The organization of circuit analysis on array architectures

Citation for published version (APA):
Kees, H. G. M. (1982). *The organization of circuit analysis on array architectures*. [Phd Thesis 1 (Research TU/e / Graduation TU/e), Electrical Engineering]. Technische Hogeschool Eindhoven.
https://doi.org/10.6100/IR112705

DOI:
10.6100/IR112705

Document status and date:
Published: 01/01/1982

Document Version:
Publisher's PDF, also known as Version of Record (includes final page, issue and volume numbers)

THE ORGANIZATION OF CIRCUIT ANALYSIS ON ARRAY ARCHITECTURES

H.G.M. KEES

THE ORGANIZATION OF CIRCUIT ANALYSIS ON ARRAY ARCHITECTURES

PROEFSCHRIFT

TER VERKRIJGING VAN DE GRAAD VAN DOCTOR IN DE TECHNISCHE WETENSCHAPPEN AAN DE TECHNISCHE HOGESCHOOL EINDHOVEN, OP GEZAG VAN DE RECTOR MAGNIFICUS, PROF.IR. J. ERKELENS,VOOR EEN COMMISSIE AANGEWEZEN DOOR HET COLLEGE VAN DEKANEN IN HET OPENBAAR TE VERDEDIGEN OP
DINSDAG 25 MEI 1982 TE 16.00 UUR

DOOR

HENDRIKUS GERARDUS MARIA KEES

GEBOREN TE BUDEL

DIT PROEFSCHRIFT IS GOEDGEKEURD DOOR DE PROMOTOREN:

Prof.dr.ing. J.A.G. Jess

en

Prof.dr. P.M. Dewilde.

CONTENTS

0. INTRODUCTION

The design process of intergrated electronic circuits requires circuit simulation facilities. The circuit analysis program, which computes the transient response of the designed circuit, is the basic simulation tool.
The processing of the circuit analysis program requires much computing time. In order to speed up the design process it is necessary to reduce the computing time. There are two possibilities: the improvement of the organization of the analysis program itself or a faster computer.
The obvious way is to replace the computer hardware by faster hardware however, very much improvement is not possible because of physical limits. Another way is the application of more hardware to do things in parallel.
The organization of a circuit analysis program on such a computer configuration is the subject of the thesis.

In order to achieve higher throughput computer configurations with multiple 'central' processors have been proposed and built [0.1]-[0.4]. If the processors are able to co-operate in processing a single job then the computer configuration will be called a "parallel computer" and the job will be called to be "processed in parallel". The job indicates the computation or process which is required by the user.
If parallel processing is going to be applied the following questions and problems arise:
- What kinds of jobs will be offered.
- How to construct algorithms suited for parallel processing. In order to make parallel processing possible the job must be decomposed into a set of tasks. These tasks must be processed by the processors according to an ordering, specified by the algorithm.
- What is an appropriate architecture. The various resources as processors, memories, buses, registers, etc., must be specified as well as the way they are interconnected.
- What is the appropriate system software.

These questions and problems should be answered and solved such that the desired performance is optimized as much as possible. Quantities to judge on performance are for instance: throughput, cost for hard-

ware etc.
It will be clear that the above stated problems and questions are not independent of each other. The difficulty of the optimization problem depends highly on the first item 'what kinds of jobs will be offered'. Here they will be restricted to the above mentioned circuit analysis programs which leads to the design of a special purpose computer.
The performance which will be optimized is the average processing time of the jobs belonging to the considered class under the condition that the user has not to supply any additional commands to the computer.

In order to 'solve' the above stated optimization problem to each of the stated questions and problems an answer or a solution will be formulated depending on the previously taken decisions. This is mainly accomplished in chapter 2, after in chapter 1 the structure of a circuit analysis program is considered. Chapter 2 consists of three parts.
Firstly a parallel algorithm will be derived directly from the results of chapter 1. A speedup analysis will be given. It shows which tasks will be further considered for parallel processing.
Secondly the special purpose computer and its organization are presented.
Thirdly the job and the parallel computer are brought into one model, the scheduling model. The system software uses this model to accomplish an optimal match between the resources of the special purpose computer and the resource demands. The system software is extended by decomposition and scheduling procedures.
In chapter 3 the task, which solves the set of linear equations resulting from the discretized and linearized circuit equations, is regarded as the primary job which must be decomposed into tasks. This is because in section 2.1 it is shown that the required processing time to solve the set of linear equations limits the speedup which can be achieved.
In chapter 4 the scheduling is demonstrated for the parallel solution of the set of linear equations, because the scheduling of this job is the most critical problem.
In chapter 5 some results of the decomposition and scheduling of the solution job for the set of linear equations are presented.

Finally, in chapter 6 some remarks are made about the remaining tasks and the conclusions are given.

1. CIRCUIT ANALYSIS

1.0 Introduction

Each circuit analysis method [1.1] starts with setting up a set of equations (1.0.1) which describes the circuit with excitation $\underline{e}(t)$ to be analysed:

$$\underline{f}(\dot{\underline{x}}(t),\underline{x}(t),\underline{e}(t),t) = \underline{0} \quad t \in [t_b,t_e] \quad \underline{x}(t_b) = \underline{x}_1 \tag{1.0.1}$$

These time dependent equations are in general nonlinear. The desired solution $\underline{x}(t)$ for $t \in [t_b,t_e]$ will be given at the discrete time instants $t_1 = t_b, t_2, \ldots, t_\ell = t_e$. To this end an approximation $\underline{x}_n$ for $\underline{x}(t_n)$ will be determined at each discrete time instant t_n. The time derivatives at each discrete time instant are replaced by some appropriate approximation in the form of a differentiation formula depending on the current value $\underline{x}_{n+1}$ and past values of $\underline{x}$:

$$\dot{\underline{x}}_{n+1} \simeq \underline{df}(\underline{x}_{n+1},\underline{x}_n,\underline{x}_{n-1},\ldots) = \underline{d}(\underline{x}_{n+1}) \tag{1.0.2}$$

By these substitutions for each discrete point of the time axis a set of nonlinear difference equations is obtained:

$$\underline{f}_d(\underline{x}_{n+1}) = \underline{f}(\underline{d}(\underline{x}_{n+1}),\underline{x}_{n+1},\underline{e}_{n+1},t_{n+1}) = \underline{0} \quad n \in 0,1,\ldots,\ell-1 \tag{1.0.3}$$

The solution of (1.0.3) for $\underline{x}_n$ is obtained by solving first for $\underline{x}_2,\underline{x}_3,\ldots,\underline{x}_n$ because the differentiation formula requires past values of $\underline{x}$.

The nonlinear difference equations are solved for $\underline{x}_{n+1}$ by an iteration method. The Newton-Raphson (N-R) iteration, given by (1.0.4), serves this purpose.

The iteration is started with $\bar{\underline{x}}_{n+1}$ which is supposed to be an appropriate prediction for $\underline{x}_{n+1}$. $A^{(j)}_{n+1}$ is the Jacobian, see (1.0.5)

$$A^{(j)}_{n+1}(\underline{x}^{(j+1)}_{n+1} - \underline{x}^{(j)}_{n+1}) = -\underline{f}_d(\underline{x}^{(j)}_{n+1}); \quad \underline{x}^{(0)}_{n+1} = \bar{\underline{x}}_{n+1} \text{ for } n = 0,1,\ldots,\ell-1 \tag{1.0.4}$$

$$A^{j}_{n+1} = \frac{\partial}{\partial \underline{x}} \underline{f}_d(\underline{x})\Big|_{\underline{x} = x^{(j)}_{n+1}} \tag{1.0.5}$$

In the next sections the time discretization, the circuit description and some aspects of complexity and organization will be presented.

1.1 Time discretization

The total amount of computational work depends highly on the number of time steps. The number of necessary time steps depends on the differentiation formula applied, the circuit itself and the time interval, $TT = t_e - t_b$. In general the circuit is stiff.This requires differentiation formulas with the property that the time step and order can be adjusted easily. To be concrete, the prediction-based differentiation formulas (PBD) [1.2] are chosen and will be restated here. The superscript of the variables denotes the order of the PBD formula.

The prediction formula for $\bar{x}_{n+1}$ of order i+1 is defined recursively by:

$$\bar{x}^{i+1}_{n+1} = \bar{x}^{i}_{n+1} + d_i(x_n - \bar{x}^{i}_{n}) \text{ and } \bar{x}^{1}_{n+1} = x_n \qquad (1.1.1)$$

with d_i defined by

$$d_1 = h_1/h'_1 \quad ; \quad d_{i+1} = d_i . h_{i+1}/h'_{i+1}$$

where

$$h_i = t_{n+1} - t_{n+1-i} \text{ and } h'_i = t_n - t_{n-i}.$$

The k-th order differentiation formula is given by:

$$\dot{x}_{n+1} \simeq \sum_{i=1}^{k} (x_{n+1} - \bar{x}^{i}_{n+1})/h_i = d^k(x_{n+1}) \qquad (1.1.2)$$

Evaluation of $\dot{x}_{n+1}$ needs the recursive calculation of the predictions for x_{n+1} given by (1.1.1).

The time step is variable and increasing or decreasing the order is simply performed by changing the number of terms in (1.1.2).

The performance is measured by the local truncation error $E^k_{n+1}(h_1) = x(t_{n+1}) - x^c_{n+1}$, where x^c_{n+1} is the solution of (1.0.3) if $x_n, \bar{x}^i_n, i=2,\ldots,k$ are exact. Estimates for the truncation error depending on the order are given by:

$$E^{k}_{n+1}(h_1) \simeq (\bar{x}^{k+1}_{n+1} - x^{c}_{n+1}) / \sum_{i=1}^{k} (h_{k+1}/h_i) \tag{1.1.3}$$

$$E^{k+u}_{n+1}(h_1) \simeq (\bar{x}^{k+1+u}_{n+1} - x^{c}_{n+1} - E^{k}_{n+1}(h_1)) / (\sum_{i=1}^{k+u} h_{k+u+1}/h_i) \tag{1.1.4}$$

for $u \in \{-1,1\}$

The local truncation error is used to control the order as well as the time step. Let $\varepsilon(v)$ denote the maximum allowed error of component v of $\underline{x}$ over the interval TT. This requirement is assured to be met if the local truncation error per time unit is smaller than $\varepsilon(v)/TT$. Let $E^{m}(h,v)$ denote the truncation error of component v. The maximum time step $h^{m}(v)$ for component v made by a PBD formula of order m is given by:

$$K^{m}_{n+1}(h^{m}(v)/K^{m}_{n+1}(h_1) = \varepsilon(v)h^{m}(v)/(TT.E^{m}_{n+1}(h_1,v)) \tag{1.1.5}$$

where

$$K^{m}_{n+1}(h) = -h \prod_{i=2}^{m} (h+h'_{i-1})/(1/h + \sum_{i=2}^{m} 1/(h+h'_i)) \text{ with } h'_1,h'_2,\ldots,h'_m$$

at time t_{n+1}.

Let $h^{m} = \min(\{h^{m}(v) \mid v \in I_{xc}\})$, where I_{xc} is the set of the controlled components, and $h^{j} = \max(h^{k-1},h^{k},h^{k+1})$, then the new time instant $t_{n+2} = t_{n+1} + h$ will be calculated with PBD formulas of order j.

1.2 Circuit description

The Modified Nodal Analysis (MNA) approach [1.3] will be used to state the circuit description in this thesis.

Assume a circuit with (p+1) nodes and q elements (an r terminal element with r > 2 is counted as r elements and is described by r relations as given by (1.2.1) or (1.2.2)).

Let $\underline{v}_n = (v_{n1},\ldots,v_{np})^T$ and $\underline{i}_b = (i_{b1},\ldots,i_{bq})^T$ denote the node voltages with respect to the reference node and the (branch) currents, respectively.

The structure of the circuit and the orientation of the currents are given by the incidence matrix K.

The elements are described by the constitutive relations given by:

$$g_\ell(\dot{\underline{v}}_n, \dot{\underline{i}}_b, \underline{v}_n, \underline{i}_b, t) = 0 \quad \text{for} \quad \ell \in \{1,2,\ldots,q\}. \tag{1.2.1}$$

If the branch current can be given explicitly the relation is given by:

$$f_\ell(\dot{\underline{v}}_n, \dot{\underline{y}}_\ell, \underline{v}_n, \underline{y}_\ell, t) = i_{b\ell} \quad \text{for} \quad \ell \in \{1,2,\ldots,q\} \tag{1.2.2}$$

where : $\underline{y}_\ell = (i_{b1},\ldots,i_{b(\ell-1)},i_{b(\ell+1)},\ldots,i_{bq})^T$.

The output variables in the MNA approach are given by: $\underline{x}^T = (\underline{v}_n^T \mid \underline{i}^T)$. The components of $\underline{i}$ are those branch currents which cannot be given by (1.2.2) or are desired as an output variable or are used to control other elements. (The current $i_{b\ell}$ controls an element k if the constitutive relation of this element depends on $i_{b\ell}$). The currents appearing in $\underline{i}$ are called "output currents". Let u denote the number of output currents. Assume the elements are renumbered such that elements, whose branch current is an output current, have got a number ℓ such that $p < \ell \le p+u$.
The MNA equations, consisting of the p Kirchhoff's current equations and the u constitutive relations of elements, whose branch currents are output currents, are given by:

$$\left.\begin{array}{l} K \cdot \begin{bmatrix} f_1(\dot{\underline{v}}_n, \dot{\underline{y}}_1, \underline{v}_n, \underline{y}_1, t) \\ \vdots \\ f_s(\dot{\underline{v}}_n, \dot{\underline{y}}_s, \underline{v}_n, \underline{y}_s, t) \\ \hline \underline{i} \end{bmatrix} = \underline{0} \\ g_{p+1}(\dot{\underline{v}}_n, \dot{\underline{i}}_b, \underline{v}_n, \underline{i}_b, t) = 0 \\ \vdots \qquad\qquad \vdots \\ g_{p+u}(\dot{\underline{v}}_n, \dot{\underline{i}}_b, \underline{v}_n, \underline{i}_b, t) = 0 \end{array}\right\} \tag{1.2.3}$$

Time discretization and substitution of the time derivatives followed by linearization by means of the N-R method results in (1.0.4).

Consider an element ℓ between the nodes m and n with a current direction from m to n. The contribution of this element to the MNA set

of linear equations for the jth iteration at time $t = t_{n+1}$ and $\underline{x} = \underline{x}_{n+1}^{j}$ is given by:

$$a_\ell(m,j) = -a_\ell(n,j) = \frac{\partial}{\partial x_j} f_\ell(\underline{d}(\underline{v}),\underline{d}(\underline{y}_\ell),\underline{v},\underline{y}_\ell,t) \tag{1.2.4a}$$

$$b_\ell(m) = -b_\ell(n) = -f_\ell(\underline{d}(\underline{v}),\underline{d}(\underline{y}_\ell),\underline{v},\underline{y}_\ell,t) \tag{1.2.4b}$$

if i_ℓ is not an output current.

$$a_\ell(m,\ell) = -a_\ell(n,\ell) = 1 \tag{1.2.4c}$$

$$a_\ell(\ell,j) = \frac{\partial}{\partial x_j} g_\ell(\underline{d}(\underline{v}),\underline{d}(\underline{i}),\underline{v},\underline{i},t) \tag{1.2.4d}$$

$$b_\ell(\ell) = -g_\ell(\underline{d}(\underline{v}),\underline{d}(\underline{i}),\underline{v},\underline{i},t) \tag{1.2.4e}$$

if i_ℓ is an output current

The coefficient $a(i,j)$ and right hand component $b(i)$ are given by:

$a(i,j) \leftarrow a(i,j) + a_\ell(i,j)$ and $b(i) \leftarrow b(i) + b_\ell(i)$,

for $1 \le \ell \le q$. This process is called updating of the matrix entries.

The resulting matrix will be regarded as a structural symmetrical matrix, with a dominant diagonal. The dominant diagonal is assured by a suitable pivoting [1.3].

In case of bipolar circuits, where the transistors are modelled by a Ebers Moll model, these assumptions are (almost) true.

In chapter 6 some further remarks will be made to these points.

1.3 Aspects of complexity and organization

In this section some aspects concerning the complexity and organization of a circuit analysis program will be considered. This will give information at which instances parallel processing will be considered in later chapters.

A circuit analysis program consists of two phases: the setup- and computation phase.

Setup phase. The user specifies his circuit to the analysis

program in its input language. The statements of the input language must be interpreted in order to generate the data structure and the procedures required for the actual execution in the computation phase. To this end for each specified element its model must be retrieved from the element library. Such a model contains the procedures to obtain its contribution given by (1.2.4) to the MNA set of linear equations.
In the computation phase the data structure and the procedures remain unchanged.

Computation phase. Table (1.3.1) gives a summary of the various coefficients and variables which must be computed in one pass through the time loop and one N-R iteration.
In this table the following assumptions are made.
The MNA set of linear equations consists of N equations.
The contribution of an element, ℓ, to the MNA set of linear equations, given by (1.2.4), is calculated by two procedures: vart(ℓ) and varx(ℓ).
vart(ℓ) evaluates the coefficients which depend on the time step and varx(ℓ) evaluates the coefficients which depend on the $\underline{x}$ vector.
Let the sets of indices I_{nd}, $I_{\ell d}$ and I_{nr} denote the indices of the nonlinear dynamical, linear dynamical and nonlinear resistive elements respectively.
Let the sets of components of $\underline{x}$, I_{xd} and I_{xp} denote the components to which the PBD formula is applied and which are predicted respectively.
The already defined set I_{xc} contains the controlled components ($I_{xd} \subseteq I_{xp}$ and $I_{xc} \subseteq I_{xp}$).

Some of the quantities concerning the operations count which are reported in [1.4] are cited.
The number of time steps, denoted by n_t, is about a thousand, even for small circuits (typical value 1000).
The number of required N-R iterations, denoted by n_ℓ, varies from 3 to 4 per time step (typical value 3).
The solution of the linear equations requires about 10-20%, denoted by rlin, of the total execution time necessary to execute one N-R iteration (typical value 15%).
The reported values depend highly on the size and function of the offered circuit and the desired accuracy.

The values given above for n_t and n_ℓ are the justification for the strategy to do a lot of preprocessing to construct an optimal data-structure for the execution of the time and N-R iteration loops.
In the following the preprocessing will be extended to take advantage of a computer configuration with a number of processors operating in parallel.

time loop	1.	determine: $\bar{x}^1_{n+1},\ldots,\bar{x}^k_{n+1}$	$x\in I_{xp}$
	2.	determine: $d^k(x_{n+1})$	$x\in I_{xd}$
	3.	evaluate : $vart(\ell)$	$\ell\in I_{nd}\cup I_{\ell d}$
	4.	update : $b(i),\ a(i,j)$	$i,j\in\{1,2,\ldots,N\}$
N-R iteration	5.	evaluate : $varx(\ell)$	$\ell\in I_{nd}\cup I_{nr}$
	6.	update : $b(i),\ a(i,j)$	$i,j\in\{1,2,\ldots,N\}$
	7.	solve : $A\ \Delta\ \underline{x} = \underline{b}$ $\underline{x}^j_{n+1} \leftarrow \underline{x}^{j+1}_{n+1}$	
	8.	decide : convergence $\underline{x}^c_{n+1} \leftarrow \underline{x}^j_{n+1}$	
time loop	9.	determine: $E^k_{n+1}(h_1,v)$	$v\in I_{xc}$
	10.	determine: $E^m_{n+1}(h_1,v)$	$v\in I_{xc},\ m\in\{k-1,k+1\}$
	11.	solve : (1.1.5) for $h^m(v)$	$v\in I_{xc}$
	12.	determine: h^m	$m\in\{k-1,k,k+1\}$
	13.	determine: $h_{new},\ k_{new}$ $k,h_1,t_{n+1} \leftarrow k_{new},\ h_{new},t_{n+1}+h_{new}$ $\bar{x}^i_n \leftarrow \bar{x}^i_{n+1}$ $\quad 1 \le i \le k+1$ $h_{i+1} \leftarrow h_i$ $\quad 1 \le i \le k+1$	
	14.	decide : $t_{n+1} > t_e$	

Table (1.3.1) One pass through the time loop and Newton-Raphson iteration.

2. PARALLEL PROCESSING MODEL FOR THE CIRCUIT ANALYSIS PROGRAM.

2.0 Introduction

In this chapter the parallel algorithm (program) for the circuit analysis program will be outlined and a parallel computer organization will be stated in order to execute this algorithm (program). The parallel algorithm (program) and parallel computer organization are brought together in the scheduling model. The scheduling model will be used by the system software to accomplish a matching between the offered job and the available computer system.

2.1 Task graph

A problem is solved by some algorithm which is accomplished by a computer program. Consider a computer program, written in Algol, consisting of a sequence of statements which operate on the variables, these will be called global variables. The statements are not allowed to be conditional or for-statements. The algorithm or its computer program will be called the "job" and a statement will be called a "task". A task itself may vary from a simple assignment statement to a block in which all kinds of statements are allowed. A task operates on a subset of the global variables and possibly on a set of local variables which are declared in the program block of the task.
To obtain a correct solution the tasks must be executed according to a certain partial ordering after the global and local variables have been initialized with the proper values. The sequence in which the tasks are given is one of the possible sequences which are allowed by the partial ordering.
If the job is to be executed by a parallel computer the ordering must be made explicit in the program (algorithm), in which case it will be called a "parallel program" (algorithm).

Let $T = \{u_1, u_2, \ldots, u_n\}$ denote the set of n tasks of the considered job. Let the partial ordering of the tasks be given by: $PO = \{(u,v) \mid (u,v \in T) \wedge PO(u,v)\}$, where the relation $PO(u,v)$ is defined by: $PO(u,v) \leftrightarrow$ u must precede v. The job will be represented by a graph (T,S), the "task graph", where $S = \{(u,v) \mid PO(u,v) \vee PO(v,u)\}$. The precedence relation of $(u,v) \in S$ is given by $PO(u,v)$. The task graph together with the precedence relations is denoted by $(T,\vec{S})$.

If PO(u,v), then u will be called a "predecessor task of v" and v will be called a "successor task of u".
A proper parallel algorithm will be represented by a task graph $(T,\vec{S})$ without loops.
A task u will be called "free" if all its predecessor tasks have been executed. If between two tasks $u,v \in T$ does not exist a path in $(T,\vec{S})$ they can be processed in parallel after they have been made free.

Consider the execution of a job with task graph $(T,\vec{S})$ by a fictitious "ideal parallel computer". The ideal parallel computer consists of:
- A large memory, which contains the tasks and all variables. The access to this memory takes no time.
- A set of m identical processors capable of executing any of the tasks. The execution of a task starts with taking a copy of the task and all variables on which it operates. The execution ends by replacing the old values of the copied variables by the new values.
- An operating system which schedules the tasks. The scheduling determines for each task during which time interval and by which processor it will be processed.

Let I_u and O_u denote the set of input and output variables of task $u \in T$.
The sets are defined by:
$I_u = \{x \mid x$ is a global variable to which is referred by any of the statements of task $u\}$
$O_u = \{x \mid x$ is a global variable to which a value is assigned by any of the statements of task $u\}$.
In order to avoid data interleaving, the PO must assure that for any two tasks u and $v \in T$, which are allowed to be processed in parallel holds:

$$((O_u \cap I_v = \phi) \wedge (O_v \cap I_u) = \phi) \wedge (O_u \cap O_v = \phi)) \qquad (2.1.1)$$

The performance of a parallel algorithm depends on the applied parallel computer and the applied scheduling of the tasks. The ideal parallel computer is often used to evaluate the performance. This is, of course, far from realistic; for instance no attention is paid to the memory access at all.

Two performance measures are considered here:

- the total elapsed time or schedule length to process the job on a parallel computer with m processors, to be denoted by $\omega = \omega(m)$,
- the speedup ratio, to be denoted by $SR = SR(m)$, which is given by $SR(m) = \omega(1)/\omega(m)$.

To achieve a fair value of $SR(m)$, $\omega(1)$ must be obtained from the best known sequential algorithm. If no other convention is made the sequential execution of the parallel algorithm will be used to obtain $\omega(1)$. Let $\tau(u)$ denote the required processing time to execute a task $u \in T$, then the length of the critical path in $(T,\vec{S})$, based on the required processing times, gives the minimum achievable ω.

