $\lambda\pi_{(k'',k)}$ since the main process leaves the waiting state only through state (k'',k).

  On the other hand, the mean rate at which execution is completed is equal to the departure rate from state $(2,2)$, which is $2\lambda\pi_{(2,2)}$, since each execution includes exactly one visit to this state. Hence, the desired quantity is:

$$
f_k = \frac{\pi_{(k'',k)}}{2\pi_{(2,2)}} \qquad (19)
$$

- The exit rate γ_k from the waiting state k'' can be derived as the departure rate from state k'' given that the main process has entered this state. The probability of the latter event is $\sum_{j=2}^{k}\pi_{(k'',j)}$. Hence, we obtain:

$$\gamma_k = \frac{\lambda\pi(k'',k)}{\sum_{j=2}^{k}\pi(k'',j)} \qquad (20)$$

Summarizing, our heuristic approach can be described as follows:

- For the values of k from 2 to n-1 we successively solve the linear system (14)-(18) and compute the corresponding parameters f_k, γ_k from (19), (20) respectively.
- At each step the solution of the linear system makes use of the values of the parameters computed at the previous steps.
- At the end of this procedure the mean parallel processing time of the graph can be obtained as the mean processing time of the main process:

$$T_p = (n-1)\,\frac{1}{\lambda} + \sum_{k=1}^{n}\frac{1}{\mu_k} + \sum_{k=2}^{n-1}\frac{f_k}{\gamma_k} \qquad (21)$$

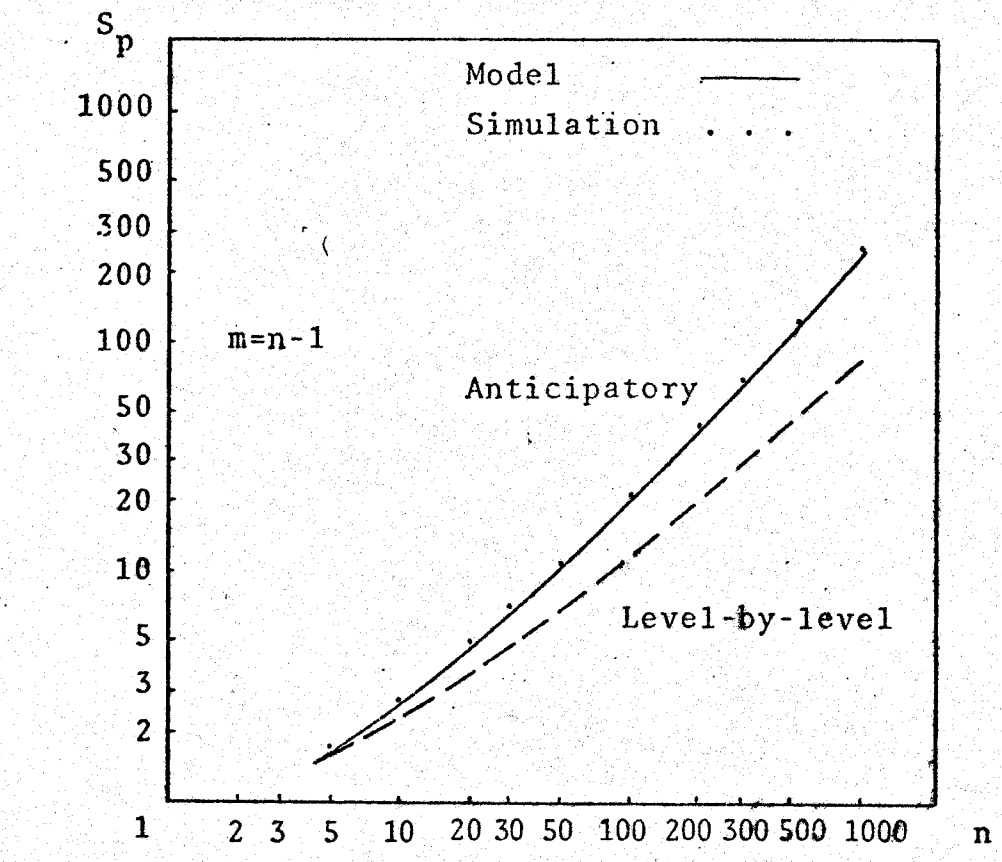
By using the value T_s of the mean sequential processing time from (11) we obtain for the speed-up and the efficiency of the parallel scheme:

$$S_p = T_s/T_p \qquad (22)$$

$$E_f = S_p/(n-1) \qquad (23)$$

It can be easily verified that the method provides for an easily implemented, efficient computation requiring $O(n^2)$ time and $O(n)$ space.

The accuracy of our markovian approximation was validated by simulation results obtained using the method of independent replications. More precisely, for each value of the problem size n considered, after a minimum of 30 replications, the replication process was carried on until a 5 percent relative precision was obtained considering a 95 percent confidence interval or until a maximum of 1000 replications had been performed. In all cases examined, the values of the mean parallel processing time obtained from the model exceeded the corresponding simulation estimates by less than 5 percent.