In general different algorithms may be applied to solve a particular problem. The choice of the algorithm to be selected depends on the operations count, weighted by the respective execution times, numerical aspects and the demand for storage. By the introduction of the parallel computer a new aspect has to be taken into consideration. Namely, the partial ordering of the tasks can become more important than the operations count. The behaviour of the function $SR = SR(m)$ for the various algorithms must be compared. Further a degradation of the ω and SR may be expected due to the architecture and its parameters of the actual parallel computer configuration in as much as it deviates from the ideal parallel computer.

The parameters of the parallel computer organization will influence the design of the parallel algorithm (program) which will be used to solve a problem, "the decomposition of the job into tasks".

Parallel algorithms can be obtained in two ways:

- By recognizing the parallelism which is often in a sequential algorithm, called "inherent parallelism". The algorithms in linear algebra contain often a great deal of inherent parallelism.
- By the construction of entirely new algorithms. For instance, linear recurrence systems are transformed into equations on which the recursive doubling technique may be applied [2.1]. However, in [2.2] it is shown that for nonlinear recurrence systems a speedup can only be achieved by the parallel execution of the recurrence equation itself. This result is important because in the circuit analysis program the iteration loops (recurrence equations) contain in general nonlinear functions. Hence, the only speedup which can be achieved is given by the speedup ratio that can be obtained by parallel processing

of the program inside the time and the N-R iteration loop.

Finally, a decomposition for the circuit analysis job will be considered. One pass through the time loop and N-R iteration will be regarded as the job. The decomposition is simply found by inspection of the job, given by table (1.3.1). The resulting task graph is shown in fig. (2.1-1). The labels at the tasks refer to the entries in table (2.3-1). Two empty tasks are added. The task with label 15 indicates the beginning and the task with label 16 is introduced to obtain the same task graph for the 1-th as well as for the j-th N-R iteration ($j > 1$).

If the evaluation of tasks, labeled with the same label, is assumed to take the same time and $\tau(i)$ denotes the required processing time of a task with label i, then the critical path length is given by $\sum_{i=1}^{14} \tau(i)$.

In practice, the number of processors is far smaller than the number of components of the unknown vector $\underline{x}$. Hence ω exceeds the critical path length. If $m << N$ then it is reasonable to suppose that the total required processing time of all tasks with the same label is proportional to $1/m$, except for the tasks with label 7, 8, 12, 13 and 14. The task with label 14, the decision whether to initialize a new time iteration or not, will be left out of consideration because $\tau(14) << \sum_{i=1}^{13} \tau(i)$. The task with label 7 will be considered as a job which is further decomposed into subtasks. In chapter 6 it will be shown that the processing times of the tasks labeled with 8, 12 and 13, are also proportional to $1/m$.

Suppose the required processing time, denoted by ω_{lin}, to solve the set of linear equations on a parallel computer is given by (2.1.2). Let m_0 denote the number of processors where all parallelism of the parallel algorithm is exploited. (Further increase of the number of processors does not decrease ω anymore). The function $\alpha(m)$ denotes how efficient the m processors are used. The $\alpha(1) = 1$ and $\alpha(m)$ is supposed to have a constant value c, for $1<m\leq m_0$. This function is a rough approximation of the functions given in chapter 5.

$$\omega_{lin}(m) = \begin{cases} \omega_{lin}(1) \;/\; (\alpha(m)*m) & \alpha(1) = 1,\ \alpha(m) = c, \text{ for } 1<m<m_0 \\ \omega_{lin}(1) \;/\; (\alpha(m_0) * m_0) & \text{for } m\geq m_0 \end{cases} \qquad (2.1.2)$$

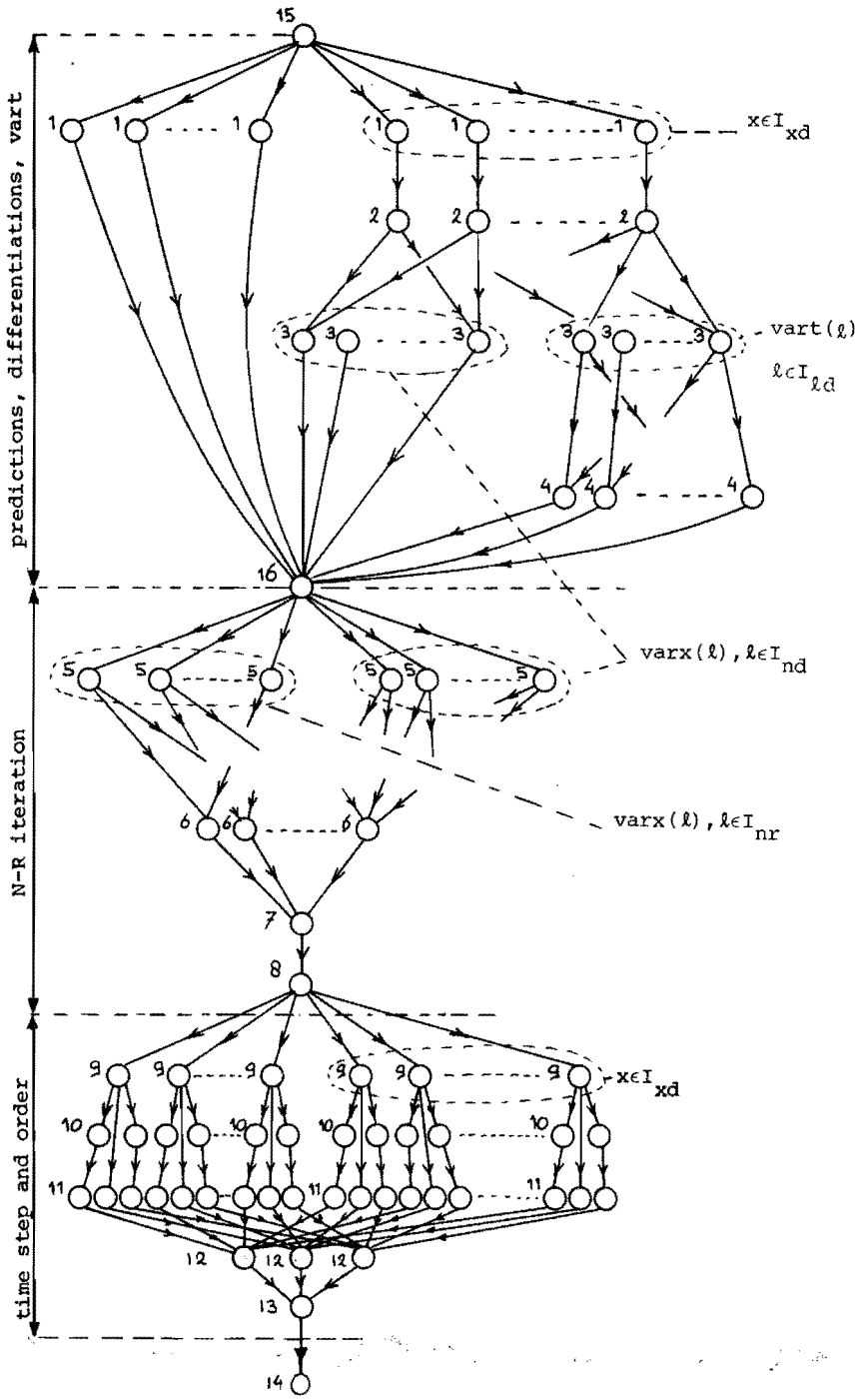


Fig.(2.1-1) Task graph for one pass through time loop and N-R iteration.

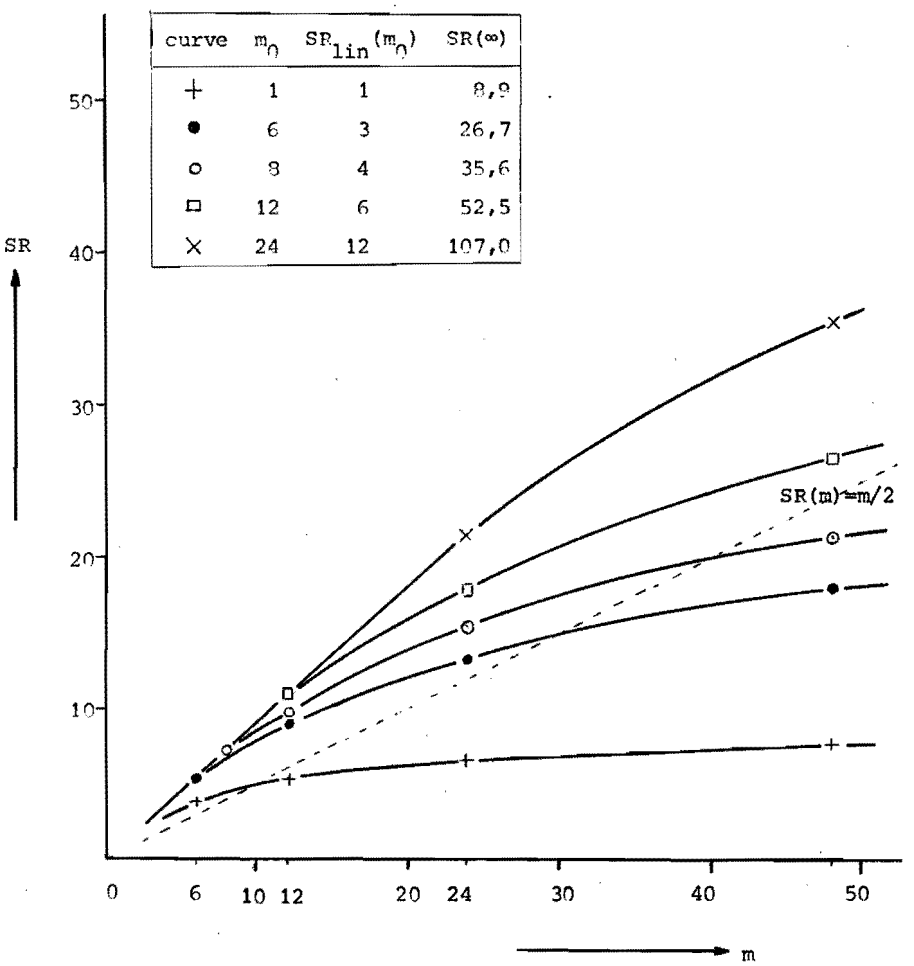


Fig.(2.1-2) Expected speedup as a function of m for the computation phase with formula (2.1.2) for ω_{lin} as parameter.

Let τ_{time} and τ_{NR} denote the time necessary to evaluate the tasks outside the N-R iteration and inside the N-R iteration on a single processor respectively.
Under the assumptions given above and with the notations of section 1.3 the following speedup ratio can be expected:

$$SR = \frac{(\tau_{NR}*n_\ell + \tau_{time})\, n_t}{(\tau_{NR}*n_\ell\ (rlin*(\omega_{lin}(m)/\omega_{lin}(1)) + (1-rlin)/m) + \tau_{time}/m)\, n_t} \quad (2.1.3)$$

Sequential evaluation requires n_t times the execution of the time loop and inside the time loop the N-R iteration must be executed n_ℓ times.
Parallel evaluation reduces the time necessary to evaluate the tasks inside the loops. The processing time to solve the set of linear equations is given by (2.1.2). The processing time of the other tasks is proportional to 1/m.
Fig. (2.3-2) shows the speedup function given by (2.1.3) with $\tau_{time} = \tau_{NR}$ and the typical values given in section 1.3, for five different $\omega_{lin}(m)$ functions.
This analysis serves to establish quantitative estimates of the achievable speedup and the number of required processors. Under the assumption that m << N it is shown that the obtained speedup depends highly on the speedup which can be achieved for the solution of the set of linear equations. Hence, the solution of the set of linear equations by the parallel computer is extremely important.

2.2 The asynchronous array computer

In this section a parallel computer organization which is more realistic than the ideal parallel computer will be defined. Actually the design is completed to so much detail that predictions about the performance are fairly dependable.

The proposed parallel computer, "the asynchronous array computer", consists of a host computer, m computer modules and a connection network, see fig.(2.2-1).
- The host (computer) is a general purpose computer. The host can gain control of each computer module in order to access its memory. The host may be interrupted by the computer modules by a signal on

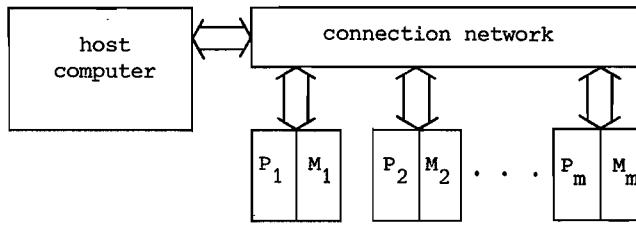


fig.(2.2-1) The asynchronous array computer.

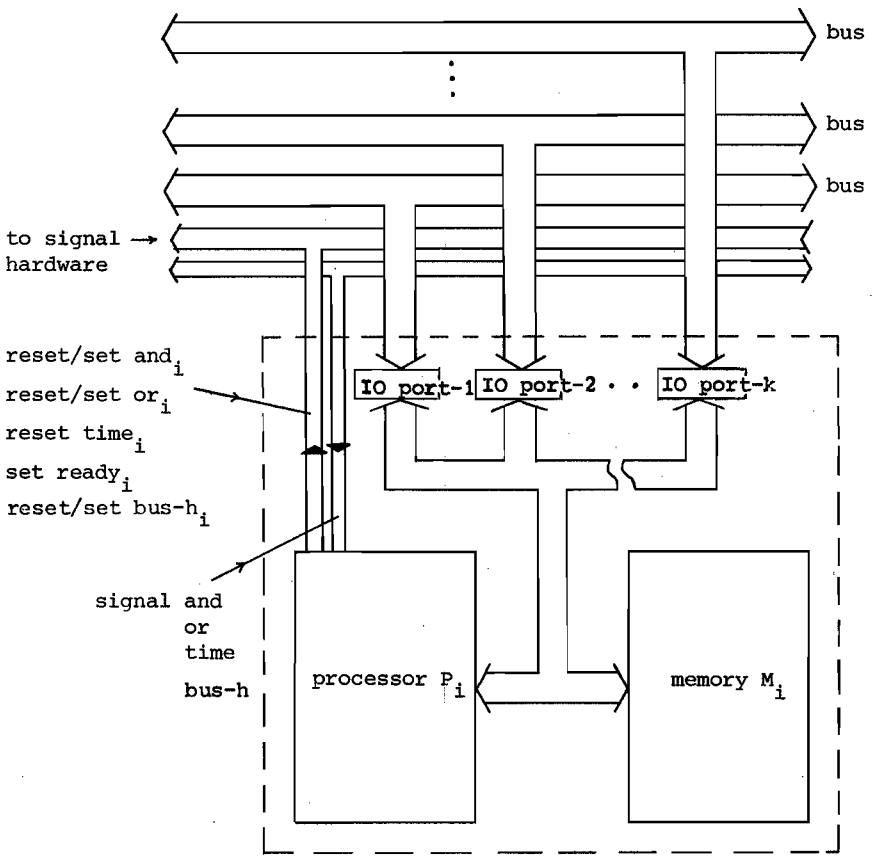


fig.(2.2-2) Computer module CM_i

the line "ready".

- A computer module, CM_i as shown in fig.(2.2-2), consists of a processor, P_i, and a memory M_i, for $i \in \{1,\ldots,m\}$. The computer module is especially equipped to perform floating point operations.

To communicate with the outside world k IO ports are provided. To be concrete let the IO port consist of a simple bidirectional register.

To provide the necessary synchronization facilities the following signal lines are provided: "and", "or", "ready", "time". The and and or signal values are T(rue) or F(alse). The time signal value is a non-negative integer.

The instructions set(signal) and the reset(signal) cause the value of the signal to be T or 0 and F or 0 respectively.

The time signal is provided by a counter which is part of the connection network. A reset instruction starts the counter again with a zero value.

The instruction test(signal,x) operates on the time signal, where x is a non-negative integer which is supplied by the programmer. Execution of a test(signal,x) by a computer module results in active waiting until the value of the signal is larger or equal to the supplied value of x.

The synchronization tools are summarized in table (2.2.1).

signal set of values	supplied to	instructions executed by CM_i	transition caused by:
and signal {T,F}	all CM	set(and) reset(and)	F→T:after all CM have executed the set(and) T→F:after the execution of the reset(and) by any CM
or signal {T,F}	all CM	set(or) reset(or)	F→T:after the execution of the set(or) by any CM T→F:after the execution of the reset(or) by any CM
ready signal {T,F}	host	set(ready)	F→T:after all CM have executed the set(ready) T→F:only possible by the host
time signal {0,1,2...}	all CM	reset(time) test(time,x)	current value→0:after the execution of reset(time) by any CM

Table(2.2.1) Synchronization tools.

The communication is accomplished with the following instructions:

(1) IO port-h ← source

(2) send IO port-h

(3) take IO port-h

(4) destination ← IO port-h

where: $h \in \{1,2,\ldots,k\}$

Only the instructions (2) and (3) need some further explanation. They interact on the bus h and the "bus-h signal" line. The bus-h signal value is either F(alse) or T(rue) for $h \in \{1,\ldots,k\}$.
Instruction (2) connects the register to bus h and makes the bus-h signal T during its execution.
Instruction (3) forces processor P_i to wait until the bus-h signal is T, then the IO port-h is connected to bus-h. The instruction ends with disconnecting IO port-h of bus h.
If the time between two successive broadcast instructions is larger than the time between two successive receive instructions, no further synchronization is necessary.

The connection network connects the m computer modules and the host computer. The connection network consists of k buses: {bus 1,..., bus k}, the signal lines, set and reset lines and also the implementation of the signal functions.
A bus h consists of the data lines which are connected with IO port-h of all computer modules, for $h \in \{1,\ldots,k\}$.

The operating system of the asynchronous array computer resides completely in the host computer. The system software accomplishes the necessary preprocessing before the actual job, the computation phase, is executed by the computer modules. In section 1.3 it was already mentioned that doing a lot of preprocessing is justified.
The system software consists of:

- the conventional setup phase. This part is already described in section 1.3
- the decomposition of the job into tasks. Besides the decomposition resulting into the task graph shown in fig.(2.1-1) in chapter 3 the job which solves the set of linear equations will be decomposed into tasks
- the scheduling of the tasks. This part assigns each task to a processor and determines the time interval during which it has to be

executed. A task is called "assigned" to a processor P_i or a computer CM_i if its instructions are stored in M_i and will be processed by P_i. The scheduling assumes that the necessary processing time of the tasks can be determined in advance. In chapter 6 remarks will be made for the case where this assumption is unrealistic.
In the next section the scheduling model is presented by which the assignment of the tasks and the time intervals will be determined.
- the assembly of the data structure and codes. The necessary synchronization instructions are inserted in the code. To assure that a task will be executed during the determined time interval, say [x,y), it will be preceded by a test(time,x) instruction.
For the computation job this is achieved as follows. By the three sets of labels: {15,1,2,3,4},{16,5,6,7,8} and {9,10,11,12,13,14} three sets of tasks are determined. The necessary synchronization times for tasks of the first two sets are given with respect to the start of the tasks 15 and 16 respectively. The synchronization times for tasks of the third set are given with respect to the end of the task with label 8. For each new time loop or N-R iteration the time signal must be reset.To this purpose at the reference places reset(time) instructions are placed.
- the loading and unloading of the computer modules.
During the computation phase the host computer will be free until all processors signal that they are ready.

Because of the necessary data exchange between different computers over the connection network the set of tasks is extended by communication tasks. The procedures which accomplish the data exchange between computer modules will be considered now.
Let X denote a data set with coefficients x(ix), for ix∈DX, where DX is the set of indices ix of the data set. Assume the array ISX contains the indices of those coefficients of X which must be transmitted. Further, let the array IRX contain the indices of those locations in X in which received coefficients must be stored. In the following in the above notations X will be replaced by the actual name.
Consider two data sets A and B which are allocated to computer module CM_i and CM_j respectively. The array ISA contains the indices of those coefficients of A which must be transmitted to CM_j. The array IRB contains the indices of those locations of B in which the transmitted

coefficients must be stored. After the communication b(IRB(k)) = = a(ISA(k)), for $k \in \{1,\ldots,|ISA|\}$ must hold.
This communication will be accomplished by a procedure "broadcast (A,ISA,h)" and procedure "receive (B,IRB,h)" which are executed by CM_i and CM_j respectively. To assure synchronization they are preceded by a synchronization instruction, test(time,t_x), with t_x the time at which the communication is planned.

```
procedure broadcast(A,IBA,h);
begin
  comment A is allocated to this CM, IB contains the indices of coef-
  ficients to be broadcasted, h is the used bus;
for u = 1 step 1 until |IBA| do
    begin
    IO port-h ← a(IBA(u));
    send IO port-h;
    end;
end;
```

```
procedure receive(B,IRB,h);
begin
  comment B is allocated to this CM, IRB contains the indices of coef-
  ficients to store received data, h is the used bus;
for u = 1  step 1 until |IRB| do
    begin
    take IO port-h;
    b(IRB(u))← IO port-h;
    end;
end;
```

To obtain a correct communication process the number of coefficients in the arrays IBA and IRB must be the same. Further, assume that if IRB(k)$\notin$ DB then the received coefficient will be stored nowhere.
If the received coefficients have to be stored in two different data sets this will be accomplished by an analogous procedure.
Let "procedure receive (B,IRB,C,IRC,k)" denote that the first |IRB| coefficients have to be stored in data set B in the locations given by IRB and the next |IRC| coefficients in data set C in the locations given by IRC.

In order to avoid data interleaving and memory conflicts the following communication rules are given.
A communication task will be considered as an indivisible action. The communicating computer modules are completely devoted to the communication task. The broadcasting computer module will be regarded as master and the receiving computer modules are regarded as slaves.

2.3 Scheduling model

The operating system has to assign the tasks to the computer modules and has to determine time intervals for the processing of each task such that some performance measure is optimized. This optimization problem is called the "scheduling problem". Here the performance measure will be the elapsed time ω to process all tasks.

First a detailed description of the scheduling model based on the model as presented in [2.3], will be given. In chapter 4 a heuristic solution method for a scheduling model derived from the job which solves the set of linear equations will be given. The scheduling model consists mainly of two parts: the resource system and the job system.

- The resource system.

Everything that may be required for the processing of any task is called a "resource". The set of resources establishes the "resource system". The resource system is partitioned into two sets, the set P of processors and the set R of "additional" resources.
Let $P = \{P_1, P_2, \ldots, P_m\}$ be the set of processors. The functional capability and the processing speed of the individual porcessors are not necessarily equal.
Let $R = \{R_1, R_2, \ldots, R_s\}$ be the set of additional resources. For each resource $R_i \in R$ a restricted amount is available, to be denoted by $rm(R_i)$.
Resources in the set R are for instance buses and memories. Some of the resources in the set R may be artificial resources. For instance consider two tasks u and v which are not allowed to be processed at the same time, however the sequence of execution is arbitrary. These tasks may occur in the task graph as tasks to be executed in parallel. In order to avoid parallel processing of u and v an additional resource R_q with amount $rm(R_q) = 1$ will be introduced. The resource demands of tasks u and v are extended by a demand for resource R_q by

an amount of 1. By this parallel execution of the tasks u and v is impossible but the sequence of execution is left free. The condition given by (2.1.2) which avoids data interleaving is not necessary in this case.

- The job system.

The task graph $(T,\vec{S})$ together with the resource demands of each task is called the "job system".

The required processing time and the required additional resources may depend on the assignment of the tasks to the processors. Let

$$F_A : T \rightarrow P \tag{2.3.1a}$$

be a mapping defined by:

$$F_A(u) = P_i \quad \text{for } u \in T \text{ and } P_i \in P \tag{2.3.1b}$$

The required processing time of a task $u \in T$, "the task duration", executed by a processor $P_i \in P$ and with an assignment of the tasks given by F_A will be denoted by: $\tau(u,P_i,F_A)$. Iff all processors are the same then the task duration is independent of P_i. In that case the second argument will be deleted.

The amount of resource $R_q \in R$ which is required by a task $u \in T$ during the entire execution time and with an assignment of the tasks given by F_A will be denoted by: $r(R_q,u,F_A)$. If the task durations and resource demands are independent of the assignment the argument F_A will be dropped in the expressions.

If a task requires more than one processor, one of the processors will be regarded as the master of the other required processors. This situation can be modeled by treating the processors also as additional resources. The set of additional resources becomes $R = \{R_1,\ldots R_s,P_1,\ldots,P_m\}$. The resource amount $rm(P_i) = 1$, for $1 \le i \le m$. The resource demand of a task u for the newly introduced additional resources is given as follows:

$$r(P_i,u,F_A) = \begin{cases} 1 & \text{if } P_i \text{ equal to } F_A(u) \text{ and for all } P_i \text{ which are slaves} \\ & \text{of } F_A(u) \text{ during the execution of task } u \\ 0 & \text{otherwise} \end{cases}$$

The task duration is determined by the master which is given by the mapping F_A.

Once the resource system and the job system are defined the scheduling problem can be stated formally.

Assume the time interval for execution of task $u \in T$ is given by $[\sigma(u), \delta(u))$, where $\sigma(u)$ and $\delta(u)$ denote the time instants of starts and finish of task u respectively ($\sigma(u) \in [\sigma(u), \delta(u))$ and $\delta(u) \notin [\sigma(u), \delta(u))$).

Let T_a denote the time axis: $T_a = [0, \infty)$ and let I denote the set of time intervals: $I = \{[ts, tf) \mid ts \leq tf \text{ and } ts, tf \in T_a\}$. Let

$$F_I : T \rightarrow I \tag{2.3.2a}$$

be a mapping defined by:

$$F_I(u) = [\sigma(u), \delta(u)) \qquad \text{for } u \in T \tag{2.3.2b}$$

A schedule will be defined by F_A and F_I. In defining F_A and F_I certain constraints have to be observed. Before going in more details two more mappings are defined. Let

$$f_p : P \times T_a \rightarrow T \tag{2.3.3a}$$

be a time dependent mapping defined by:

$$f_p(t, P_i) = \{u \in T \mid ((F_A(u) = P_i) \vee (r(P_i, u, F_A) \neq 0)) \wedge (t \in F_I(u))\},$$
$$\text{for } P_i \in P \tag{2.3.3b}$$

The mapping $f_p(t, P_i)$ defines the task which is processed at time t by processor P_i (or coprocessed if P_i is used as a slave). A graphical representation of $f_p(t, P_i)$ is a so called "Gantt chart"[2.4]. Further let

$$f : T_a \rightarrow M(T) \tag{2.3.4a}$$

be a time dependent mapping defined by:

$$f(t) = \bigcup_{P_i \in P} f_p(t, P_i). \tag{2.3.4b}$$

The mapping f(t) defines the set tasks which are processed at time t by any of the processors.

Now a schedule is proper if:

$$\forall_{u,v \in T} [((u,v) \in \vec{S}) \rightarrow \sigma(v) \geq \delta(u)] \tag{2.3.5}$$

that is, the precedence relations are satisfied,

$$\forall_{u \in T} [F_A(u) = P_i \rightarrow \delta(u) \geq \sigma(u) + \tau(u, P_i, F_A)] \tag{2.3.6}$$

that is, any task is given enough processing time,

$$\forall R_i \in R \; \forall t \in T_a \; [\sum_{u \in f(t)} r(R_i, u, F_A) \leq rm(R_i)] \qquad (2.3.7)$$

that is, the allocated additional resources are at any time instant t smaller or equal than their given amount.

The scheduling task is to minimize ω, the schedule length(elapsed time) under the constraints given by (2.3.5), (2.3.6) and (2.3.7).
If the scheduling model contains identical processors, no additional resources and each task $u \in T$ requires only one arbitrary processor for a time interval $\tau(u)$ then it will be referred to as the "basic model". If to a basic model a set of resources R is added and the resource demand for any $R_v \in R$ of each task u is given by $r(R_v, u)$ it will be referred to as an "augmented basic model". If the durations and resource demands are also dependent of F_A the model will be referred to as a "general model".

Further sequencing constraints may be imposed on the schedule. By these constraints the set of proper schedules is partitioned into classes. Two important classes are distinguished: "preemptive" and "nonpreemptive schedules".
- In a preemptive schedule it is allowed to stop the execution of a task on a processor and to postpone the remainder of the task. This is called a "preemption". The remainder of the task may be processed by a different processor and again preemption may occur. If preemption is allowed the task can be split into a chain of smaller tasks.
- In a nonpreemptive schedule each task which is started must run until completion is achieved without interruption.
Other sequencing constraints may also be imposed to limit the number of proper schedules. For instance any task will be started as soon as possible. List schedules, as defined in chapter 4, use this sequencing constraint. Sequencing constraints may reduce the efficiency of the result. Of course, sequencing constraints will be considered only if their impact on the performance is tolerable.
In chapter 4 the finally proposed scheduling will be specified.

3. SOLUTION OF THE LINEAR EQUATIONS

3.0 Introduction

A set of linear equations is given by (3.0.1), where A is a non-singular square matrix of dimension n. To solve equation (3.0.1)

$$A\underline{x} = \underline{b} \tag{3.0.1}$$

direct or indirect methods may be used [3.0].

- The direct methods compute the solution $\underline{x}$ in a fixed number of operations. If no truncation errors are made then the solution is exact.
- The indirect (iterative) methods compute successive approximations to the solution $\underline{x}$. The required number of operations depends on the desired accuracy.

Indirect methods will not be considered here because of possible convergence difficulties. Though the procedure which must be executed for one iteration may result in a highly parallel algorithm [3.1], [3.2], the resulting speedup depends also on the number of required iterations.

The procedures which will be considered, obtain the solution by L\U-decomposition, forward substitution and back substitution. By this solution procedure the matrices may remain sparse.

Matrix A may be expressed as the product of two matrices L and U:

$$A = L\ U \tag{3.0.2}$$

where L is a lower triangular and U an upper triangular matrix. The solution can now be obtained in two steps. The first step solves (3.0.2) for $\underline{c}$, an auxiliary vector, by a forward substitution.

$$L\underline{c} = \underline{b} \tag{3.0.3}$$

The second step solves (3.0.4) for $\underline{x}$ by a back substitution

$$U\underline{x} = \underline{c} \tag{3.0.4}$$

The coefficients of A, U and L are denoted by a(i,j), u(i,j) and $\ell(i,j)$ for $i,j \in \{1,\ldots,n\}$. The coefficients can be determined by the

formulas [3.0] given by (3.0.5).

$$\ell(j,j)\ u(j,j) = a(j,j) - \sum_{k=1}^{j-1} \ell(j,k)\ u(k,j) \qquad 1 \le j \le n$$

$$\ell(i,j) = (a(i,j) - \sum_{k=1}^{j-1} \ell(i,k)\ u(k,j))/u(j,j) \qquad (3.0.5)$$

$$1 \le j < i \le m$$

$$u(j,i) = (a(j,i) - \sum_{k=1}^{j-1} \ell(j,k)\ u(k,i))/\ell(j,j)$$

The determination of the L and U matrices, L\U-decomposition, is accomplished by the procedures "Gauss" or "Crout" which are given below. In both procedures the diagonal coefficients of L are chosen to be 1.

```
procedure Gauss;
begin
for i ← 1 step 1 until n do
    begin
    for each k∈{i+1,...,n} do a(k,i) ← a(k,i)/a(i,i);
    for each (k,h)∈({i+1,...,n})^2 do a(k,h) ← a(k,h) - a(k,i) * a(i,h);
    end; i loop
end; procedure Gauss
```

```
procedure Crout;
begin
for i ← 1 step 1 until n do
    begin
    for each j∈{i,...,n} do
        for k ← 1 step 1 until (i-1) do
            a(i,j) ← a(i,j) - a(i,k) * a(k,j);
        for each j∈{(i+1),...,n} do
            begin
            for k ← 1 step 1 until (i-1) do
                a(j,i) ← a(j,i) - a(j,k) * a(k,i);
        a(j,i) ← a(j,i)/a(i,i);
        end; j loop
    end; i loop
end; procedure Crout
```

The coefficients a(i,j) in the procedures are initially equal to the matrix coefficients a(i,j) of A for $i,j \in \{1,\ldots,n\}$. After the execution of the procedures the coefficients a(i,j) and a(k,m) are equal to the matrix coefficients $\ell(i,j)$ and u(k,m) respectively, for $1 \le j < i \le n$ and $1 \le k \le m \le n$. The coefficients $\ell(i,i)$ are not stored, for $i \in \{1,\ldots,n\}$.

The forward substitution may be done in combination with the L\U-decomposition procedure. Assume coefficient a(i,n+1) is initially equal to component b(i) and after the execution of the L\U-decomposition procedure the coefficient is equal to component c(i), for $i \in \{1,\ldots,n\}$. In the Gauss procedure the index set at line 6 from which the indices k and h are chosen must be replaced by $\{i+1,\ldots,n\} \times \{i+1,\ldots,n,n+1\}$. In the Crout procedure the index set at line 5 must be extended by the index n+1.
The back substitution is performed by the following procedure backsolve.

```
1. procedure backsolve;
2. begin
3. for i ← n step -1 until 1 do
4.     begin
5.     a(i,n+1) ← a(i,n+1)/a(i,i);
6.     for each j∈{1,...,(i-1)} do
7.         a(j,n+1) ← a(j,n+1) - a(j,i) * a(i,n+1);
8.     end; i loop
9.end; procedure backsolve.
```

This procedure backsolve assumes that the coefficients of L and $\underline{c}$ are given by a(i,j) and a(i,n+1) respectively, for $i \in \{1,\ldots,n\}$ and $j \in \{i,\ldots,n\}$.
The number of required operations (multiplications, additions, substractions and divisions) to perform the L\U-decomposition is approximately $2n^3/3-n^2/2+n/2$ for both methods. The number of required operations for the forward substitution and back substitution is n(n-1) and n^2 respectively. In order to solve $A\underline{x} = \underline{b}$ for full matrices $2n^3/3+3n^2/2-n/2$ operations are required, thus $O(n^3)$.

In general the numerical stability of the method depends on the given ordering of the linear equations and the components of the solution vector $\underline{x}$. To assure a numerically stable solution a pivot

strategy may be necessary.

After having introduced general aspects in the next section 3.1 the solution process will be considered for sets of equations where A is sparse. A graph model will be introduced to describe the structure of the matrices during the decomposition process. The graph model is useful to study pivoting.
In section 3.2 the parallel processing of the solution process will be studied. With the aid of the graph structure a parallel algorithm is constructed. The properties of the data structure (e-tree) which guides this algorithm are considered.
In section 3.4 a block decomposition method is described which brings the matrix into the doubly bordered block diagonal form [3.2].

3.1 Sparse matrices and their associated graph model

The set of linear equations which results from a circuit analysis program has in general the following characteristics:
- the number of equations may be very large (about thousand);
- the average number of nonzero matrix coefficients per row is much smaller than the dimension of the matrix.

Matrices which exhibit the last property are called "sparse matrices". Further, due to the choice of the MNA method used to formulate the circuit analysis equations the set of linear equations is assumed to have the following properties:
- the matrices are structurally symmetric, thus $a(i,j) \neq 0 \leftrightarrow a(j,i) \neq 0$.
- numerical stability is assured if only diagonal coefficients of A are regarded as pivot candidates.

The sparsity of the matrices requires an efficient data structure and special attention as to the pivoting, because a straightforward implementation of the ordinary solution procedures would result into a huge amount of storage locations containing zeros and a lot of operations on zero value coefficients. An adequate sparse data structure has proved to be a "row-pointer-column-index"-structure such as given in fig.(3.1-1), which illustrates the corresponding locations of various matrix coefficients in the sparse matrix.
The number of required operations to solve the linear equations depends of course on the degree of sparsity. During the solution procedure coefficients which are initially zero may become nonzero

coefficients; such coefficients are called "fill-ins".The generation of fill-ins depends on the pivot sequence which is chosen during the solution procedure. If no special attention is paid to reducing the number of fill-ins, the resulting decomposition matrices L and U may be non sparse.

	1	2	3	4	5	6	7	8	9	10
1	a(1,1)	a(1,2)	a(1,3)							
2	a(2,1)	a(2,2)	a(2,3)	a(2,4)	a(2,5)					
3	a(3,1)	a(3,2)	a(3,3)	a(3,4)	a(3,5)					
4		a(4,2)	a(4,3)	a(4,4)	a(4,5)	a(4,6)	a(4,7)	a(4,8)		
5		a(5,2)	a(5,3)	a(5,4)	a(5,5)	a(5,6)			a(5,9)	a(5,10)
6				a(6,4)	a(6,5)	a(6,6)		a(6,8)		a(6,10)
7				a(7,4)			a(7,7)	a(7,8)		
8				a(8,4)		a(8,6)	a(8,7)	a(8,8)		
9					a(9,5)				a(9,9)	a(9,10)
10					a(10,5)	a(10,6)			a(10,9)	a(10,10)

array index	1	2	3	4	5	6	7	8	9	10	11
row pointer	1	3	6	8	12						
column index	2	3	3	4	5	4	5	5	6	7	8
upper off. diag.coeff.	a(1,2)	a(1,3)	a(2,3)	a(2,4)	a(2,5)	a(3,4)	a(3,5)	a(4,5)	a(4,6)	a(4,7)	a(4,8)
lower off. diag.coeff.	a(2,1)	a(3,1)	a(2,2)	a(4,2)	a(5,2)	a(4,3)	a(5,3)	a(5,4)	a(6,4)	a(7,4)	a(8,4)
diag. coefficients	a(1,1)	a(2,2)	a(3,3)	a(4,4)	a(5,5)	a(6,6)	a(7,7)	a(8,8)	a(9,9)	a(10,10)	

Fig.(3.1-1) Illustration of a sparse data structure for a structurally symmetric matrix

The matrix A and its data structure can be associated with a graph (V,E) [3.3], with $|V| = n$ and $|E|$ equals the number of nonzero off-diagonal pairs. The definitions of the graph notations are given in appendix Graph notations. Assume a bijective mapping $g : \{a(1,1), a(2,2), \ldots, a(n,n)\} \to V$. This mapping associates with each diagonal coefficient a(i,i) a vertex g(a(i,i)). To each pair of non-zero off-diagonal coefficients a(i,j) and a(j,i) corresponds an edge $(g(a(i,i)), g(a(j,j))) \in E$. The graph (V,E) is called the "associated graph" of matrix A.

The sets inc(v,E), adj(v,E) and def(v,E) can easily be given a meaning

in terms of sets of matrix coefficients. If $g^{-1}(v)$ is some diagonal coefficient then the set inc(v,E) corresponds to the nonzero off-diagonal coefficients in the respective row and column. The set adj(v,E) identifies a set of diagonal coefficients which determines a submatrix of A. This submatrix is associated with the subgraph (adj(v,E), E(adj(v,E))). The zero off-diagonal coefficients in this submatrix are identified by def(v,E).

In order to solve the set of linear equations by Gaussian elimination a pivot sequence must be selected. Only diagonal coefficients are considered as pivots. After reordering the rows and columns of A in accordance with the selected pivot sequence a matrix A' is obtained, $A' = PAP^T$, where P is a permutation matrix. In the associated graph structure this amounts to a reordering of vertices. Assume the special ordering α of (V,E) such that $\alpha^{-1}(g(a(i,i))) = i$ for $i \in \{1,2,\ldots,n\}$, this ordering corresponds to the initial matrix A.If the graph (V,E) is ordered by β the corresponding matrix A' is given by:
$A' = [a'(i,j)] = [a(\alpha^{-1}(\beta(i)), \alpha^{-1}(\beta(j)))]$. It is clear that the associated graph (V,E) represents the class of matrices PAP^T where P is any permutation matrix.

Consider now the L\U-decomposition using the Gaussian process. Assume pivot a(p,p) is selected. The pivot-row is converted into a row of the U-matrix and the pivot-column which is divided by a(p,p) is converted into a column of the L-matrix. The matrix A without this pivot row and -column is used for selecting further pivots.This matrix is updated by the dyadic product of the L-matrix column and U-matrix row. In general this causes fill-ins.
In the graph model two graphs $(\bar{V},\bar{E})$ and $(\hat{V},\hat{E})$ are introduced to account for this process. The graph $(\bar{V},\bar{E})$ corresponds to the L\U matrix constructed so far and $(\hat{V},\hat{E})$, the "elimination graph" corresponds to the remaining matrix. Before the first elimination step $(\hat{V},\hat{E}) = (V,E)$ and $(\bar{V},\bar{E}) = (\phi,\phi)$. The structural updating of these graphs induced by the associated Gaussian elimination step can be described as follows:

```
1. procedure update(u);
2. begin
3. V̄ ← V̄ ∪{u} ∪adj(u,Ê); Ē ← Ē ∪inc(u,Ê);
4. V̂ ← V̂\{u}; Ê ← (Ê ∪ def(u,Ê))\inc(u,Ê);
5. end;
```

The set def$(u,\hat{E})$, the set of "fill-in-edges", represents the fill-ins which are generated.

The L\U-decomposition, according to the Gaussian elimination scheme, which takes account of the sparsity, is formally described by "procedure Gauss". It is assumed that some ordering β has been defined prior to the execution of "procedure Gauss".

```
procedure Gauss;
begin
(V̂,Ê) ← (V,E); (V̄,Ē) ← (φ,φ);
for i ← 1 step 1 until n do
    begin
    u ← β(i); ℓ ← α^-1(u);
    for each v ∈ adj(u,Ê) do
        begin
        j ← α^-1(v);
        a(j,ℓ) ← a(j,ℓ)/a(ℓ,ℓ);
        for each w ∈ adj(u,Ê) do
            begin
            k ← α^-1(w);
            a(j,k) ← a(j,k) - a(j,ℓ)*a(ℓ,k);
            end;
        end;
    update(u);
    end; i loop
end; procedure Gauss
```

Similarly, the Crout decomposition can be described.

```
procedure Crout;
begin
(V̂,Ê) ← (V,E) ; (V̄,Ē) ← (φ,φ);
for i ← 1 step 1 until n do
    begin
    u ← β(i) ; ℓ ← α^-1(u);
    for each v ∈ adj(u,E) ∪ {u} do
        begin
        k ← α^-1(v);
        for all w ∈ adj(v,Ē) ∩ adj(u,Ē) do
```

```
            begin
            m ← α^-1(w);
            a(ℓ,k) ← a(ℓ,k) - a(ℓ,m) * a(m,k);
            end;
        end;
    for each v ∈ adj(u,Ê) do
        begin
        k ← α^-1(v);
        for all w ∈ adj(v,Ē) ∩ adj(u,Ē) do
            begin
            m ← α^-1(w);
            a(k,ℓ) ← a(k,ℓ) - a(k,m) * a(m,ℓ);
            a(k,ℓ) ← a(k,ℓ) / a(ℓ,ℓ);
            end;
        end;
    update(u);
    end; i loop
end; procedure Crout.
```

An ordering with the property that for $i = 1,\ldots,n$ always $def(\beta(i),E) = \phi$ during execution of procedure Gauss or Crout is a "perfect ordering". A graph (V,E) permitting such a perfect ordering is called a "perfect elimination graph". Rose [3.4],[3.5] has proved the equivalence between a perfect elimination graph and a triangulated graph in the sense of Berge [3.6]. Hence the pivoting problem is equivalent to finding a suitable triangulation of the associated graph (V,E).

Two objectives are possible: either to find a minimum triangulation or a triangulation which results in a minimum number of required arithmetic operations for the L\U-decomposition.

Heuristic strategies [3.7] have been developed such as Berry's criterion [3.8], which 'minimizes' the triangulation and the minimum degree criterion [3.9] which 'minimizes' the number of required operations.

Rose proved also various properties of triangulated graphs. Some which are important for the parallel algorithm are stated below.

Lemma 1 [3.4]

[(V,E) is triangulated] $\leftrightarrow$ [$\forall_{u,v \in V}$ [any minimal u,v-separator is a clique]].

Lemma 2 [3.4]

Let (V,E) be triangulated. Then V can be partitioned into two disjoint subsets V_1 and V_2 ($V_1 \cup V_2 = V$, $V_1 \cap V_2 = \phi$) such that

- $\forall_{v \in V_1}$ [def(v,E) = ϕ]
- $\forall_{v \in V_2}$ [v is in some minimal u,w-separation clique].

Lemma 3 [3.4]

In any triangulated graph (V,E) there is at least one vertex $v \in V$ such that def(v,E) = ϕ.

Lemma 4 [3.5]

Let (V,E) be triangulated and let $S \subset V$ be some separation clique. Let $D_i \subset V$, $i=1,\ldots,k$ be the components with respect to S. Then in any component there is at least one vertex $v_i \in D_i$ such that def(v_i,E) = ϕ.

Corollary 1

Assume a triangulated, connected graph (V,E) where V is not a clique. Then there must be at least one minimal u,v-separation clique.

proof:
Since V is not a clique there are at least two nonadjacent vertices u and v. Then any chain between u and v is of length >2 and can be broken by removing one of its "inner" vertices. Select a minimal set of vertices such that S is a minimal u,v-separator. Since (V,E) is triangulated S is a minimal u,v-separation clique. (end of proof).

Lemma 5

Assume a triangulated, connected graph (V,E). Consider $v \in V$ such that def(v,E) = ϕ. Then ($V \setminus \{v\}$, $E(V \setminus \{v\})$) is triangulated and connected.

proof:
The statement follows from the fact that adj(v,E) is a clique. (end of proof).

Lemma 6 [3.4]

For some triangulated graph (V,E) all perfect orderings are equivalent in that they result in the same number of multiplications, additions, substractions and divisions for the L\U-decomposition process.

Assume β is some perfect ordering. The operations count for the numerical operations stated in lemma 6 is given by (3.1.1). Hence the operations count

$$\sum_{i=1}^{n} (2 \mid madj(\beta(i)) \mid^2 + \mid madj(\beta(i)) \mid) \qquad (3.1.1)$$

is of order $O(n\, d_{gem}^{\;2})$ where $d_{gem}^{\;2} = \frac{1}{n} \sum_{i=1}^{n} \mid madj(\beta(i)) \mid^2$.

The essence of the above statements can be summarized as follows. Once a triangulated graph is obtained by the insertion of the necessary fill-in-edges many other perfect orderings can be obtained. For this purpose it is only necessary to select as pivots at any instant such vertices exhibiting zero deficiency. All orderings obtained by this principle will be equivalent in that they all result in the same number of numerical operations for the L\U-decomposition.

3.2 The parallel solution of linear equations

Csanky [3.10] presented an algorithm which requires $O(\log^2 n)$ parallel operations to evaluate A^{-1} with $O(n^4/\log n)$ processors. The solution $\underline{x}$ can also be obtained by $O(\log^2 n)$ parallel operations. This algorithm is only of theoretical value because of the sensitivity for rounding errors and the large amount of processors which are required (For instance if n = 8, approximately one thousand processors are required to reach the critical path length). The necessary communication between this huge amount of processors may cause a severe degradation of the potential parallellism.

To obtain parallel algorithms which are numerically stable only a standard solution method, L\U-decomposition (Gauss) followed by forward- and backsubstitution, will be considered.