S_p
1000
500
300
200
100
50
30
20
10
5
3
2
1
Model
Simulation . . .
m=n-1
Anticipatory
Level-by-level
2 3 5 10 20 30 50 100 200 300 500 1000
n

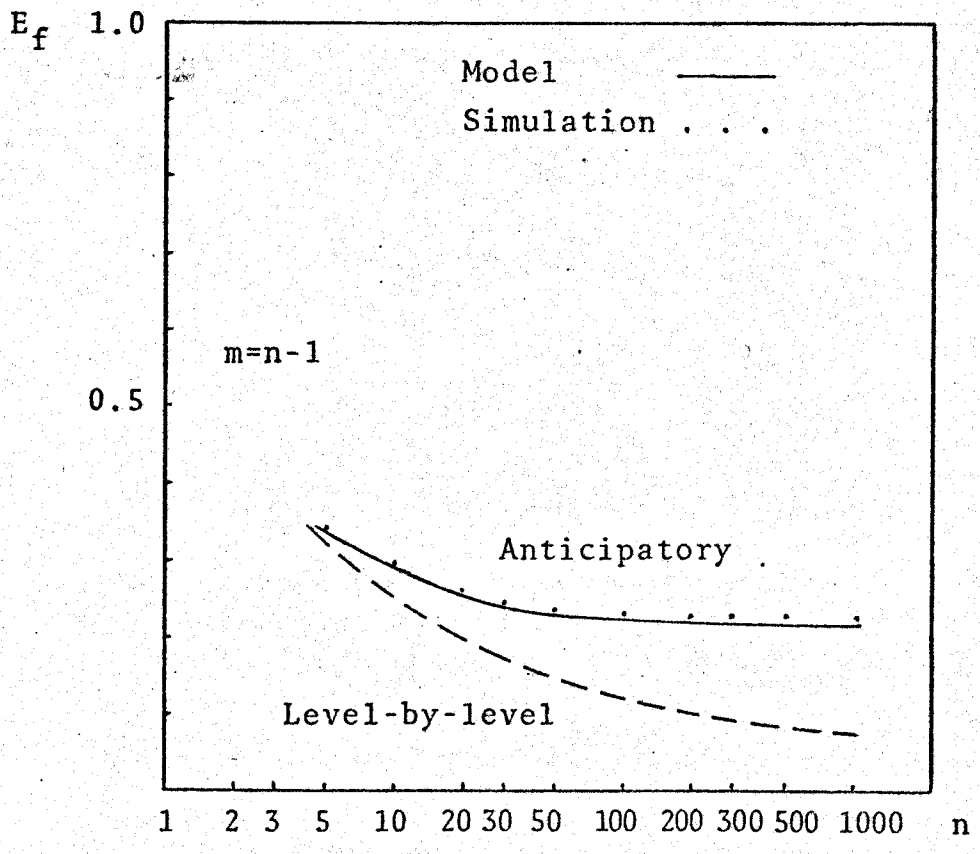

E_f
1.0
0.5
Model
Simulation . . .
m=n-1
Anticipatory
Level-by-level
1 2 3 5 10 20 30 50 100 200 300 500 1000
n

Figure 5

Numerical results for the anticipatory policy are represented by the bold line curves of Fig.5, which illustrate the variation of the speed-up and the efficiency as a function of the problem size n. The corresponding curves for the level-by-level policy are plotted in dashed line for comparison and single points represent simulation results. Again the speed-up varies almost linearly with respect to n, whereas, as expected, the anticipatory policy is clearly superior to the level-by-level policy.

6. Conclusion

We have studied in this paper the performance of a particular parallel processing case, which is representative of a class of parallel linear system solvers. A markovian model of the system was introduced considering two different synchronization policies for the parallel execution of the algorithm. The model was solved exactly in the first case for an arbitrary number of processors, whereas a heuristic method was developed providing the solution in the other case for an infinite number of processors. The efficiency of our heuristic method was validated by simulation results. Numerical results obtained from the model illustrate the speed-up and the efficiency of the parallel computation scheme. The speed-up is important but is far from beeing ideal, because of the limited intrinsic parallelism of the algorithm and the effects of synchronization.

Our experience has shown that analytical methods based upon the theory of stochastic processes may be very useful in the performance evaluation of parallel processing. Furthermore, our heuristic decoupling approach, which provided quite good results, is an interesting alternative in the study of complex distributed systems.

References

[1] D.Heller, "A Survey of Parallel Algorithms in Numerical Linear Algebra", SIAM Rev.20 (1978).

[2] D.J.Evans, M.Hatzopoulos, "A Parallel Linear System Solver", Intern. J. Comput. Math. 7 (1979).

[3] B.Plateau, A.Staphylopatis, "Modeling of the Parallel Resolution of a Numerical Problem on a Locally Distributed Computing System", ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems, Seattle, WA, Aug. 1982.

[4] P.Heidelberger, K.S.Trivedi, "Queueing Network Models for Parallel Processing with Asynchronous Tasks", IEEE Trans. on Computers, Vol. C-31, No.11, Nov. 1982.

[5] E.Gelenbe, A.Lichnewsky, A.Staphylopatis, "Experience with the Parallel Solution of Partial Differential Equations on a Distributed Computing System", IEEE Trans. on Computers, Vol. C-31, No.12, Dec. 1982.

[6] G.Fayolle, P.J.B.King, I.Mitrani, "On the Execution of Programs by Many Processors", PERFORMANCE '83, A.K.Agrawala ans S.K.Tripathi (editors), North-Holland, 1983.

[7] L.M.Adams, T.W.Crockett, "Modeling Algorithm Execution Time on Processor Arrays", IEEE Computer, July 1984.

[8] Ph.Mussi, Ph.Nain, "Evaluation of Parallel Execution of Program Tree Structures", ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems, Cambridge, Mass., Aug. 1984.

[9] M.Hatzopoulos, N.M.Missirlis, "Advantages for Solving Linear Systems in an Asynchronous Environment", Journal of Computational and Applied Math. 12 & 13, 1985.