Fig.(3.2-1a) shows a matrix which will be considered. Fig.(3.2-1b) shows the sequence of operations which will be executed by the Gauss procedure to decompose the given matrix. The operations are numbered according to their order. Each operation will be considered to be a

task. The set of tasks is : $T = \{u_1,\ldots,u_{20}\}$. The sets O_j and I_j denote the set of output variables of task u_j and the set of input variables of u_j respectively, for $1 \le j \le 20$. The edges of task graph $(T,\vec{S})$ are given by:

$$(u_j,u_i) \in \vec{S} \text{ iff } [p \in I_i \wedge j=\max(\{\ell \mid p \in O_\ell \wedge \ell < i\})], \text{ for } 1 \le i \le 20$$

order	operation or task
1.	a(2,1) ← a(2,1) / a(1,1)
2.	a(3,1) ← a(3,1) / a(1,1)
3.	a(4,1) ← a(4,1) / a(1,1)
4.	a(2,2) ← a(2,2) - a(2,1) x a(1,2)
5.	a(2,3) ← a(2,3) - a(2,1) x a(1,3)
6.	a(2,4) ← a(2,4) - a(2,1) x a(1,4)
7.	a(3,2) ← a(3,2) - a(3,1) x a(1,2)
8.	a(3,3) ← a(3,3) - a(3,1) x a(1,3)
9.	a(3,4) ← a(3,4) - a(3,1) x a(1,4)
10.	a(4,2) ← a(4,2) - a(4,1) x a(1,2)
11.	a(4,3) ← a(4,3) - a(4,1) x a(1,3)
12.	a(4,4) ← a(4,4) - a(4,1) x a(1,4)
13.	a(3,2) ← a(3,2) / a(2,2)
14.	a(4,2) ← a(4,2) / a(2,2)
15.	a(3,3) ← a(3,3) - a(3,2) x a(2,3)
16.	a(3,4) ← a(3,4) - a(3,2) x a(2,4)
17.	a(4,3) ← a(4,3) - a(4,2) x a(2,3)
18.	a(4,4) ← a(4,4) - a(4,2) x a(2,4)
19.	a(4,3) ← a(4,3) / a(3,3)
20.	a(4,4) ← a(4,4) - a(4,3) x a(3,4)

"coefficient update operation" or short update.

Fig.(3.2-1b) Sequence of operations executed by the Gauss procedure.

$$A = \begin{bmatrix} a(1,1) & a(1,2) & a(1,3) & a(1,4) \\ a(2,1) & a(2,2) & a(2,3) & a(2,4) \\ a(3,1) & a(3,2) & a(3,3) & a(3,4) \\ a(4,1) & a(4,2) & a(4,3) & a(4,4) \end{bmatrix}$$

Fig.(3.2-1a) The matrix which serves the example.

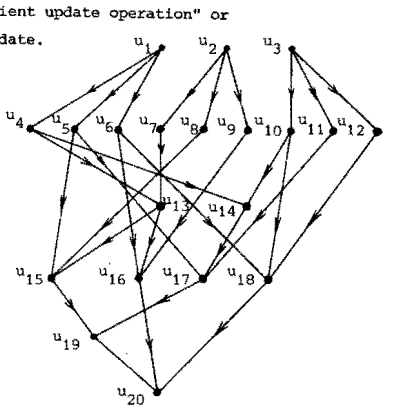


Fig.(3.2-1c) Task graph for the example.

The resulting task graph is shown in fig.(3.2-1c). The task graph may be executed in six parallel operations with 9 processors. In general $2(n-1)$ parallel operations executed by $(n-1)^2$ processors are required to decompose a matrix with dimension n.

Two possible implementations of the algorithm on the asynchronous array computer will be presented.
A straight forward implementation [3.11] is obtained if the processors are working synchronously. To this end it is assumed that each task requires the same amount of time to be executed. The time axis is divided in time intervals, "time-slots", with equal length. The

scheduler assigns each task to a processor and to a time-slot. (This can be accomplished by a list schedule; see chapter 4). Each computer module is loaded with a copy of the matrix and the tasks assigned to it, together with the time-slot number.

A processor can be in one of two states; the execution state and the communication state. At any time instant the state of all processors is the same. During the i-th execution state a processor will execute the task with time-slot i or if none, it will be idle. During the i-th communication state each processor broadcasts in turn the matrix coefficient which was computed during the i-th execution state, while all others receive this coefficient and store it.

Assume a task requires τ time units and to broadcast a single coefficient requires τ_b time units. The actual time necessary to execute the i-th time slot will be increased by $M * \tau_b$ if M coeffcients have to be broadcasted.This causes a severe degradation of the potential speedup.

The implementation given in [3.12] operates also synchronously. In its simplest form the number of processors is equal to the dimension of the matrix. In computer module CM_i the i-th row of the matrix is stored.

Assume the first (i-1) U-rows and L-columns have been determined. During the next communication state CM_i broadcasts a(i,i) to CM_{i+1}, $CM_{i+2},\ldots,CM_n$ which will receive this coefficient. During the next execution state CM_{i+1}, $CM_{i+2},\ldots,CM_n$ compute the new L-column.During the next communication states CM_i broadcasts the remaining coefficients a(i,i+1),...,a(i,n) while $CM_{i+1},\ldots,CM_n$ receive these. After a(i,j) is received by CM_k the operation $a(k,j) \leftarrow a(k,j)-a(k,i)*a(i,j)$ is performed during the execution state, for $k,j \in \{i+1,\ldots,n\}$. Fig. (3.2-2) shows the resulting task graph, where the tasks u_{21} - u_{29} are communication tasks.

The resulting task graph can be derived from the task graph given in fig.(3.2-2) by imposing the sequence constraints due to the row wise organization on the task graph of fig.(3.2-1c) and insertion of the broadcast tasks.

In the first implementation method the required communication will decrease the potential speedup. Improvement is possible if asynchronous operation is allowed, because not all processors need the same coefficients. The communication may be spread.However, the

number of tasks which must be scheduled is of the same order as the number of numerical operations.
In the second implementation the number of active processors during the computation states may be reduced if the matrix is sparse.

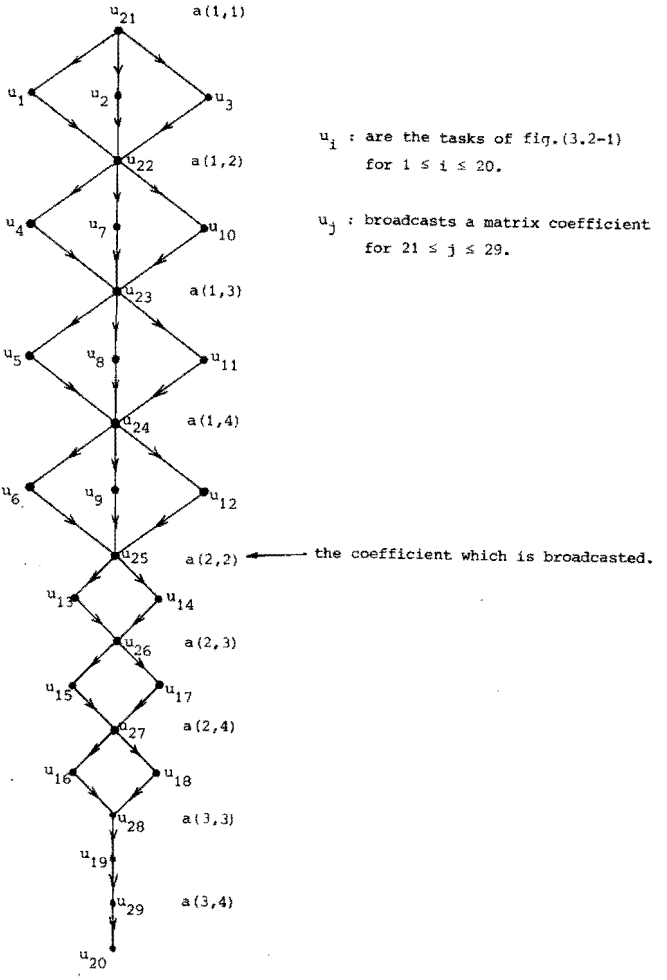


Fig.(3.2-2) Task graph for one row stored per computer module.

3.3 The elimination tree

3.3.1 Strategy to order vertices

In the previous section attention was paid to the parallelism which is in the L\U-decomposition procedure itself, the inherent parallelism. Here the attention will be focussed on the observation that in a sparse matrix the computations which are associated with certain pivots can

be executed simultaneously. These associated computations depend on the applied procedure. The Gauss procedure will be used for this purpose. Later it will become apparent that the derived results also hold for the Crout procedure. The computations which are associated with a pivot are given by the compound statement in lines 5-18 of the Gauss procedure in section 3.1.
The parallelism due to the sparsity has been studied by various authors [3.13], [3.14]. In fact methods [3.15]-[3.19] known as "tearing" and decomposition into "bordered block diagonal form" or "bordered block triangular form" are in close relation to this item.
The inherent parallelism of the L\U-decomposition is accounted for by assuming that whatever work is involved by processing one pivot can be done is a fixed time-slot. In fact the parallel algorithm is developed for the ideal parallel computer model where each processor itself is again a parallel computer, able to exploit all parallelism associated with a pivot. This processor model is realistic to some extend because the computer modules may be equiped with several arithmetic units. The length of the time-slot is equal to the time necessary to accomplish a divide and a coefficient update operation (coefficient update operation $a \leftarrow a - b * c$).

The parallel algorithm will be developed by applying the associated graph (V,E) of the matrix.
Parallel processing of two pivots a(i,i) and a(j,j) with $\alpha(i) \in adj(\alpha(j),E)$ is impossible. Namely if a(i,i) has been chosen as a pivot than a(j,j) can be a pivot only after the updating step $a(j,j) \leftarrow a(j,j) - a(i,j) * a(j,i) / a(i,i)$ has been completed. During the updating memory conflicts may occur if the two submatrices indicated by $adj(\alpha(i),E)$ and $adj(\alpha(j),E)$ have coefficients in common. However, this will be allowed because it is possible to store the dyadic product temperarily.

A method to parallel processing, described in graph terms may be as follows. A mapping $\gamma: V \rightarrow \{1,2,\ldots,k\}$ assigns to each vertex a "label", with $k \leq n$. On basis of these labels "label classes" are defined to be denoted by $X_i = \{x \in V \mid \gamma(x) = i\}$. A label class X_i will contain vertices which can be processed in parallel after all vertices of the previous classes have been processed. Procedure "classwise L\U-decomposition" describes symbolically the decomposition on the basis of these classes. The operations implied by line 7 may be executed in parallel.

```
1. procedure classwise L\U-decomposition;
2. begin
3. (V̂,Ê) ← (V,E); (V̄,Ē) ← (φ,φ);
4. for i ← 1 step 1 until k do
5.     begin
6.     X ← {x ∈ V | γ(x) = i};
7.     for each u ∈ X do update(u);
8.     end;
9. end;
```

The graphs $(\hat{V},\hat{E})$ and $(\bar{V},\bar{E})$ constructed by the compound statement in lines 5-8 after the i-th turn will be denoted by $(\hat{V}_i\hat{E}_i)$ and $(\bar{V}_i,\bar{E}_i)$. The graphs $(\hat{V},\hat{E})$ and $(\bar{V},\bar{E})$ in line 3 are denoted by $(\hat{V}_0,\hat{E}_0)$ and $(\bar{V}_0,\bar{E}_0)$. The set X in line 6 corresponds to the already defined set X_i.

Now the label class X_1 may be constructed as follows. Initialize $X_1 \leftarrow \phi$. Each vertex $v \in \hat{V}_0$ will be inspected. If $adj(v,\hat{E}_0) \cap X_1 = \phi$ then v will be inserted into X_1. After completing X_1 all its vertices are eliminated from (V_0,E_0) to obtain (V_1,E_1). The whole process is repeated until all vertices are inserted into a label class. The last label class is assumed to be X_k.

However, this strategy to construct the label classes may cause an enormous amount fill-in edges. This may result in a demand for an unacceptable number of storage locations and also the operations count may be extremely large because the operation count is given by $O(nd^2_{gem})$.

The number of operations must be limited because in practice the computer modules of the asynchronous array computer will have a limited processing power. To deal with this fact it will be assumed that during preprocessing a pivot sequence was constructed which minimized the fill-ins and or the number of operations. The associated graph (V,E) of the matrix in which the fill-ins are inserted is a triangulated graph. If in the above strategy during the selection of label class X_i only vertices v will be labeled i with $def(v,\hat{E}_{i-1}) = \phi$ then no new fill-ins are introduced. Hence the number of fill-ins and the operations count of the L\U-decomposition is determined by the applied pivot criterion during preprocessing.

Procedure "e-tree" accomplishes the above given strategy to label the vertices and constructs a graph (V,B) on basis of these labels.

The graph (V,B), which is defined by the procedure e-tree, will be called "elimination-tree" ("e-tree").In the next sections it will be proved that (V,B) is a spanning tree of (V,E) where the edges together with the labels represent the partial ordering by which the vertices must be eliminated to obtain a perfect elimination graph. In [3.20] a formal definition is given of an e-tree followed by a procedure to obtain an e-tree. The operational definition of the e-tree is used here because it is more in accordance with the strategy to label the vertices.

```
procedure e-tree;
begin
comment (V,E) is supposed to be triangulated;
(V̂,Ê) ← (V,E); i ← 1;
while V̂ ≠ ϕ do
    begin
    U ← V̂;
    while U ≠ ϕ do
        begin
        pick some v ∈ U; U ← U\{v};
        if def(v,Ê) = ϕ then
            begin
            γ(v) ← i;
            comment γ(v) is the label of v;
            U ← U\adj(v,Ê);
            V̂ ← V̂\{v}; Ê ← Ê\inc(v,Ê);
            end;
        end;
    i ← i+1;
    end;
B ← ϕ;
comment (V,B) will be the e-tree;
for each v ∈ V do
    begin
    ℓ(v) ← {γ(w) | (v,w) ∈ E ∧ γ(w) > γ(v)};
    B ← B ∪ {(v,w) ∈ E | γ(w) = min(ℓ(v))};
    end;
end;
```

3.3.2 Procedure e-tree

The algorithm generates a graph (V,B). By proving the following lemma's it is shown that (V,B) is a spanning tree of (V,E).

Lemma 7

During the execution of procedure e-tree on a connected triangulated graph (V,E) any $v \in V$ is assigned to exactly one label-class.

proof:

By lemma 3 there will be a nonempty subset $X_1 \subset V$ given label "1" in line 12 of the algorithm. Any time a vertex is assigned to X_1 it is removed from the graph in line 14, this it cannot be assigned to any other label class. Arriving at line 18 $(\hat{V},\hat{E}) = (V\backslash X_1, E(V\backslash X_1))$ with $\hat{V} \subset V$ is obtained. $(\hat{V},\hat{E})$ is again connected and triangulated (lemma 5). Thus the procedure can continue with nonempty sets $X_2, X_3, \ldots, X_k$ (lemma 3) until $\hat{V} = V\backslash(X_1 \cup X_2 \cup \ldots \cup X_k)$ is empty meaning that all vertices have been labeled. (end of proof)

With respect to line 23 of procedure e-tree w is called the "successor of v" and v a "predecessor of w".

Lemma 8

Any vertex in some label class X_i, $i<k$, has at least one successor.

proof:

For $i=0,\ldots,k-1$ $(\hat{V}_i,\hat{E}_i)$ is triangulated, connected and nonempty. Thus for $v \in X_i$, $i<k$, the set $adj(v,\hat{E}_{i-1})$ is nonempty. Since all these vertices will be labeled in later executions of line 12 one of them must be a successor to v. (end of proof)

The lemma implies that only the vertices in X_k may have no successor. In fact they will have no successor since for some $v \in X_k$ $adj(v,\hat{E}_{k-1}) = \phi$.

Lemma 9

All vertices of a clique X will be assigned to different label classes.

proof:

Assume $x \in X$ is the first vertex of the clique which receives a label, say i. Then by line 13 of procedure e-tree $X\backslash\{x\}$ is deleted from U which prevents that any vertex $y \in X\backslash\{x\}$ will get the same label i. As

$X\backslash\{x\}$ is again a clique the same reasoning will hold. (end of proof)

Lemma 10

For any vertex $v \in V$ in a connected triangulated graph (V,E) there is at most one successor.

proof:

For $v \in X_i$ the successor (if it exists) will be in the vertex set $adj(v,\hat{E}_{i-1})$. Since this set must be a clique no two vertices from this set will be in the same label class (lemma 9). Since the set is finite there is exactly one vertex $w \in adj(v,\hat{E}_{i-1})$ with smallest label $\gamma(w) > \gamma(v)$ assigned as the unique successor to v. (end of proof)

Lemma 11

X_k contains exactly one vertex.

proof:

Assuming that $(\hat{V}_k,\hat{E}_k) = (\phi,\phi)$ implies $\hat{V}_{k-1} = X_k$. But $(\hat{V}_{k-1},\hat{E}_{k-1})$ is connected. Suppose there is more than one vertex in X_k then because of the connectedness there must be at least two vertices $v,w \in X_k$ such that $w \in adj(v,\hat{E}_{k-1})$. This means that v and w cannot be in the same label class and consequently $\hat{V}_k \neq \phi$ contrary to the assumption. The single vertex $r \in X_k$ is called the "root". (end of proof)

Theorem 1

The graph (V,B) generated by procedure e-tree executed on a connected triangulated graph (V,E) is a spanning tree.

proof:

Any vertex except the root is assigned one edge connecting it with its unique successor. Along those edges a chain can be established from any vertex to the root. That means (V,B) is connected. Since for any vertex except the root there is a unique successor $|B| = |V| - 1$ is obtained. Thus (V,B) is connected and has $|V| - 1$ edges implying that it is a spanning tree. (end of proof)

3.3.3 Properties of the e-tree

Assume (as indicated earlier) that all pivots in the same label class can be processed in parallel and that the processing of one pivot takes one fixed time slot. Then the critical path through the e-tree

indicates the number of time slots necessary to complete the L\U-decomposition.

The following lemmas and theorems have the purpose to show that given some triangulated connected graph (V,E)

- all orderings β such that $\forall_{x,y \in V}$ [x is a successor of y $\rightarrow$ $\rightarrow \beta^{-1}(x) > \beta^{-1}(y)$] are perfect orderings;
- the length of the critical path in some e-tree is identical with the label of the root, $\gamma(r)$;
- all possible e-trees for a given graph (V,E) exhibit critical paths of identical length.

Lemma 12

Let $v \in X_i$ and $w \in adj(v,\hat{E}_{i-1})$. Then w is a vertex in the chain from v to r in (V,B).

proof:

The proof works by induction on the label classes. Consider label classes X_1 and X_2. Any two vertices in label class X_1 cannot be adjacent. Consider $w \in X_2$. If $w \in adj(v,\hat{E}_0)$ for some $v \in X_1$ then w will be the unique successor of v. Thus w is in the chain from v to the root and the lemma holds for the first two label classes.

Assume the lemma is true for label classes up to and including X_n and consider X_{n+1}. Suppose $w \in X_{n+1}$ and $v \in X_i$, $i \leq n$. Either there is some $u_1 \in adj(v,\hat{E}_{i-1})$ such that $i = \gamma(v) < \gamma(u_1) < \gamma(w) = n+1$ or not. In the second case w is in the label class with the smallest label of all vertices in $adj(v,\hat{E}_{i-1})$. So w is the successor of v and the lemma holds. In the first case since $\gamma(u_1) \leq n$ the lemma holds for v and u_1 by induction. The same argument is repeated for some $u_2 \in adj(v,\hat{E}_{i-1})$ such that $\gamma(u_1) < \gamma(u_2) < \gamma(w)$. This way of reasoning comes to an end for some $u_j \in adj(v,\hat{E}_{i-1})$ since this set is finite. The vertices $v, u_1, u_2, \ldots, u_j, w$ obviously are all on the same chain in (V,B) from v to r. (end of proof)

Let T(v) = (V(v),B(v)) be the subtree of (V,B) with root v.

Corollary 2

Assume two disjoint subtrees (V(v),B(v)) and (V(w),B(w)) of (V,B). Then in (V,E) there is no edge (x,y) with $x \in V(v)$ and $y \in V(w)$.

proof:

Assume there is such an edge. Then x and y cannot be in the same label class. Say $\gamma(x) > \gamma(y)$. Since $x \in adj(y,\hat{E}_{\gamma(y)-1})$ x must be in the chain from y to the root (lemma 12). This however, implies that $y \in V(v)$ contrary to the assumption that T(v) and T(w) are disjoint. (end of proof).

By the labeling the graph (V,E) is converted into a palm tree [3.21]. Namely the edge set E of (V,E) is partitioned into the edge sets E1 = B and E2 = E\B. Due to the corollary 2 the edges of E1 and E2 can be identified as the tree edges and fronds respectively.

Corollary 3

Let (V,E) be a connected triangulated graph with an e-tree (V,B). An ordering β for which

$\forall_{x,y\in V}$ [x is successor of y $\rightarrow \beta^{-1}(x) > \beta^{-1}(y)$] is a perfect ordering.

proof:

The obtained ordering β is a perfect ordering if

$\forall_{x\in V}$ [$\{y \mid (y,x) \in E \wedge \beta^{-1}(x) < \beta^{-1}(y)\}$ is a clique].

Consider a chain in the tree from vertex z to the root with vertex set $\{z=z_1,\ldots,z_m = \text{root}\}$. For $\ell\in\{2,3,\ldots,m\}$ holds $\gamma(z_{\ell-1}) < \gamma(z_\ell)$ and $\beta^{-1}(z_{\ell-1}) < \beta^{-1}(z_\ell)$ because z_ℓ is successor of $z_{\ell-1}$. Hence $\gamma(z) < \gamma(z_j)$ and $\beta^{-1}(z) < \beta^{-1}(z_j)$ for $j \in \{2,\ldots,m\}$.

Assume $x \in X_i$ for $i \in \{1,\ldots,k-1\}$. Let C(x) be the vertex set of the chain from x to the root. Due to corollary 2 $adj(x,E) = L(x) \cup U(x)$ with $L(x) = \{y \mid (y,x) \in E \wedge y \in V(x)\}$ and $U(x) = \{y \mid (y,x) \in E \wedge y \in C(x)$. For any $y \in V(x)\backslash\{x\}$ holds $\gamma(x) > \gamma(y)$ and for any $z \in C(x)$ holds $\gamma(x) < \gamma(z)$ because of the chain from y to x respectively from x to the root. Hence $U(x) = adj(x,\hat{E}_{i-1})$, which is a clique by the construction of the label classes.

For each $y \in adj(x,E)$ either $y \in U(x)$ or $y \in L(x)$. In the first case $\beta^{-1}(y) > \beta^{-1}(x)$ because $y \in C(x)$. In the second case $\beta^{-1}(y) < \beta^{-1}(x)$ because of the chain from y to x in T(x).

Hence $\{y \mid (x,y) \in E \wedge \beta^{-1}(y) > \beta^{-1}(x)\} = U(x)$. (end of proof)

Lemma 13

Assume $v \in X_i$, $i>1$, such that v is in a minimal u,w-separation clique in $(\hat{V}_j \hat{E}_j)$, $j=0,\ldots,i-2$. Then there is at least one vertex, say y, such that $\gamma(y) < i$ and $v \in adj(y,\hat{E}_{\gamma(y)-1})$.

proof:

Let the minimal u,w-separation clique be S and D_u and D_w be the components with respect to S containing u and w respectively. As long as D_u and D_w are not empty there is a vertex in D_u and D_w with zero deficiency according to lemma 4, which will be labeled on execution of line 12 of procedure e-tree, whereas v cannot be labeled according to lemma 2. Either D_u or D_w will finally shrink to one vertex, say y, which will be labeled $\gamma(y) \leq i-1$. Since S was minimal $v \in adj(y,\hat{E}_{\gamma(y)-1})$. (end of proof)

Obviously all vertices in label class X_1 have no predecessors since 1 is the smallest label issued. The contrary is not so obvious and needs a proof.

Lemma 14

All vertices with no predecessors are in label class X_1.

proof:

Suppose $v \in X_n$, $n>1$ and v has no predecessor. Thus while $i<n$ during execution of procedure e-tree v is not assigned a label in line 12. Therefore either v does not satisfy the zero-deficiency condition or v is adjacent to some other vertex, say w, that got his label. In the second case v is on the chain from w to the root (lemma 12) meaning that it has a predecessor contrary to the assumption. In the first case due to lemma 2, lemma 13 applies. Thus there is some vertex y such that v is on the chain in (V,B) from y to the root implying that v has a predecessor contrary to the assumption.

Corollary 4

A vertex is in label class X_1 if and only if it has no predecessors. In other words all "tree-tops" are in label class X_1.

Corollary 5

Assume all label classes $X_1,\ldots,X_i$ removed from (V,B). Thus $(\hat{V}_i,\hat{E}_i)$ and $(\hat{V}_i,B(\hat{V}_i))$ are obtained. Then a vertex is in label class X_{i+1} if

and only if it has no predecessor in $(\hat{V}_i, B(\hat{V}_i))$.

proof:

The proof follows from the fact that after removal of the label classes $X_1,\ldots,X_i$ from (V,E) the residual graph is still connected and triangulated. If the labeling is started with "i+1" instead of "1" the statement is obtained. (end of proof)

Lemma 15

Any vertex v in some label class X_i, $i>1$, has at least one predecessor from label class X_{i-1}.

proof:

From lemma 14 it follows that any vertex in X_2 has a predecessor in X_1 otherwise it would have no predecessor at all. So consider the case that $i>2$ and all predecessors of v are in label classes X_j, $j\leq i-2$. Remove all label classes up to and including X_{i-2}. Then v will have no predecessors. This is a contradiction since only X_{i-1} can have vertices with no predecessors. (end of proof)

Theorem 2

For any $v \in V$, $\gamma(v)$ is the length of the critical path in $T(v)$.

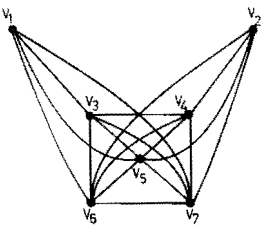
proof:

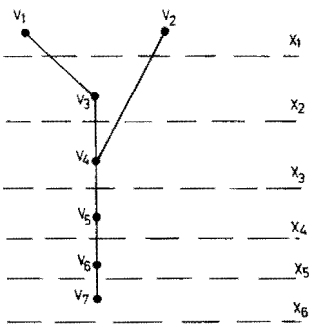
The proof is by induction on the label classes. The case is clear for X_1 and X_2. So assume that the statement is true for all label classes up to and including X_n and consider some $v \in X_{n+1}$. The truth of the statement follows now from the observation (lemma 15) that v has at least one predecessor in X_n, say w. By induction the critical path in $T(w)$ has length n. Traversing from w to v adds one edge to the critical path which proves the statement. (end of proof)

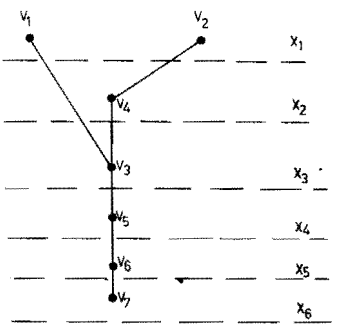
The immediate consequence of theorem 2 is of course that $\gamma(r)$ is the length of the critical path in (V,B).

Procedure e-tree need not yield a unique result. To see this consider the example in fig.(3.3-1). For this example if i=2 both v_3 and v_4 have zero deficiency and the vertex encountered first in line 9 will be inserted into X_2. In the example the length of the critical path is independent of the choice. There arises the question whether this is generally so. The question will be answered in the positive

sense as indicated earlier.







Lemma 16

Assume a connected triangulated graph (V,E) and $U := \{v \in V \mid def(v,E) = \phi\}$ then U consists of a set of cliques which are disjoint.

proof:

Define on U the relation $R(u,v)$ as follows

$\forall_{u,v \in U} [R(u,v) \leftrightarrow u \in adj(v,E) \cup \{v\}]$. It is obvious that for any pair of vertices $u,v \in U$, $R(u,u)$ and $R(u,v) \rightarrow R(v,u)$ holds. Hence the relation is reflexive and symmetric.

Assume $R(u,v)$ and $R(v,w)$ hold with $u,v,w \in U$. If $u = v$ or $v = w$ or $u = w$ then $R(u,w)$ holds. If $u \neq v$ and $v \neq w$ and $u \neq v$ then $(u,v) \in E$ and $(v,w) \in E$. The $def(v,E) = \phi$ means that $(u,w) \in E$ and $R(u,w)$ holds again. Hence the relation is also transitive. The relation is an equivalence. Consequently this equivalence induces a partition of U into the equivalence classes $U_1, U_2, \ldots, U_m$ with $U = \bigcup_{i=1}^{m} U_i$ and $U_i \cap U_j = \phi$ for all i,j where $i \neq j$. Each set U_i is a clique, according to the relation, which consists of one or more vertices. (end of proof)

Lemma 17

Assume a connected triangulated graph (V,E). Suppose $y,z \in V$ such that

$def(y,E) = \phi$, $def(z,E) = \phi$ and $y \in adj(z,E)$.
Then the graphs $(Y,E(Y))$ and $(Z,E(Z))$ with $Y = v\backslash\{y\}$ and $Z = V\backslash\{z\}$ are isomorphic.

proof:
The bijection $\lambda : Y \to Z$ with $\lambda(z) = y$ and for all $x \in V\backslash\{y,z\}$ $\lambda(x) = x$ satisfies the required conditions because $adj(y,Z) = adj(z,Y)$. Fig. (3.3.-2) illustrates this. (end of proof)

Corollary 5

Assume a connected triangulated graph (V,E). Let X_1' and X_1'' be two different choices of the first label class. Then $(\hat{V}_1',\hat{E}_1')$ and $(\hat{V}_1'',\hat{E}_1'')$ (which are the elimination graphs after removing X_1' or X_1'') are isomorphic.

proof:
Obviously the first label class is established by selecting one vertex from every equivalence class U_i, which are defined in lemma 16 by the relation $[R(u,v) \leftrightarrow u \in adj(v,E) \cup \{v\}]$, $i=1,\ldots,m$. The effect of eliminating the first label class can be studied by considering the members of the various equivalence classes U_i independently since these classes are disjoint.
Assume for some i $X_1' \cap U_i = y$ and $X_1'' \cap U_i = z$. If $y = z$ then the same elimination graphs are obtained and the isomorphism is obvious. If however, $y \neq z$ lemma 17 applies because y and z are adjacent since they are in the same equivalence class. (end of proof)

Lemma 16 says that for any triangulated connected graph (V,E) the first label class has a fixed cardinality. From corollary 5 follows that for all possible choices of $X_1 \subseteq U$ the obtained elimination graphs are isomorphic. Lemma 16 and 17 hold for each elimination graph $(\hat{V}_1,\hat{E}_1),\ldots,(\hat{V}_k,\hat{E}_k)$. This implies that also the number of label classes is constant for any given (V,E). So theorem 3 is obtained.

Theorem 3

For a triangulated connected graph (V,E) the length of the critical path in some e-tree is a property of the graph.

This statement assures that the construction of the e-tree is optimal under the present assumptions (the most important one being that the processing of one pivot takes a fixed time slot).

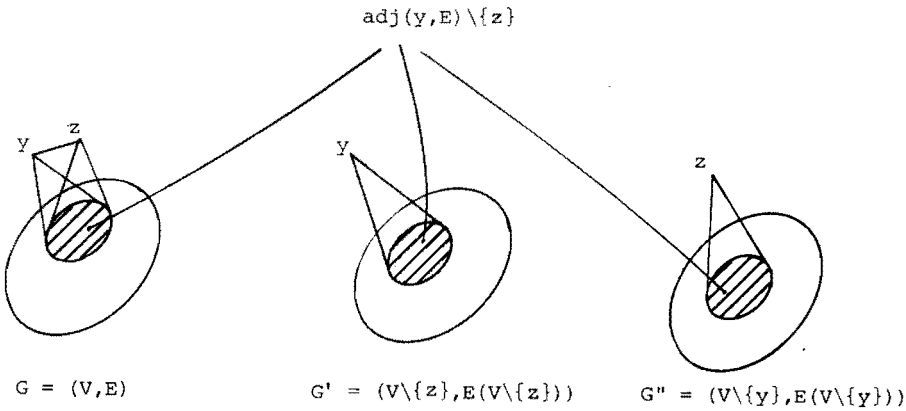


Fig.(3.3-2) Illustration of the isomorphism property.

3.4 Partitioning

Let C denote a set of h clusters in e-tree (V,B); the clusters are denoted by C_i for $i \in \{1,\ldots,h\}$. Assume the set of clusters is a partition of the vertex set V that is to say : $V = \bigcup_{i=1}^{h} C_i$ and $C_i \cap C_j = \phi$, for $i,j \in \{1,\ldots,h\}$ and $i \neq j$.

The cluster graph of (V,B) with respect to C, denoted by $(C,\bar{B})$, contains the clusters as vertices and the edges are given by:

$$(C_i,C_j) \in \bar{B} \leftrightarrow \exists_{v \in C_i, w \in C_j} [(v,w) \in B \wedge \gamma(v) < \gamma(w)] .$$

Let v denote the vertex for which holds $\forall_{x \in C_i} [\gamma(x) < \gamma(v)]$ then this vertex will be called the root of C_i, denoted by r_i, for $i \in \{1,\ldots,h\}$.

Lemma 18

Let $\bar{\beta}$ be an ordering for the clusters $\bar{\beta}$: $\{1,\ldots,|C|\} \rightarrow C$ such that:

$$\forall_{C_i,C_j \in C} [C_j \text{ is a successor of } C_i \rightarrow \bar{\beta}^{-1}(C_j) > \bar{\beta}^{-1}(C_i)].$$

An ordering β such that:

(i) $\forall_{C_i \in C} [\forall_{u,w \in C_i} [w \text{ is a successor of } u \rightarrow \beta^{-1}(w) > \beta^{-1}(u)]]$

(ii) $\forall_{C_i,C_j \in C} [\forall_{u \in C_i, v \in C_j} [\bar{\beta}^{-1}(C_j) > \bar{\beta}^{-1}(C_i) \rightarrow \beta^{-1}(v) > \beta^{-1}(u)]$

is a perfect ordering.

Proof:

For any edge $(u,v) \in B$ where v is successor of u, there are two possibilities. Firstly $u,v \in C_i$ then due to (i) corollary 3 holds, $i \in \{1,\ldots,h\}$. Secondly, $u \in C_i$ and $v \in C_j$ with $i \neq j$ means $(C_i,C_j) \in \bar{B}$ where C_j is successor of C_i due to the ordering $\bar{\beta}$ which induces by (ii) the $\beta^{-1}(v) > \beta^{-1}(v)$ such that corollary 3 holds again, for $i,j \in \{1,\ldots,h\}$. (end of proof)

The partition C and ordering according to lemma 18 yields a matrix structure given by fig.(3.4-1).The coefficients A'_{ij} of A' are matrices

$$A' = \begin{bmatrix} A'_{11} & A'_{12} & \cdots & A'_{1h} \\ A'_{21} & A'_{22} & \cdots & A'_{2h} \\ \vdots & \vdots & & \vdots \\ A'_{h1} & A'_{h2} & \cdots & A'_{hh} \end{bmatrix}$$

Fig.(3.4-1) Matrix structure associated with $(C,\bar{B})$

itself, for $i,j \in \{1,\ldots,h\}$. The diagonal matrix A'_{ii} is associated with the subgraph of (V,E) given by $(\bar{\beta}^{-1}(i),E(\bar{\beta}^{-1}(i))$, for $i \in \{1,\ldots,h\}$. The off diagonal matrices A'_{ij} and A'_{ji} are associated with the set of edges defined by $\{(u,v) \in E \mid u \in \bar{\beta}^{-1}(i) \text{ and } v \in \bar{\beta}^{-1}(j)\}$, for $i,j \in \{1,\ldots,h\}$.

Due to corollary 2, A'_{ij} and A'_{ji} are zero matrices if in the cluster graph $(C,\bar{B})$ there is no path between $\bar{\beta}^{-1}(i)$ and $\bar{\beta}^{-1}(j)$.

The cluster graph $(C,\bar{B})$ may be seen as having the same meaning as the e-tree if instead of single diagonal coefficients, diagonal matrices are taken as a pivot.

Many different partitions into clusters are possible.A partition may be obtained as follows.

Choose some vertex $v \in V$ and consider the chain from v to the root r in the e-tree (V,B). This chain defines a cluster, which will be denoted by S(v). Assume the set adj(S(v),B) is given by : $\{r_1,\ldots,r_h\}$. Each vertex $r_i \in adj(S(v),B)$ defines a cluster $V(r_i)$. If $h > 1$ then S(v) is a separator in (V,E) due to corollary 2. The h distinct components are given by: $\{(V(r_1),E(V(r_1))),\ldots,(V(r_h),E(V(r_h)))\}$.

For each component $(V(r_i),E(V(r_i)))$, with $(V(r_i),B(r_i))$ as e-tree,the above process may be repeated.

The clusters obtained this way depend on the choice of v in each component. The choice of v can be based on various criteria.In chapter 4

partitioning will be used to support the scheduling of tasks.

4. SCHEDULING

4.0 Introduction

In this chapter the scheduling strategy is presented for the solution job of the set of linear equations. The scheduling model as presented in section 2.3 will be used.
In section 4.1 the scheduling model in case of an ideal parallel computer is given in order to demonstrate the methods which will be used.
In section 4.2 the job system for the asynchronous array computer is derived.
Finally in section 4.3 the proposed scheduling strategy is presented and applied to a L\U-decomposition job.

4.1 Scheduling model for the ideal parallel computer

By means of the associated graph and the e-tree (V,B), the job systems of the L\U-decomposition and forward substitution job ("lu-job") and the back substitution job ("bs-job") are defined for the ideal parallel computer.

The resource system, containing the limited resources, is given by:

- $P = \{P_1,\ldots,P_m\}$ m identical processors, which are able to exploit all parallelism associated with any pivot.
- $R = \{R_1,\ldots,R_s\}$ s artificial resources with each an amount $rm(R_i) = 1$, for $R_i \in R$. The purpose of these additional resources will be explained shortly.

Before defining the tasks some data sets will be defined which will be stored in the memory.
With every vertex $v \in V$ will be associated the data sets : A_v and Q_v.
The data set A_v contains the coefficients $a(i,k)$, $a(k,i)$ and $a(i,n+1)$, for $i = \alpha^{-1}(v)$ and $k \in \{\alpha^{-1}(w) \mid w \in madj(v) \cup \{v\}\}$. With the notational convention used in the L\U-decomposition procedures of section 3.0.
The data set Q_v contains the coefficients $q_v(h,k)$, $q_v(h,n+1)$, for $k,h \in \{\alpha^{-1}(w) \mid w \in madj(v)\}$, to store intermediate results. The coefficients of Q_v are initialized with zero.

The job system for the lu-job.
For every vertex $u \in V$ is defined a task which accomplishes the L\U-decomposition and forward substitution on u. The task is called partial

L\U-decomposition and forward substitution, and will be denoted by: plui-task(u). The i indicates the ideal parallel computer. The plui-task(u) is given by the procedure "plui(u)" which operates on the associated data sets of u and its successor.

```
 1. procedure plui(u);
 2. begin
    comment This task operates on the data sets of u and its succes-
            sor v. The statements given in lines 4-9 accomplish the
            actual task. The statements given in lines 10-13 accom-
            plish the 'data transport' between the data sets asso-
            ciated with u and v;
 3. k ← α^-1(u); j ← α^-1(v); I ← {α^-1(w) | w ∈ madj(u)};
 4. for each ℓ ∈ I do
 5.     begin
 6.     a(ℓ,k) ← a(ℓ,k) / a(k,k);
 7.     for each m ∈ I ∪ {n+1} do
 8.         q_u(ℓ,m) ← q_u(ℓ,m) - a(ℓ,k) * a(k,m);
 9.     end;
10. for each ℓ ∈ I do
11.     for each m ∈ I ∪ {n+1} do
12.     if (ℓ=k ∨ m=j) then a(ℓ,m) ← a(ℓ,m) + q_u(ℓ,m)
13.                    else q_v(ℓ,m) ← q_v(ℓ,m) + q_u(ℓ,m);
14. end; procedure plui(u)
```

The resulting task graph $(T,\vec{S})$ for the lu-job is equal to the e-tree (V,B). The precedence relations among the tasks are given by:

$$\forall_{(u,v)\in B}[\text{plui-task}(u) \text{ is the successor of plui-task}(v) \text{ iff } \gamma(u) > \gamma(v)] \tag{4.1.1}$$

Consider two tasks: plui-task(x) and plui-task(y) which are allowed to be processed in parallel by $(T,\vec{S})$. However, if $madj(x) \cap madj(y) \neq \phi$ then update conflicts are possible during execution of the statements given in lines 10-13 of procedure plui(u). Due to the introduction of the data sets Q this can only happen if x and y have a common successor.

To avoid the update conflicts it is sufficient to exclude plui-task(x) and plui-task(y) from being executed at the same time if they have the same successor. This is accomplished by the set of artificial resources; for each vertex u a resource R_u with an amount $rm(R_u) = 1$

is introduced.

Each task will demand a processor and an amount of one of the additional resource which is associated with the successor task.

The task system for the lu-job is given by:

- the set of tasks : $T = \{\text{plui-task}(u) \mid u \in V\}$
- the task graph : $(T,\vec{S})$ is equal to (V,B) with precedence relations of (4.1.1)
- the task durations : $\tau(\text{plui-task}(u)) = \begin{cases} 1 \text{ time slot for } u \in V\setminus\{r\} \\ 0 \text{ time slots for } u = r \end{cases}$
- the resource demands : $r(R_V, \text{plui-task}(u)) = \begin{cases} 1 \text{ if } (u,v) \in B \wedge \gamma(u) < \gamma(v) \\ 0 \text{ if otherwise} \end{cases}$

The job system for the bs-job.

For every vertex $u \in V$ is defined a task which accomplishes the back substitution on u. The task is called partial back substitution and will be denoted by : pbsi-task(u). The pbsi-task(u) is given by the procedure "pbs(u)".

```
1. procedure pbsi(u);
2. begin
3. i ← α^-1(u);
4. a(i,n+1) ← a(i,n+1) / a(i,i);
5. for each ℓ ∈ {α^-1(w) | w ∈ adj(u) \ madj(u)} do
            a(ℓ,n+1) ← a(ℓ,n+1) - a(i,n+1) * a(ℓ,i);
6. end;
```

The job system of the bs-job is distinguished from the task system of the lu-job by an accent if neccessary.

- the set of tasks : $T' = \{\text{pbsi-task}(v) \mid v \in V\}$.
- the task graph : $(T',\vec{S}')$ is equal to (V,B) with precedence relations : $\forall_{(u,v)\in B}[\text{pbsi-task}(u)$ is the successor of pbsi-task(v) iff $\gamma(u) < \gamma(v)]$.
- the task durations: $\tau(\text{pbsi-task}(v)) = 1$ time slot for $u \in V \setminus X_1$.

The parallel processing of the lu-job is demonstrated in fig.(4.1-1) by a simple example (Without forward substitution). A schedule for 3 processors is obtained by a list schedule strategy which will be

presented in section 4.3.1.

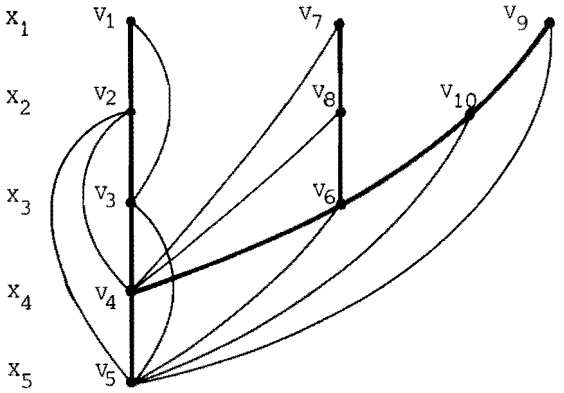


Fig.(4.1-a) Triangulated connected graph with e-tree.

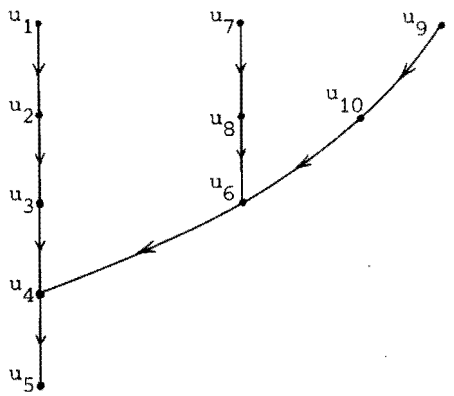


Fig.(4.1-b) $(T,\vec{S})$ for the lu-job.

$P = \{P_1, P_2, P_3\}$

$R = \{R_4, R_6\}$ with $rm(R_i) = 1$, for $R_i \in R$.

Fig.(4.1-1c) Resource system.

T	u_1	u_2	u_3	u_4	u_5	u_6	u_7	u_8	u_9	u_{10}
$\tau(u_i)$	1	1	1	1	0	1	1	1	1	1
$r(R_4,u_i)$	0	0	1	0	0	1	0	0	0	0
$r(R_6,u_i)$	0	0	0	0	0	0	0	1	0	1
$\pi^{-1}(u_i)$	1	4	7	9	10	8	2	5	3	6

Fig.(4.1-1d) Input tabel for list schedule.

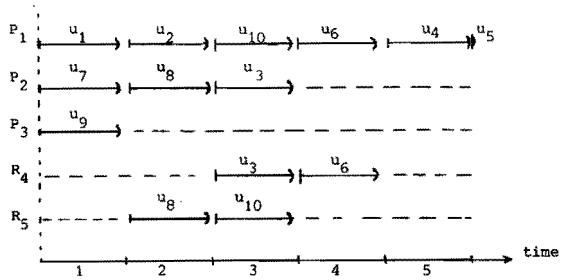


Fig.(4.1-1e) Gantt chart of the schedule

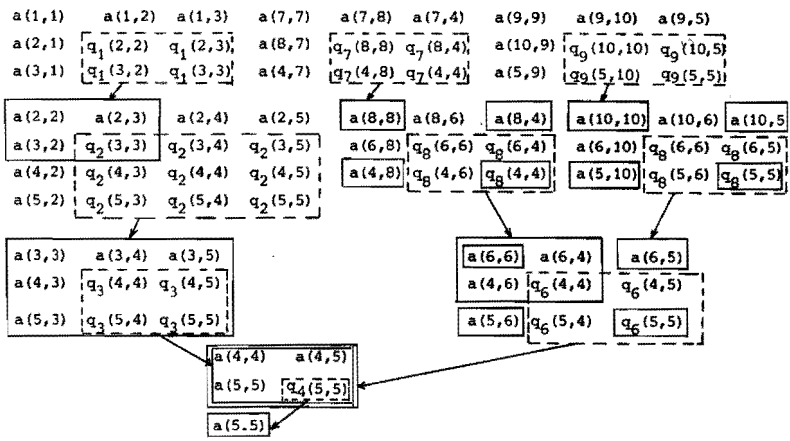


Fig.(4.1-1f) Data sets determined by the e-tree and its flow.

4.2 Job and resource system for the solution of the linear equations on the asynchronous array computer.

In the previous section job and resource systems for the solution of the linear equations on the ideal parallel computer have been given. By means of the associated graph and the e-tree a set of tasks was defined and the e-tree could be considered as the task graph.
The model of the asynchronous array computer as presented in chapter 2 accounts for the limited computation power of the processors and the restricted capacity of communication channels. Due to the non ideal computer model the job systems which are derived in the previous section have to be modified.

4.2.1 The set of tasks and task graph of the lu-job for the asynchronous array computer.

The communication between the computers depends on the interdependence of the tasks, the assignment of the tasks to the computers and on the rules according to which the data are distributed to the memory modules. Some data will be stored in all memory modules and others in one particular memory. The purpose is to store the data such that the increase of the schedule length due to the communication will be minimized.

An intuitive way to achieve this is to store the data, on which the task operates, in the memory of the computer to which the task is assigned.

Nearly the same organization will be used as in the case of the ideal parallel computer. The L\U-decomposition and forward substitution on pivot u will be accomplished by three tasks:

- the plu-task(u) which is a modification of the plui-task(u) in case of the ideal computer;
- the com-task(u) (communication task);
- the add-task(u) (addition task).

Before the tasks are specified the data allocation and structure will be given.

The data allocation and structure are determined by the assignment of the plu tasks and the associated graph.

A vertex will be called "assigned to computer C" if the plu-task(u) is assigned to computer C. The relation R(u,v) in V defined by

$\forall_{u,v\in V}[R(u,v) \leftrightarrow$ u and v are assigned to the same computer and between u and v exists a chain in $(V,B)]$,

partitions V into a set of p clusters $\{U_1,U_2,\ldots,U_p\}$, such that

$\bigcup_{i=1}^{p} U_i = V$ and $U_i \cap U_j = \phi$, for $i = j$ and $i,j \in \{1,\ldots,p\}$.

The vertex $v \in U_j$ is called the "root of U_j", to be denoted by r_j, if $\forall_{u\in U_j}[\gamma(r_j) \geq \gamma(u)]$.

The "predecessor set of U_j", to be denoted by F_j, is defined by:

$F_j = \{x \mid x \in adj(U_j,B) \cap V(r_j)\}$.

A cluster U_j is called "assigned to computer C" if r_j is assigned to this computer C.

With every cluster U_j will be associated a graph (W_j,E_j) where $W_j = U_j \cup \mathrm{madj}(r_j,E)$ and $E_j = E(W_j)$. Further with every cluster U_j are associated the following data sets A_{r_j}, Ω_{r_j} and D_{ur_j}, for $u \in F_j$.
The data set A_{r_j} contains the coefficients $a(k,\ell)$, $a(\ell,k)$ and $a(\ell,n+1)$, for $k \in \{\alpha^{-1}(w) \mid w \in W_j\}$ and $\ell \in \{\alpha^{-1}(w) \mid w \in U_j\}$.
The data set Ω_{r_j} contains the coefficients $q_{r_i}(k,\ell)$ and $q_{r_i}(k,n+1)$, for $k,\ell \in \{\alpha^{-1}(w) \mid w \in \mathrm{madj}(u,E)\}$.
The data set D_{ur_j} contains the coefficients $d_{ur_j}(k,\ell)$ and $d_{ur_j}(k,n+1)$, for $k,\ell \in \{\alpha^{-1}(w) \mid w \in \mathrm{madj}(u,E)$.
The data sets A_{r_j} and Ω_{r_j} were already introduced in section 4.1. Then coefficients of Ω_{r_j} are again initialized with zero.
The data set D_{ur_j} will be used to store the coefficients Ω_u which must be 'transported' from cluster U_i with $u = r_i$ to cluster U_j. The coefficients of D_{ur_j} are initialized with zero.
The graph (W_j,E_j) is the associated graph of the submatrices which are determined by the data sets A_{r_j} and Ω_{r_j}.
If a cluster is assigned to a computer C the associated data sets are allocated to this computer.

The plu-task(u) can now be specified. The procedure "plu(u)" accomplishes the plu-task(u). The procedure operates only on the data structure which is determined by (W_j,E_j) where $u \in U_j$

```
procedure plu(u);
begin
comment u ∈ U_j, α(n+1) ∉ V;
k ← α^-1(u); I ← {α^-1(w) | w ∈ madj(u,E)};
for each ℓ ∈ I do
    begin
    a(ℓ,k) ← a(ℓ,k)/a(k,k);
    for each m ∈ I ∪ {n+1} do
        if (α(ℓ) ∈ U_j or α(m) ∈ U_j)
        then a(ℓ,m) ← a(ℓ,m) - a(ℓ,k) * a(k,m)
        else q_r_j(ℓ,m) ← q_r_j(ℓ,m) - a(ℓ,k) * a(k,m);
    end; ℓ loop
end; procedure plu(u)
```

Consider the plu-task(u) and plu-task(v), where v is the successor of u. Assume plu-task(u) and plu-task(v) have been assigned to computer C_k and C_ℓ, respectively. Further let $u \in U_j$ and $v \in U_i$. The data sets which are associated with U_j and U_i are allocated to computer C_k and C_ℓ, respectively.

To accomplish the L\U-decomposition and forward substitution on pivot u two situations must be regarded.

- The tasks have been assigned to the same computer. In this case $U_j = U_i$ and the desired process will be accomplished by the plu-task(u), which will be executed by computer C_k.
- The tasks have been assigned to different computers. In this case u is the root of U_j. The first part is accomplished by the plu-task(u), which will be executed by computer C_k. Subsequently the intermediate results, stored in Q_{r_j}, are transmitted to computer C_ℓ by computer C_k. This is assomplished by com-task(u). In the memory M_ℓ of the computer C_ℓ the data set D_{ur_i} is used to store the received data. Finally, computer C_ℓ has to add the intermediate results, stored in D_{ur_i}, to the data sets A_{r_i} and Q_{r_i}. This is accomplished by the add-task(u).

The com-task(u) is given by the procedures broadcast(Q_{r_j}, IBQ_{r_j}, h) and receive(Dur_i, IRD_{ur_i}, h) which are thought to be executed by the computers C_k and C_ℓ, respectively. The array IBQ_{r_j} is equal to IRD_{ur_i} and contains the set of indices given by $I \times I \cup \{n+1\}$, where $I = \{\alpha^{-1}(w) \mid w \in madj(u,E)\}$.

The add-task(u) is given by procedure "add(u)", which will be executed by computer C_ℓ.

```
procedure add(u);
comment u ∈ U_j, α(n+1) ∉ V;
begin
k ← α^-1(u); I ← {α^-1(w) | w ∈ madj(u,E)};
for each ℓ ∈ I do
    begin
    for each m ∈ I ∪ {n+1} do
        if (α(ℓ) ∈ U_j or α(m) ∈ U_j) then a(ℓ,m) ← a(ℓ,m) + d_ur_i(ℓ,m)
                                      else q_r_i(ℓ,m) ← q_r_i(ℓ,m) + d_ur_j(ℓ,M);
    end; ℓ loop
end
```

The set of tasks
$\{\text{plu-task}(u) \mid u \in U_j\} \cup \{\text{add-task}(u) \mid u \in F_j\} \cup \{\text{com-task}(r_j)\}$ are associated with cluster U_j. The set of tasks will be executed by computer C_k if U_j is allocated in memory M_k.

Because the assignment of the plu-task(u) is not known in advance the tasks com-task(u) and add-task(u) are concatenated to every plu-task(u), for $u \in V \setminus \{r\}$. If v is the successor of u and both vertices are assigned to the same computer then the com-task(u) and add-task(u) are empty.
The modified task graph $(T,\vec{S})$ for the lu-job can now be given. The set of tasks is given by:

$$T = \{\text{plu-task}(u), \text{com-task}(u), \text{add-task}(u) \mid u \in V \setminus \{r\}\} \cup \{\text{plu-task}(r)\},$$

and the set of edges $\vec{S}$ is given by:

$$\vec{S} = \{(\text{plu-task}(u), \text{com-task}(u)), (\text{com-task}(u), \text{add-task}(u)), (\text{add-task}(u), \text{plu-task}(v)) \mid (u,v) \in B \wedge \gamma(u) < \gamma(v)\},$$

where com-task(u), add-task(u) and plu-task(v) are the successor tasks of plu-task(u), com-task(u) and add-task(u) respectively.
Fig.(4.2-1) shows the task graph of the lu-job for the considered sparse matrix example.

4.2.2 The set of tasks and task graph of the bs-job for the asynchronous array computer.

The back substitution on a pivot $v \in V$ will be accomplished by two tasks:

- pbs-task(v), which is a modification of the pbsi-task(u) in case of the ideal parallel computer;
- com-pbs-task(v).

The tasks will operate on the data structure and allocation as given in the previous section 4.2.1. Hence the pbs-task(v) will be assigned to the same computer as to which the plu-task(v) has been assigned. The pbs-task(v) will be given by procedure "pbs(v)"; note the row- instead of the column-wise organization.

```
procedure pbs(v);
begin
  comment v ∈ U_i;
k ← α^-1(v); I ← {α^-1(w) | w ∈ madj(v,E)};
for each ℓ ∈ I do
    if (α(ℓ) ∈ U_i) then a(k,n+1) ← a(k,n+1) - a(k,ℓ) * a(ℓ,n+1)
                    else a(k,n+1) ← a(k,n+1) - q_{r_j}(ℓ,n+1) * a(k,ℓ);
a(k,n+1) ← a(k,n+1) / a(k,k);
end;
```

The pbs-task(v) may be executed if the coefficients a(k,n+1) and $q_{r_i}(\ell,n+1)$ contain the components x(m) of $\underline{x}$, for $m \in \{\alpha^{-1}(w) \mid w \in madj(v)\}$, $\ell \in \{\alpha^{-1}(w) \mid w \in madj(v) \cap madj(r_i)\}$ and $k \in \{\alpha^{-1}(w) \mid w \in madj(v) \cap U_i\}$.

Consider a vertex $v \in U_i$ and let $F_i(v)$ be defined by:

$F_i(v) = \{x \mid (x \in F_i) \wedge (v \text{ is the successor of } x)\}$

To accomplish the back substitution on pivot v two situations must be regarded:

- The set $F_i(v)$ is empty. In this case the desired process will be performed by the pbs-task(u).
- The set $F_i(v)$ is not empty. In this situation data exchange will be necessary between the computer C_ℓ, to which v has been assigned, and the computer C_{k_s}, to which $u_s \in F_i(v)$ has been assigned for $1 \le s \le |F_i(v)|$.

After completion of the pbs-task(v) computer C_ℓ has to transmit the coefficients a(m,n+1) to computer C_{k_s}, for $m \in \{\alpha^{-1}(w) \mid madj(u_s,E)\}$. Computer C_{k_s} receives and stores the coefficients in the corresponding locations of Ω_{u_s}.

It is proposed to organize this data exchange by a single communication task called com-pbs-task(v).

The components x(m) of $\underline{x}$, for $m \in \{\alpha^{-1}(w) \mid w \in madj(x) \mid x \in F_i(v)\}\}$, have to be broadcasted. These components are stored in the data sets A_{r_i} and Ω_{r_i} in computer C_ℓ. The array IBA_{r_i} contains the indices of the coefficients in A_{r_i} given by $\{\alpha^{-1}(w) \mid w \in madj(v) \cap U_i\}$, and array $IB\Omega_{r_i}$ contains the indices of the coefficients in Ω_{r_i} given by $\{\alpha^{-1}(w) \mid w \in madj(v) \cap madj(r)\}$. The procedure broadcast(A_{r_i}, IBA_{r_i}, Ω_{r_i}, $IB\Omega_{r_i}$, k) accomplishes the desired broadcast when it is executed by computer C_ℓ.

Let IRQ_{u_s} contain the indices of the coefficients in Q_{u_s} given by $\{\alpha^{-1}(w) \mid w \in madj(v) \cap (U_i \cup madj(r_i))\}$.
Each computer C_{k_s} receives and selects the required coefficients by execution of procedure receive(Q_{u_s}, IRQ_{u_s}, k) for $1 \le s \le |F_i(v)|$.

Because the assignment of the vertices $u \in V$ is not known in advance to every pbs-task(u), $u \in V \setminus X_1$, a com-pbs-task(u) is concatenated.
The new set of tasks T' for the bs-job is given by:

$$T' = \{\text{pbs-task}(u), \text{com-pbs-task}(u) \mid u \in V \setminus X_1\} \cup \{\text{pbs-task}(u) \mid u \in X_1\},$$

and the new set of edges $\vec{S}'$ is given by:

$$\vec{S}' = \{(\text{pbs-task}(v), \text{com-pbs-task}(v)), (\text{com-pbs-task}(v), \text{pbs-task}(u)) \mid (u,v) \in B \wedge (\gamma(u) < \gamma(v))\},$$

where com-pbs-task(v) is the successor task of pbs-task(v) and pbs-task(u) is a successor task of com-pbs-task(v).
Fig.(4.2-2) shows the task graph for the considered sparse matrix example.

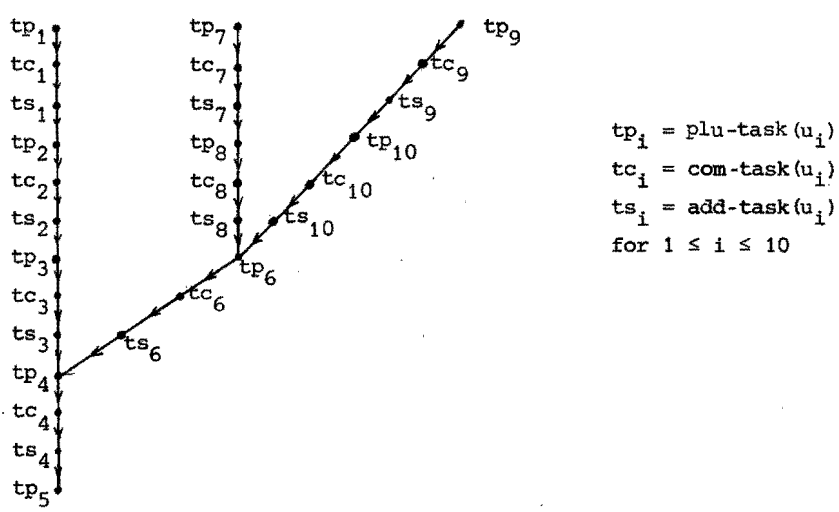


Fig.(4.2-1) Modified task graph $(T,\vec{S})$ for lu-job.

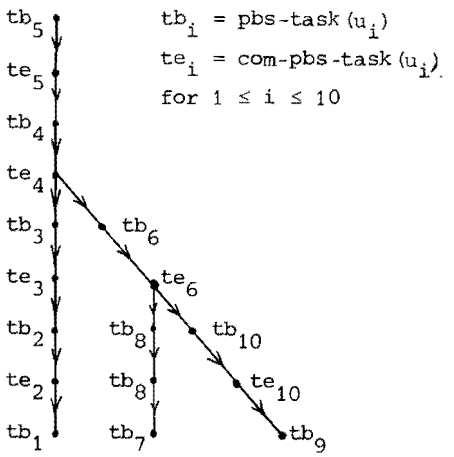


Fig.(4.2-2) Modified task graph $(T',\vec{S}')$ for bs-job.

4.2.3 Task duration of tasks of the lu- en bs-job.

If the necessary synchronization between the computers is obtained by the proposed timing instructions it is desirable to have the exact values for the task duration. If this is not possible estimates of the duration have to be used. These estimates must be upper limits for the task durations.

The duration of a task depends on the value of the operands in the task instructions, the data structure and organization of the hardware. Note that the time necessary to perform an arithmetic operation may depend on the values of the operands. Assume the durations of the tasks, occurring during the execution of the lu-job and bs-job, are determined with sufficient accuracy by counting only the floating point operations.

The operations which are involved with column (n+1) will be out of consideration.

The time needed for a divide, multiply and add or substract operation is given by: τ_{div}, τ_{mul} and τ_{add}.

The communication time to exchange k floating point values is given by:

$$\text{communication time} = k * \tau_{com} + \tau_{overh} \qquad (4.2.1)$$

where τ_{com} is the communication time to transport one value and τ_{overh}

is the required overhead time which occurs every time the communication task is started to transmit a vector of k values.

The duration of the tasks of the lu-job.
If all tasks would be assigned to a single computer then the durations of the com-tasks and add-tasks are zero and the duration of any plu-task(u), denoted by $\tau 1(u)$, is given by:

$$\tau 1(u) = |\mathrm{madj}(u)| * \tau_{div} + |(\mathrm{madj}(u)|^2 * (\tau_{mul} + \tau_{add}). \qquad (4.2.2)$$

If the tasks are assigned to two or more computers the task durations cannot be obtained in this simple way. If an update operation is performed on a coefficient of some data set Ω it is necessary to distinguish whether the coefficient is zero or not. Hence the duration of the tasks will also depend on the assignment of the tasks and the sequence in which they are processed.

Consider a cluster $U_i = \{u_1,\ldots,u_n\}$ with $r_i = u_n$ and $F_i = \{u_0\}$ where $(U_i, B(U_i))$ is a chain from u_1 to u_n in the e-tree. Let $u_0 \in U_j$. The durations of the tasks which are associated with the cluster U_i are now determined.
First the add-task(u_0), as given by procedure add(u_0), has to update the coefficients $q_{u_0}(k,\ell)$ by the coeffcients $d_{u_0u_n}(k,\ell)$ for $k,\ell \in \{\alpha^{-1}(w) \mid w \in \mathrm{madj}(u_0)\}$.
Distinction must be made whether the update operation is performed on a zero or nonzero coefficient. At this moment all $|\mathrm{madj}(u_n)|^2$ coefficients of data set Ω_{u_n} are zero. The number of update operations on coefficients which are initially zero is given by $|\mathrm{madj}(u_0,u_n)|^2$. Hence only $|\mathrm{madj}(u_0)|^2 - |\mathrm{madj}(u_0,u_n)|^2$ add operations are necessary. The duration for add-task(u_0) is given by:

$$\tau(\text{add-task}(u_0)) = (|\mathrm{madj}(u_0)|^2 - |\mathrm{madj}(u_0,u_n)|^2) * \tau_{add} \qquad (4.2.3)$$

Now $|\mathrm{madj}(u_n)|^2 - |\mathrm{madj}(u_0,u_n)|^2$ locations of Ω_{u_n} are still zero.
The next task, plu-task(u_1), as given by procedure plu(u_1), has to perform $|\mathrm{madj}(u_1)|^2$ update operations from which $|\mathrm{madj}(u_1,u_n)|^2$ on coefficients of Ω_{u_n}. However $|\mathrm{madj}(u_0,u_n)|^2$ of those coefficients have become nonzero. The resulting number of update operations on zero coefficients is given by: $|\mathrm{madj}(u_1,u_n)|^2 - |\mathrm{madj}(u_0,u_n)|^2$. The duration of plu-task(u_1) is given by:

$$\tau(\text{plu-task}(u_1)) = \tau 1(u_1) + (-|\text{madj}(u_1,u_n)|^2 + |\text{madj}(u_0,u_n)|^2)\tau_{add}.$$

The above reasoning can be repeated for $u_\ell \in \{u_2,u_3,\ldots,u_n\}$, (4.2.4) gives the duration of a plu-task(u_ℓ).

$$\tau(\text{plu-task}(u_\ell)) = \tau 1(u_\ell) + (-|\text{madj}(u_\ell,u_n)|^2 + |\text{madj}(u_{\ell-1},u_n)|^2)\tau_{add}. \tag{4.2.4}$$

The durations of add-task(u_ℓ) and com-task(u_ℓ) are zero for $u_\ell \in \{u_1,\ldots,u_{n-1}\}$.
The duration for the com-task(u_n) is given by:

$$\tau(\text{com-task}(u_n)) = |\text{madj}(u_n)|^2\tau_{com} + \tau_{overh}.$$

The total duration for the plu-tasks for the chain $u_1,u_2,\ldots,u_k$ with $k \le n$ is given by:

$$|\text{madj}(u_0,u_n)|^2\tau_{add} + \sum_{i=1}^{k} \tau 1(u_i) - |\text{madj}(u_k,u_n)|^2\tau_{add}. \tag{4.2.5}$$

The second term is the duration in case of one computer, the third term accounts for the zero coefficients of Ω_{u_n}, which would be updated, and the first term gives the number of coefficients in Ω_{u_n} which have been made nonzero by u_0.

Consider a cluster $U_i = U_i' \cup U_i'' = \{u_1,\ldots,u_n\} \cup \{v_i,\ldots,v_\ell\}$ with $r_i = u_n$ and $F_i = \{u_0,v_0\}$ such that $(U_i',B(U_i'))$ and $(U_i'',B(U_i''))$ are chains in (V,B) from u_1 to u_n and v_1 to v_ℓ, respectively. The chains are connected by $(v_\ell,u_k) \in B$. Let $u_0 \in U_j$, $v_0 \in U_m$, $(u_0,u_1) \in B$ and $(v_0,v_1) \in B$. Fig.(4.2-3) shows the considered U_i.

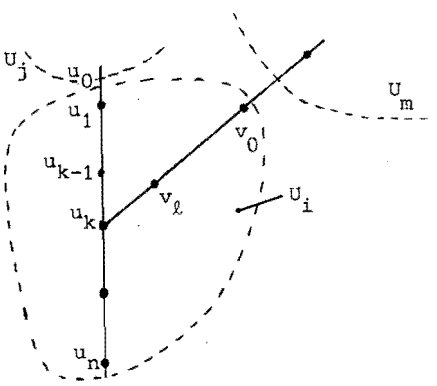


Fig.(4.2-3) The considered cluster U_i.

If $\text{madj}(u_s, v_t, u_n) \neq \phi$, for any $u_s \in \{u_0, \ldots, u_{k-1}\}$ and $v_t \in \{v_0, \ldots, v_\ell\}$ it is evident that the durations of the associated plu-tasks interfere with each other by means of the number of nonzero coefficients in data set Ω_{u_n}.

If a submatrix, defined by $\text{madj}(u_{k-1}, v_\ell, u_n)$ is used to store the intermediate update results produced by the tasks $\{\text{add-task}(v_0)\} \cup$ $\cup \{\text{plu-task}(v_t) \mid 1 \leq t \leq \ell\}$, this interference is avoided. Due to this extra submatrix the already stated equations (4.2.1) - (4.2.5) hold for the tasks associated with $\{u_0, \ldots, u_{k-1}\}$ and $\{v_0, \ldots, v_\ell\}$.

The $\text{plu-task}(u_k)$ starts with adding the intermediate update results stored in the submatrix to the coefficients defined by $\text{madj}(u_{k-1}, v_\ell, u_n)$ of Ω_{u_n}; this takes a time $|\text{madj}(u_{k-1}, v_\ell, u_n)|^2 \tau_{add}$. Besides the divide and multiply operations $|\text{madj}(u_k)|^2$ update operations have to be performed. $|\text{madj}(u_k, u_n)|^2$ update operations have to be performed on coefficients of Ω_{u_n}. Of these $|\text{madj}(u_k, u_n)|^2$ coefficients $|\text{madj}(u_{k-1}, u_n)|^2 +$ $+ |\text{madj}(v_\ell, u_n)|^2 - |\text{madj}(u_{k-1}, v_\ell, u_n)|^2$ coefficients are nonzero and hence have to be taken into account for calculation of the task duration. The duration of the task $\text{plu-task}(u_k)$ is given by:

$$\tau(\text{plu-task}(u_k)) = (|\text{madj}(u_{k-1}, u_n)|^2 + |\text{madj}(v_\ell, u_n)|^2)\tau_{add} + \\ + \tau 1(u_k) - |\text{madj}(u_k, u_n)|^2 \tau_{add}. \qquad (4.2.6)$$

Equation (4.2.6) is obtained from (4.2.4) by addition of the factor $|\text{madj}(v_\ell, u_n)|^2 \tau_{add}$. This can be extended to the general case of a vertex with m predecessors.

Now assume the vertices $v_1, \ldots, v_\ell$ do not exist. The duration for the $\text{add-task}(v_0)$ is given by (4.2.3); if the coeffcients of the data set $D_{v_0 u_n}$, defined by $\text{madj}(v_0, u_{k-1}, u_n)$, are processed by the $\text{plu-task}(u_k)$.

It is now possible to state the expressions for the general case. Consider a cluster U_j. The duration of the tasks which are associated with this cluster are given by the equations (4.2.7) - (4.2.9).

$$\tau(\text{add-task}(u)) \begin{cases} = (|\text{madj}(u)|^2 - |\text{madj}(u, r_j)|^2) * \tau_{add} & \text{for } u \in F_j \\ = 0 & \text{for } u \in U_j \end{cases} \qquad (4.2.7)$$

$$\tau(\text{plu-task}(u)) = \sum_{v \in I} |\text{madj}(v,r_j)|^2 \tau_{add} + \tau 1(u) - |\text{madj}(r_j,u|^2 \tau_{add}$$

$$\text{where } I = \{w \mid (w,u) \in B \wedge \gamma(u) > \gamma(u)\} \quad \text{for } u \in U_j \qquad (4.2.8)$$

$$\tau(\text{com-task}(r_j)) = \tau_{overh} + |\text{madj}(r_j)|^2 \tau_{com} \qquad (4.2.9)$$

The duration of the tasks of the bs-job.

In contrast to the previously considered tasks the durations of the pbs-tasks are independent of the assignment. The duration of a pbs-task(u) is given by:

$$\tau(\text{pbs-task}(u)) = |\text{madj}(u)| * (\tau_{mul} + \tau_{add}) + \tau_{div} \qquad (4.2.10)$$

if it is assumed that the component of $\underline{c}$ determined by u is nonzero. The duration of a com-pbs-task(u) is given by:

$$\tau(\text{com-pbs-task}(u)) = |\{\text{madj}(x) \mid x \in F_i(v)\}\ \tau_{com} + \tau_{overh} \quad \text{for } u \in U_i \qquad (4.2.11)$$

4.2.4 Resource system and resource demands

The asynchronous array computer model accounts only for the number of computers, their processing capacity and communication resources. The resource system of the proposed parallel computer is given by:

- $P = \{P_1,\ldots,P_m\}$ m identical processors with limited computing capacity.
- $R = \{R_1,\ldots,R_m,R_{m+1}\}$ (m+1) additional resources with amount: $rm(R_i) = 1$, for $1 \le i \le m$ and $rm(R_{m+1}) = k$.

A processor P_i and its memory M_i can be regarded as one unit, due to the communication rules, and may be denoted by processor or computer. The resources: $R_i = P_i$, $1 \le i \le m$, have been added to the additional resources to describe the master slave construction during execution of the communication tasks. Resource R_{m+1} accounts for the k buses.

The resource demand is determined by the assignment of the tasks. From the assignment of the plu-task(u), $u \in V$, the assignment of all other tasks is determined.

$F_A(\text{com-task}(u)) = F_A(\text{plu-task}(u)) \quad u \in V$

$F_A(\text{add-task}(u)) = F_A(\text{plu-task}(v)) \quad (u,v) \in B \text{ and } \gamma(u) < \gamma(v)$

$F_A(\text{pbs-task}(u)) = F_A(\text{plu-task}(u)) \quad u \in V$

$F_A(\text{com-pbs-task}(u)) = F_A(\text{plu-task}(u)) \quad u \in V$

The demand for the additional resources is determined by F_A. The resource demand of any task $p = T' \cup T$ is given by:

- if the considered task p is any plu-task(u) or pbs-task(u) then

$$r(R_\ell, p) = \begin{cases} 1 \text{ for } R_\ell \in \{F_A(p)\} \\ 0 \text{ otherwise} \end{cases}$$

- if the considered task p is any com-task(u) with $(u,v) \in B$ and $\gamma(u) < \gamma(v)$ then

$$r(R_\ell, p) = \begin{cases} 1 \text{ for } R_\ell \in \{F_A(p),\ F_A((\text{plu-task}(v)),\ R_{m+1}\} \\ 0 \text{ otherwise} \end{cases}$$

- if the considered task p is any com-pbs-task(u) then

$$r(R_\ell, p) = \begin{cases} 1 \text{ for } R_\ell \in \{F_A(p),\ R_{m+1}\} \cup \{F_A(\text{plu-task}(v)) \mid u \text{ is the successor of } v\} \\ 0 \text{ otherwise} \end{cases}$$

4.3 Scheduling of the solution job

After the scheduling model as stated in section 2.3 has been completed the actual scheduling can start. The problem is to determine the two mappings $F_A : T \cup T' \rightarrow P$ and $F_I : T \cup T' \rightarrow I$, the assignment of tasks to resources and to time intervals respectively, such that all constraints are satisfied, such that the object function, the schedule length ω, is minimized.

The above schedule problem is divided into two parts: the lu-job scheduling and the bs-job scheduling.

Attention will be paid only to the lu-job scheduling. The assignment F_A is determined by minimizing the schedule length ω_{lu} instead of $(\omega_{lu} + \omega_{bs})$. The lower script indicates to which job the ω belongs. This may result in a non optimal schedule for the lu-job and bs-job together. But a near optimal schedule is assumed because ω_{lu} is minimized and ω_{bs} is much smaller than ω_{lu}. Namely, consider the case where the number of processors is large enough to exploit all parallelism, then the critical path length is a good estimate for the schedule length. From the equations (4.2.8) and (4.2.10) it may be concluded that $\omega_{bs} << \omega_{lu}$.

The resource demands and the duration of the tasks depend on the task assignment (general schedule model). Already without this depen-

dency, in the case that the communication aspects would be neglected, the stated scheduling problem belongs to the class of NP-complete problems [4.1]. To keep the required preprocessing within acceptable limits heuristics will be used to determine a near optimal schedule. The schedule which will be considered belongs to the class of non-preemptive list schedules [2.3].

4.3.1 Nonpreemptive list schedules

The performance of list schedules is in general quite adequate and the computational complexity is polynomial bounded. First an operational description of the list scheduling will be given for the augmented basic model. According to some strategy an ordering $\pi : \{1,\ldots,|T|\} \to T$ is determined. The tasks are inserted into a list, denoted by L, according to the ordering $L = (\pi(1),\pi(2),\ldots,\pi(|T|))$. "Scanning" the list L means that the tasks are inspected one by one in the sequence imposed by the ordering.

A task $u \in T$ with predecessor tasks $\{u_1,u_2,\ldots,u_k\}$ is free at time t if $\delta(u_i) \leq t$ for $u_i \in \{u_1,\ldots,u_k\}$.

The mappings F_I and F_A will be constructed. The mappings under construction are distinguished by a bar : $\bar{F}_I$ and $\bar{F}_A$.

Consider the assignment to the time intervals $\bar{F}_I : T \to I$; task $u \in T$ will be called "covered" if $\bar{F}_I(u) \neq \phi$ otherwise this task u will be called "uncovered". If all $u \in T$ are covered then F_I will be called "complete", otherwise it will be called "partial".

The same notations hold for $\bar{F}_A$. If $\bar{F}_A(u) = P_i$ then task u is covered by P_i. The functions $\bar{f}_p$ and $\bar{f}$ are derived from (2.3.3) and (2.3.4) by replacing F_I and F_A by $\bar{F}_I$ and $\bar{F}_A$ respectively. Assume the partial mappings $\bar{F}_A$ and $\bar{F}_I$. A task u will be called "ready" at time t if:

- u is free and uncovered by $\bar{F}_I$
- $\forall_{R_i \in R}[r(R_i,u) \leq rm(R_i) - \sum_{v \in \bar{f}(t)} r(R_i,v)]$

A processor P_i will be called "idle" at time t if $\bar{f}_p(t,P_i) = \phi$. The $\bar{F}_A$ and $\bar{F}_I$, which are determined by the procedure "list schedule", represent the required schedule.

Relatively much is known about list schedules for the basic schedule model. Some results which are important for the purpose here are cited.

```
procedure list schedule;
begin
comment schedule will be represented by F_A = F̄_A and F_I = F̄_I
t ← 0; ∀_{u ∈ T}[F̄_A(u) ← 0; F̄_I(u) ← φ;]
while (F̄_I partial) do
    begin
    while ((# idle processors ≠ 0) ∧ (# ready tasks ≠ 0)) do
        begin
        u ← scan L for a ready task;
        P_i ← arbitrary idle processor;
        F̄_A(u) ← P_i;
        F̄_I(u) ← [t,t + (τ(P_i,u)));
        end;
    t ← min({δ(w) | F̄_A(w) ≠ φ ∧ (δ(w) ≥ t)});
    end;
end;
```

Consider two identical schedule models A and A' which are equal except for one of the following items: the list L, the precedence relations $\vec{S}$, the task durations τ and the number of processors m. The items of the two models are distinguished by a dash if necessary. Formula (4.3.1) [4.2] gives an upper bound by which the ratio ω'/ω may change.

$$\frac{\omega'}{\omega} \leq 1 + \frac{m-1}{m'} \tag{4.3.1}$$

Sometimes the result is counter intuitive namely even if $m' > m$ or $\vec{S}' \subset \vec{S}$ or $\tau'(u) \leq \tau(u)$, for $u \in T$, then a schedule length $\omega' > \omega$ may result. (Scheduling anomalies). Consider the case where model A has a list which produces the minimal ω, to be denoted by ω_0 and where A' has an arbitrary list L. Formula (4.3.2) [4.2] illustrates the necessity to pay attention to the construction of the list L.

$$\frac{\omega'}{\omega_0} \leq 2 - \frac{1}{m} \tag{4.3.2}$$

Consider a task system where the task graph is an in-tree. For each task $u \in T$ a "level", denoted by $\ell(u)$, will be defined. The level $\ell(u)$ is equal to the sum of the duration of the task itself and all the tasks on the path to and the root task. Now a mapping

π: $\{1,\ldots,|T|\} \rightarrow T$ is determined such that $\ell(\pi(i)) \geq \ell(\pi(j))$, for $i < j$. List schedules with a list determined by the levels as in the above way, are called level schedules.
If the task durations are equal the level schedule will be optimal [4.3]. If the task durations are not equal a schedule length ω will result which in general is not optimal. An upper bound for the ratio between ω and ω_0 is given by [4.4].

$$\frac{\omega}{\omega_0} \leq 1 + (m-1).\max(\{\tau_1,\ldots,\tau_n\}) / \sum_{i=1}^{n} \tau_i \qquad (4.3.3)$$

The bound given by (4.3.3) is only of importance if the number of processors m is not large enough to obtain the minimal schedule length.

4.3.2 Determination of F_A and F_I.

To obtain a near optimal schedule the problem of finding the two mappings F_A and F_I will be treated separately, although they are not independent of each other. (The lower script lu in ω_{lu} is deleted because only the lu-job is considered).
The job system in case of the ideal parallel computer without the additional resources and with task durations given by eq.(4.2.2) will be used to this purpose. Hence the possible update conflicts and the effects due to the data distribution are out of consideration at this stage. The scheduling model reduces to the basic model. The items belonging to this model will be indicated by the superscript b if necessary.
From the mapping F_A^b the mapping F_A may be derived according to the rules stated in section 4.2.4. On the general schedule model stated in section 4.2 this F_A will be imposed as a constraint. This means only schedules with F_A determined by F_A^b will be allowed. From the task system the tasks with duration zero will be deleted.The scheduling problem is reduced to finding the mapping F_I for an augmented schedule model. The items of this model will be referred to without a superscript.
For the purpose at hand, a mapping F_A^b will be called (near) optimal if with F_A being derived from F_A^b also a (near) optimal F_I can be obtained. The quality of the resulting schedule may be measured by the

ratio given by (4.3.4) if the ratio given by (4.3.5)

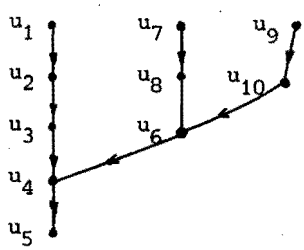
$$\frac{(\omega - \omega^b)}{\omega} \tag{4.3.4}$$

is known to be small. Because of the task graph being a tree and, if level scheduling is applied an estimate for (4.3.5) is given by (4.3.3).

$$\frac{\omega^b - \omega_0^b}{\omega_0^b} \tag{4.3.5}$$

Nonpreemptive list schedule for the basic schedule model. The list is determined by the task levels because the task graph is an in-tree. The level schedule strategy is performed by the procedure list schedule, where the schedule model is the basic schedule model of section 4.1, and the two mappings F_A^b and F_I^b are generated. The mapping F_A which is derived from F_A^b is imposed as a constraint on the schedule model of section 4.2 by which it reduces to an augmented schedule model.
Again the nonpreemptive level schedule strategy will be applied to determine the mapping F_I. The procedure list schedule, where the schedule model is the augmented model, generates the mapping F_I.

The whole scheduling process is applied to the sparse matrix example in order to illustrate the scheduling; see fig.(4.3-1).



u_i = plui-task(v_i) for $0 < i \leq 10$

Fig.(4.3-1a) The task graph $(T^b, \vec{S}^b)$ for the lu-job.

u_i	u_1	u_2	u_3	u_4	u_5	u_6	u_7	u_8	u_9	u_{10}
$\tau 1(u_i)$	10	21	10	3	0	10	10	10	10	10
$\ell(u_i)$	44	34	13	3	0	13	33	23	33	23
$\pi^{-1}(u_i)$	1	2	7	9	10	8	3	5	4	6

Fig.(4.3-1b) Table with input for list schedule on basic model, where $\tau_{add} = \tau_{mul} = \tau_{div} = 1$ unit.

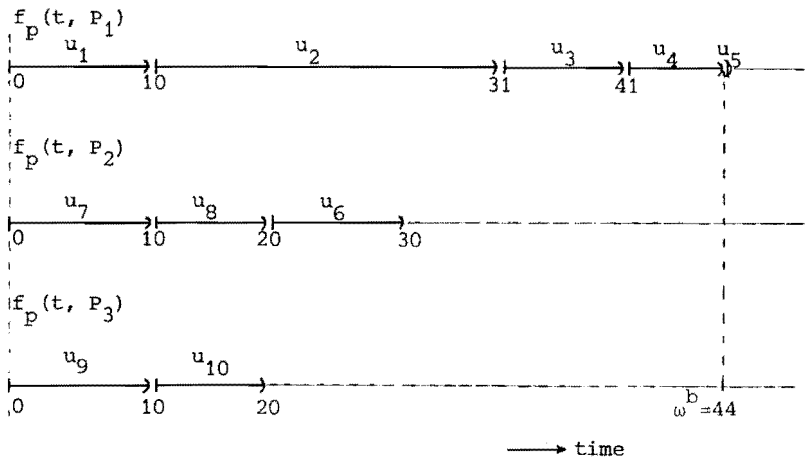


Fig.(4.3-1c) Gantt charts for the basic scheduling.

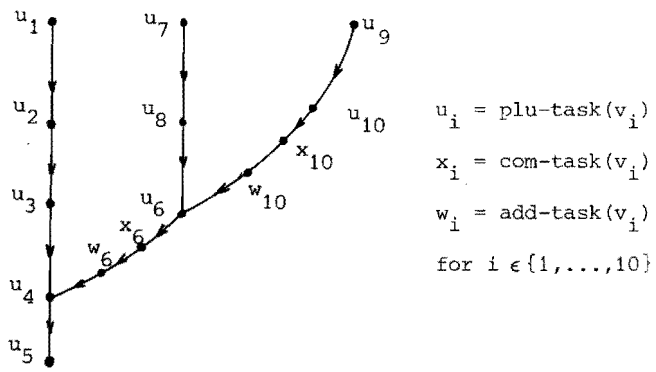


Fig.(4.3-1d) Modified task graph $(T,\vec{S})$.

y_i	u_1	u_2	u_3	u_4	u_5	u_6	u_7	u_8	u_9	u_{10}	x_6	x_{10}	w_6	w_{10}
$F_A^b(y_i)$	P_1	P_1	P_1	P_1	P_1	P_2	P_2	P_2	P_3	P_3				
$F_A(y_i)$	P_1	P_1	P_1	P_1	P_1	P_2	P_2	P_2	P_3	P_3	P_2	P_3	P_1	P_2
$\tau(y_i)$	10	21	10	3	0	8	9	10	9	7	2	2	4	3
$r(R_1,y_i)$	1	1	1	1	1	0	0	0	0	0	1	0	1	0
$r(R_2,y_i)$	0	0	0	0	0	1	1	1	0	0	1	1	0	1
$r(R_3,y_i)$	0	0	0	0	0	0	0	0	1	1	0	1	0	0
$r(R_{bus},y_i)$	0	0	0	0	0	0	0	0	0	0	1	1	0	0
$\ell(y_i)$	44	34	13	3	0	17	36	27	38	29	9	22	7	20
$\pi^{-1}(y_i)$	1	4	10	13	14	9	3	6	2	5	11	7	12	8

Fig.(4.3-1e) Table with input for list schedule on augmented model, where $\tau_{com} = 0.5$ units and $\tau_{overh} = 0.$ units.

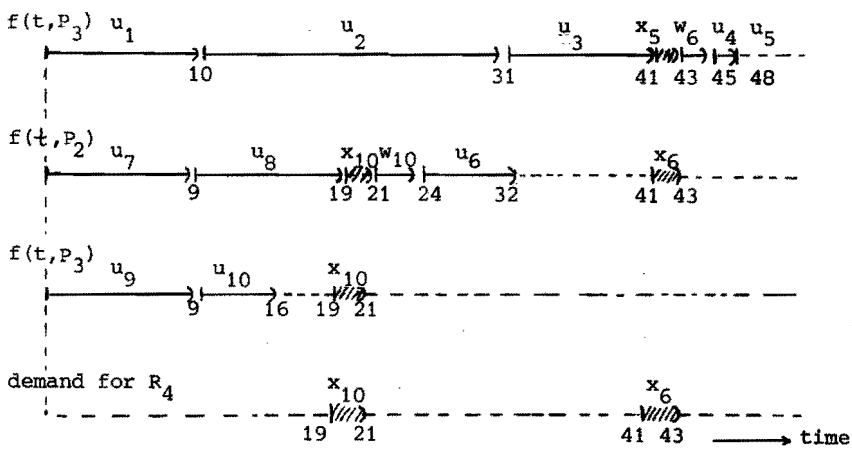


Fig.(4.3-1f) Gantt charts for the resulting schedules.

4.3.3 Modification on the strategy to determine F_A

The schedule length ω is in general larger than ω^b due to the introduced communication tasks, modified task duration and the resource demands of tasks; the difference $(\omega - \omega^b)$ will be called the "communication overhead". The strategy which was pointed out in section 4.3.2 may be improved considerably if during the construction of

$\bar{F}_A^b$ attention is paid to the communication overhead which may be introduced. The overhead will be reduced in two ways:

- "local strategy". If during determination of F_A^b the predecessor tasks have been assigned to different computers then it will be tried to minimize the overhead by a proper assignment of the successor task.
- "global strategy". The number of communication tasks which might become active will be reduced.

Local strategy.

During the list scheduling to obtain F_A^b the assignment of a ready task is done to an arbitrary processor if there are more idle processors. The schedule length ω^b is independent of which processors will be chosen. However the resulting schedule length ω will be dependent of which idle processor has been chosen, even if durations of the communication tasks are neglected.

In case of sufficient many processors the critical path length of $(T,\vec{S})$ is a measure for ω. The dependency of the assignment for the critical path length is demonstrated in the following.

Consider a vertex u with predecessors $\{w_1,\ldots,w_n\}$ and a successor v. Let $v \in U_i$. Assume all vertices given by $V\backslash V(u)$ have already been assigned to some processor. The level of plu-task(v) does not depend on the assignment of any vertex $x \in V(u)$ because the duration of the tasks does not depend on the assignment of the predecessors, see equations (4.2.7) and (4.2.8).

If $u \in U_i$ then the level of plu-task(u) becomes: ℓ(plu-task(v)) + + τ(plu-task(u)), which is given by:

$$\ell(\text{plu-task}(u)) = \ell(\text{plu-task}(v)) + \tau 1(u) + \sum_{w=w_1}^{w_n} |\text{madj}(w,r_i)|^2 \tau_{add} - |\text{madj}(u,r_i)|^2 \tau_{add} \tag{4.3.6}$$

If $u \notin U_i$ then the level of plu-task(u) becomes: ℓ(plu-task(v)) + + τ(add-task(u) + τ1(plu-task(u), which is given by:

$$\ell(\text{plu-task}(u)) = \ell(\text{plu-task}(v)) + (|\text{madj}(u)|^2 - |\text{madj}(u,r_i)|^2)\tau_{add} + \tau 1(u) + \sum_{w=w_1}^{w_n} |\text{madj}(w,u)|^2 \tau_{add} - |\text{madj}(u,u)|^2 \tau_{add} \tag{4.3.7}$$

In case of $u \notin U_i$ the level of plu-task(u) is increased by the amount given by:

$$\text{change in level} = \sum_{w=w_1}^{w_n} (|\text{madj}(w,u)|^2 - |\text{madj}(w,r_i)|^2)\tau_{add} \qquad (4.3.8)$$

The critical path is equal to the highest level. If plu-task(u) lies on the critical path, then the critical path length is increased by the above amount.

The critical path length does not take into account the resource demands some tasks are forced to be executed one after another, this in contrast to the precedence relations. Assume all tasks on the critical path in the task graph of the augmented schedule model are assigned to the same computer C_r and all other tasks to computers C_i, for $i \neq r$ and $1 \leq i \leq m$. Let U_j be the cluster assigned to C_r. The tasks add-task(u) for $u \in F_j$ will be assigned to C_r, hence this computer requires an extra amount of processing time given by:

$$\sum_{u \in F_j} (|\text{madj}(u)|^2 (\tau_{com} + \tau_{add}) + \tau_{overh}) \qquad (4.3.9)$$

This amount will be indicated as "addition overhead". The schedule length ω is the sum of the critical path length in $(T,\vec{S})$ and the addition overhead.

The local strategy tries to reduce the addition overhead.The critical path length in $(T,\vec{S})$ may be increased.
Assume at time t the ready task u is obtained from the list scan. Let PI(u) denote the predecessor tasks of u which have been assigned to one of the processors which are idle at time t. If PI(u) is empty, select an arbitrary processor out of the idle processors.Otherwise select the task $v \in PI(u)$ which is finished first, and take the idle processor given by $\bar{F}_A(v)$.
Line 9 in procedure list schedule is replaced by:

9.1 if $(PI(u) = \phi)$ then pick some $P_i \in$ {idle processors} else pick some
9.2 $P_i \in \{\bar{F}_A(v) \mid v \in PI(u) \wedge (\delta(v) = \min(\{\delta(w) \mid w \in PI(u)\}))$;

This criterion will be called "switch criterion" because consecutive tasks which belong to a critical path of a (sub)-tree tend to be assigned to distinct computers if the task has more than one predecessor

task.

Fig.(4.3-2) shows the influence of the switch criterion on the schedule by means of a simple example.

At time instant t^* during construction of F_A^b the plu-task$(u_4)^b$ must be assigned while all three computers are idle. Due to the switch criterion plu-task$(u_4)^b$ will be assigned to P_3. This because of P_2 and P_3 being expected to be idle, apart from communication tasks and add-task, in the schedule during the time intervals (δ(plu-task(u_2)), δ(plu-task(u_1))) and (δ(plu-task(u_3), δ(plu-task(u_1))).

The longest interval is chosen and will be used to perform the data exchange and add-tasks as much as possible. In the given example this is possible, hence the only overhead is due to the com-task(u_1).

Fig.(4.3-3) shows the scheduling process with the switch criterion for the sparse matrix example. The communication overhead is reduced by two time units.

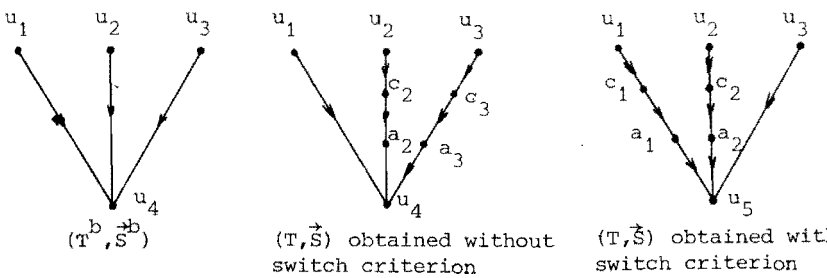


Fig.(4.3-2a)

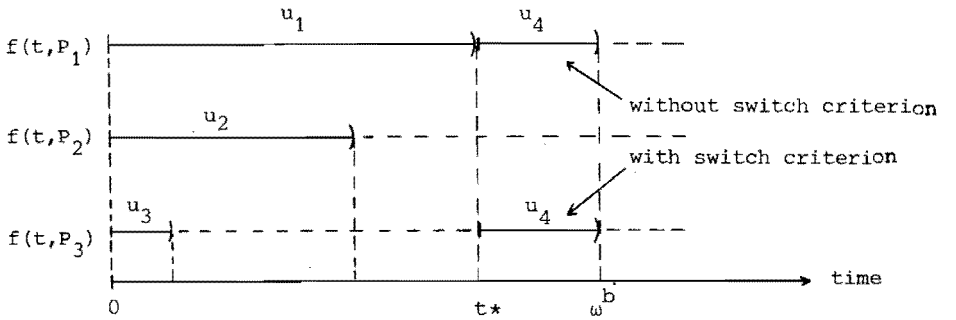


Fig.(4.3-2b) Gantt charts for schedule of the basic model.

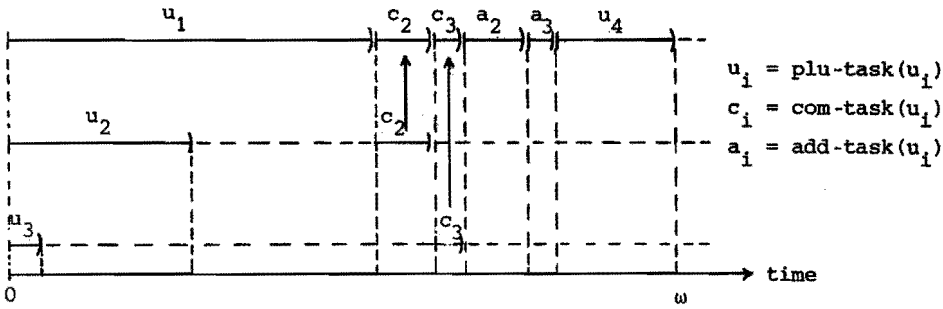


Fig. (4.3-2c) Gantt charts for schedule without switch criterion.

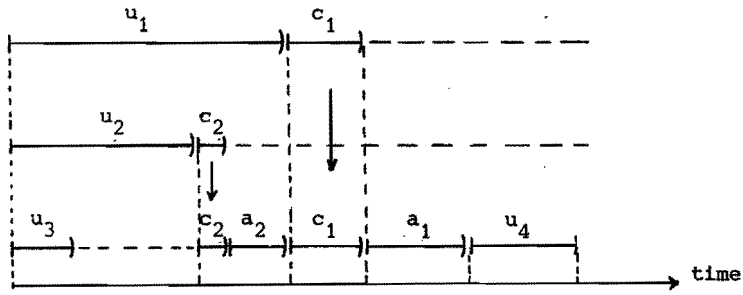


Fig. (4.3-2d) Gantt charts for schedule with switch criterion.

fig. (4.3-2) Example to demonstrate the influence of the switch criterion

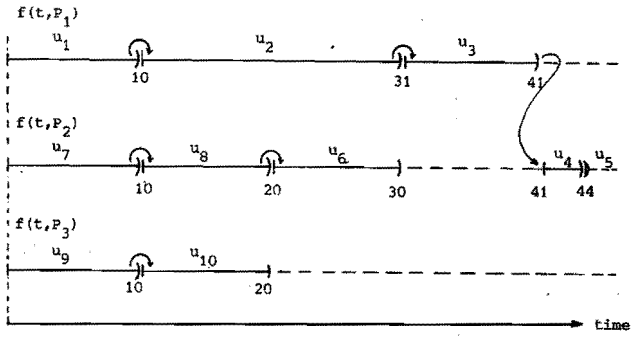


Fig. (4.3-3a) Gantt charts for the basic schedule with "switch mode".
Task system is given in fig. (4.3-1a,b)

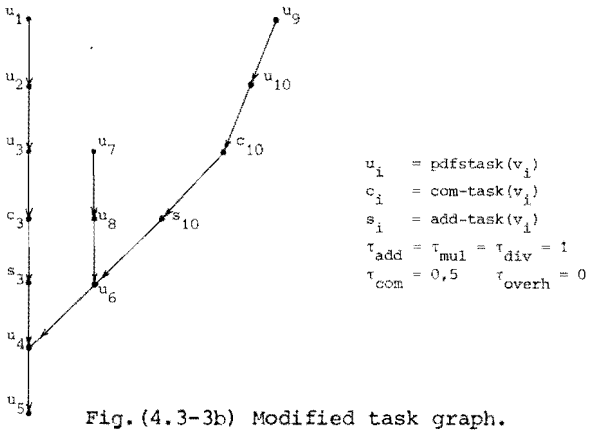


Fig. (4.3-3b) Modified task graph.

v	u_1	u_2	u_3	u_4	u_5	u_6	u_7	u_8	u_9	u_{10}	c_3	c_{10}	s_3	s_{10}
$F_A^b(v)$	P_1	P_1	P_1	P_2	P_2	P_2	P_2	P_2	P_3	P_3	-	-	-	-
$F_A(v)$	P_1	P_1	P_1	P_2	P_2	P_2	P_2	P_2	P_3	P_3	P_1	P_3	P_2	P_2
$\tau(v)$	10	17	10	3	0	10	10	10	9	7	2	2	4	4
$r(R_1,v)$	1	1	1	0	0	0	0	0	0	0	1	0	0	0
$r(R_2,v)$	0	0	0	1	1	1	1	1	0	0	1	1	1	1
$r(R_3,v)$	0	0	0	0	0	0	0	0	1	1	0	1	0	0
$r(R_{bus},v)$	0	0	0	0	0	0	0	0	0	0	1	1	0	0
$l(v)$	46	36	19	3	0	13	33	23	35	26	9	19	7	17
$\pi^{-1}(v)$	1	2	7	13	14	10	4	6	3	5	11	8	12	9

Fig. (4.3-3c) Table with input for list schedule on augmented model.

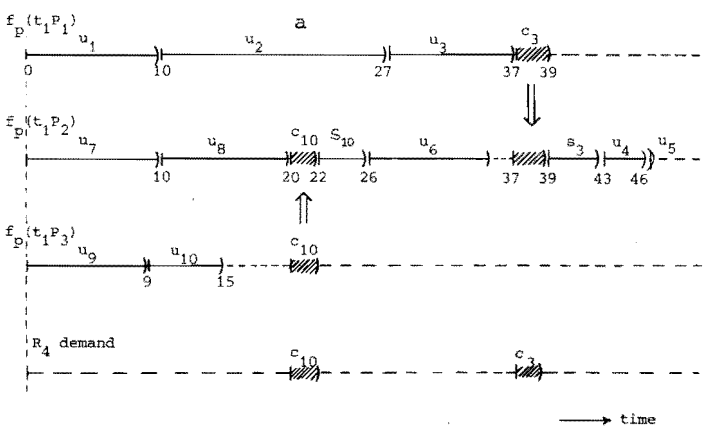


Fig. (4.3-3d) Gantt chart.

Global strategy.

By means of the mapping F_A^b and the e-tree (V,B) the clusters U_i have been determined for $1 \le i \le p$. The number of communication tasks (com-tasks) is (p-1), hence the number of clusters must be minimized. Consider the proposed procedure to determine F_A^b. During execution of the "while"-statement, lines 6-12, the next ready task u is selected by a scan of the list L.

If the switch criterion is not used the selected task will be assigned to an arbitrary idle processor P_i. So no attention is paid to the question whether it would have been possible to extend some cluster or not.

If the switching criterion is used, there are two possibilities:

- $PI(u) \neq \phi$, in this case the task u will be assigned to $P_i \in \{\bar{F}^b(w) \mid w \in PI(u)\}$ which leads to the extension of a cluster.
- $PI(u) = \phi$, in this case an arbitrary processor is chosen which gives rise to a new cluster.

Due to the above observations it may be expected that the determined F_A^b will generate many clusters if during the list scheduling the num-

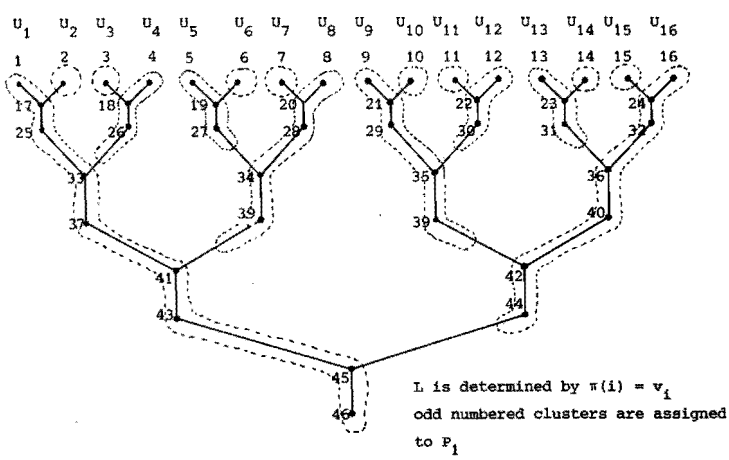


Fig.(4.3-4) Illustration of a poor clustering if the number of ready tasks is larger than the number of processors.

ber of ready tasks is large compared to the number of processors. Consider a task system where all tasks have an equal duration and the task graph as given in fig.(4.3-4). In this figure the assignment of the tasks is given for the case of two processors. The number of resulting clusters is far from minimal.

In general the number of processors is not large enough to exploit all parallelism which is in the task tree. In the beginning of the construction of F_a^b by the level scheduling the number of ready tasks is far larger than the number of processors which results in many small clusters and hence many communication tasks.

In order to avoid 'unnecesssary communication tasks' as much as possible the task system will be derived from a cluster graph $(C,\bar{B})$ instead of directly from the e-tree itself. The tasks are now associated with the clusters C_i, for instance
plu-task(C_i) = {plu-task(u) | $u \in C_i$}, for $1 \le i \le h$.
The number of clusters U which result after determination of the assignment is reduced but precaution must be taken that the schedule length ω^b will not be increased.

To obtain a suitable partitioning of vertices into clusters the method, which was proposed in section 3.4, will be used. The cluster S(v) will be determined by a strategy which is accomplished by procedure "trunk(V,B)".
To this purpose a weight will be determined for each vertex $u \in V$. The weight of u, denoted by w(u), is given by:

$$w(u) = \sum_{x \in V(u)} \tau(x) \qquad (4.3.10)$$

w(u) is equal to the duration of all plu-tasks which are associated with V(u). The weight w(u) is called the "workload" of V(u).
The procedure trunk constructs a chain from the root of (V,B) to some vertex v. The vertices of this cluster are denoted by S(v). Let u denote the predecessor of v with the largest weight. Unless the process is terminated the chain is extended with u and the process is repeated with v = u.
The process is terminated in two ways:
- the selected predecessor u of v is a top vertex. This stop criterion is not likely to occur.
- the selected predecessor u of v is not the vertex with the largest

weight in the set adj(S,B).

The procedure trunk results in the cluster S(v). The set adj(S(v),B) = $\{r,\ldots,r_h\}$ defines a set of clusters $\{V(r_1),\ldots,V(r_h)\}$. The cluster S(v) will be called the "trunk" cluster and $V(r_i)$ will be called a "leaf" cluster, for $1 \le i \le h$. The procedure trunk may be used again for each leaf cluster until all leave clusters have a weight smaller than a given value; called "max". This recursive process will be called "clustering".

```
procedure trunk(V,B);
begin
  comment given a graph (V,E), |V| ≥ 2, and its e-tree (V,B),which
  is not a chain, a cluster S will be generated such that S is a
  separator of (V,E). pred(x) denotes the predecessor set of x in
  (V,B).
x ← r; S ← φ; max branch ← 0;
y←arbitrary vertex of {z | z∈pred(x) ∧ w(z) = max({w(u) | u∈pred(x)})};
while ((w(y) > max branch) and (pred(x) ≠ φ)) do
    begin
    S ← S ∪ {x};
    max branch ← max ({w(z) | z ∈ adj(S,B) \ {y}});
    x ← y;
    y arbitrary vertex of {z|z∈pred(x)∧w(z) = max({w(u) |u∈pred(x)})};
    end;
end;
```

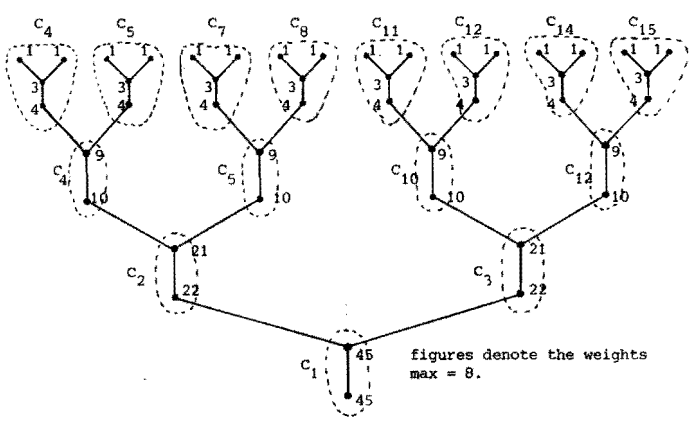


Fig.(4.3-5) Shows the result of the clustering applied on the task graph of figure (4.3-4).

Figure (4.3-5) shows the result of the clustering where max is set equal to eight task durations.
In case of a nice tree structure (the critical path length << weight of the root and a symmetric structure), such as given by fig.(4.3-4), the procedure trunk constructs a trunk with a number of leaves which have equal weights. Although, in practice a nice tree structure will be rare it may be expected that the leaves with the largest workload have approximately the same workload.
The critical path of the task graph derived from the cluster tree $(C,\bar{B})$ may be lengthened compared to the task graph which has been derived directly from the e-tree due to two effects:
- the parallelism which is in the leaf clusters is neglected. The increase of the critical path length is bounded by the parameter max. However this parallelism would also be lost because the number of available processors is too small to exploit all parallelism at the highest levels. The parameter max should be chosen such that the number of large leaf clusters is of the same order as the number of processors to avoid an unnecessary schedule length increase.
- the trunk clusters are not responsible for any increase because the tasks of these clusters are already forced to be executed sequentially due to the precedence relations. In some cases increase of the critical path may occur due to the precedence relations among the clusters which are more severe, see fig.(4.3-5). If an edge (u_6,u_{11}) is added to (V,B) the same precedence relations as in the cluster graph are obtained.

The resulting assignment can also be thought to be obtained directly from the e-tree. A set of edges is added to B to account for the sequence of the clusters. Further, if execution of the first task of a cluster is started, all other tasks of the cluster must be processed without delay by the same processor, which can be accomplished by an appropriate list. Sometimes a shorter schedule may be obtained which is due to the already stated anomalies.

The resulting strategy to determine F_A consists of the clustering with a proper value for the parameter max followed by the list scheduling.

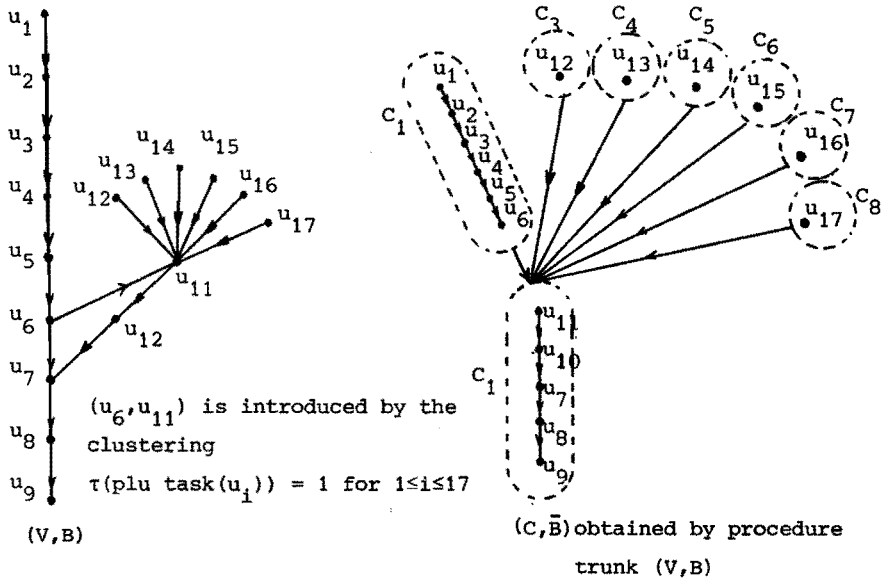


Fig.(4.3-5) Increase of the critical path length due to clustering.

4.3.4 Modification on the strategy to determine F_I

Sometimes it may be favourable to interrupt a task, which is no communication task, by a communication task. This will be called "interrupt mode".

5. RESULTS

5.0 Introduction

In this chapter some results concerning the parallel processing of the lu-job are presented and discussed. These results are obtained by a simulation of the asynchronous array computer.
The results concern mainly the associated graph of the MNA matrix which is derived from the electronic circuit μA758 [5.1], shown in fig. (5.1-1). The circuit is thought to be representative for the class of electronic circuits which are going to be simulated on the asynchronous array computer.

5.1 e-tree results

The associated graphs of the circuits considered are directly obtained from the topology of the given circuit where the bipolar transistors are replaced by the Ebers Moll model.
Besides the associated graph of the μA758 the associated graphs of two power networks are considered, the standard power network AEP 118 [5.2] and a large power network USNET [5.3].
Two triangulation criteria are used: Berry's criterion [3.8] and the minimum degree criterion [3.9].
Table (5.1.1) shows the main characteristics of the investigated associated graphs and e-tree results. The first example will be referred to as the "example" in the sequel. The "sparsity factor", the ratio given by the number of vertices and the number of classes, indicates the parallelism due to the sparsity. The total workload and the critical path length are determined with help of (4.2.2). The duration of the parameters τ_{mul}, τ_{div} and τ_{add} are all equal to one time unit. (Due to this choice the operations count and the duration of each task except for the communication tasks have the same value). The speedup, the ratio between the total workload and the critical path length, varies between 3.20 and 5.14.
Table (5.1.2) gives the cardinalities of the label classes and workload respectively for the example. Table (5.1.2) shows that after 8 classes have been eliminated an almost full matrix remains which is 16 x 16 representing a workload of 1398 units. That is about 22% of the total workload.

Figure (5.1-2) shows the e-tree for the example.

5.2 Clustering results

Figure (5.2-1) shows the cluster graphs of the e-tree for the example, which are obtained by the clustering with parameter max equal to a half, a quarter and an eighth of the total workload. For each cluster C_i, the root r_i, the duration of the plu-task(C_i) and $|madj(r_i)|$ is given.

Figures (5.2-2) and (5.2-3) show the partitions of the circuit for max is equal to a half and a quarter of the total workload respectively. Only those nodes which are associated with the vertices of the leaf clusters are shown.

Finally, the histograms in fig.(5.2-4) show the distribution of the workload over the leaf clusters which are obtained for the different values of parameter max. Only the workload of the leaf clusters is shown.

5.3 Results of scheduling

Some results of scheduling for the lu-job of the example considered are presented.

The schedule length ω depends on the parameters of the scheduling model. In the following $\omega = \omega(x_1,\ldots,x_p)$ denotes that ω is regarded as a function of the parameters $x_1,\ldots,x_p$. In the graphical representation of the considered functions the successive points of each function are connected by straight line pieces. In general the obtained curves will not be smooth due to the problem itself and anomalities.

Four schedule modes are determined by whether the switch criterion is used or not and whether interrupt is allowed or not. The modes are denoted by a boolean vector (x,y) where x is true if the switch criterion is used and y is true if interrupts are allowed. Let m_0 denote the number of processors necessary to obtain the optimal schedule length ω_0^b in the basic scheduling model. ω_0^b is equal to the critical path length of $(T^b,\vec{S}^b)$.

The influence of the schedule mode and τ_{com}.

Fig.(5.3-1) shows the following entities as a function of the number of computers for the specified parameters:

$\omega^b = \omega^b(m)$, the schedule length of the basic scheduling model.

$\omega = \omega(m)$, the schedule length for each schedule mode.

ω =total workload/m, the lower bound of the schedule length.

The parameters τ_{com} and τ_{overh} are chosen to be zero in fig.(5.3-1). The increase of the schedule length relative to the respective schedule length for $\tau_{com} = 0$ is denoted by $\Delta\omega$. Fig.(5.3-2) and fig. (5.3-3) show $\Delta\omega = \Delta\omega(\tau_{com})$ for each of the four schedule modes for a number of computers of 4 and 64 respectively. Hence, the schedule length for m=4 and m=64 can be obtained from the figures (5.3-1) and (5.3-2) for each of the values of τ_{com}.

The four modes of scheduling will be compared.

For small values of τ_{com} the $\Delta\omega = \Delta\omega(\tau_{com})$ is grossly the same for the different modes.

For large values of τ_{com} the $\Delta\omega = \Delta\omega(\tau_{com})$ differs significantly if $m > m_0$ depending whether the assign criterion is used or not.

To compare the $\omega = \omega(m)$ with $\tau_{com} \neq 0$ for the four schedule modes small and large values of τ_{com} are treated separately.

- small values of τ_{com}.

The relative position of the curves $\omega=\omega(m)$ for the different modes is given by fig.(5.3-1).

Two cases are distinguished:

$m < m_0$. The value of ω depends mainly on whether interrupt is allowed or not in favour of allowed interrupt.

$m > m_0$. The ω depends mainly on the switch criterion in favour of the switch criterion.

- large values of τ_{com}.

Two cases are distinguished:

$m < m_0$. The relative position of the curves $\omega = \omega(m)$ for the different modes is given by fig.(5.3-1). Again the value of ω depends mainly on whether interrupt is allowed or not in favour of allowed interrupt.

$m > m_0$. The relative position of the curves $\omega = \omega(m)$ for the different modes is no longer equal to the position given by fig.(5.3-1). The ω also depends mainly on the switch criterion, however, in favour of the mode without the switch criterion.

The difference between the schedules is determined by the communication overhead $\omega(m) - \omega^b(m)$, because ω^b does not depend on the applied modification.

The communication overhead consists of three parts:

1. -change of the task durations due to F_A. Fig. (5.3-4) shows the critical path length in $(T,\vec{S})$ for the mappings F_A determined with and without switch criterion as a function of the number of computers.

2. -tasks with a common successor task are excluded from being processed in parallel. Let ω_∞ denote the schedule length of a schedule with m/2 buses, allowed interrupt and $\tau_{com} = \tau_{overh} = 0$. For $m > m_0$ an estimate of this contribution is given by the difference between ω_∞ and the critical path length in $(T,\vec{S})$ where $\tau_{com} = \tau_{overh} = 0$. To this purpose $\omega_\infty = \omega_\infty(m)$ with and without switch criterion is shown for $m > m_0$ in fig.(5.3-4).

3. -communication tasks may introduce overhead by the duration of the tasks and by the time spent to wait until the necessary resources are available. Fig.(5.3-5) shows the increase of the critical path length in $(T,\vec{S})$ for both mappings F_A due to the com-tasks lying on this path as a function of the number of computers.

The influence of the switch criterion is significant for $m > m_0$ as shown in the figures (5.3-4) and (5.3-5).

For small values of τ_{com} the increase due to the contributions of the first and third part is more than compensated by a lower contribution due to the second part.

For large values of τ_{com} the expected idle time interval of the addressed computer is too small such that the assumptions of the switch criterion do not hold any longer. The com-tasks become part of the critical path (due to the resource demands). The number of com-tasks is for both modes the same, but in general the number of transmitted words will be larger in case of the switch criterion, which explains the curves in fig.(5.3-3). The number of words to be exchanged is expected to be larger because the communication occurs between the tasks lying on the critical path in (T^b,S^b).

The influence of the interrupt on the schedule length. If $m < m_0$ then at the beginning of the schedule for each computer there is a large number of ready tasks and there are com-tasks which would be ready if the required bus is available and if the addressed computer is idle. A ready task with a lower level will be started and when the addressed computer becomes idle it will start to execute also a ready task with a lower level. Hence, the com-tasks are blocked by tasks with a lower level. If $m > m_0$ the above process will be restricted

because the number of ready tasks will be small.

Fig.(5.3-6) shows the schedule length as a function of parameter max for the specified parameters. If the value of τ_{com} is small the clustering may be favourable if the number of computers is small compared to the number of tasks. However, if the value of τ_{com} is large even for a large number of computers the clustering may be favourable. In these cases only a part of the available computers will be used. If the size of the offered circuits varies, it will be necessary to determine for each job an appropriate value of parameter max. It is proposed to do this automatically by some heuristic formula, for instance: max = (total workload of the lu-job) / (2*m).

triangulated graph	number of vertices	number of edges	number of classes	dimension of class one	sparsity factor	total* operation count	critital* path length	speedup*
μA 758 (193v,358e) Berry	193	658	20	86	10	6234	1534	4.06
μA 758 (193v,358e) min degree	193	673	22	81	9	6583	1665	3.95
AEP 118 (118v,179e) Berry	118	264	17	48	7	1572	390	4.03
AEP 118 (118v,179e) min degree	118	265	15	49	8	1579	307	5.14
USNET (1637v,2237e) min degree	1637	4474	48	719	34	48938	15311	3.2

$^*\tau_{add} = \tau_{mul} = \tau_{div} = 1$

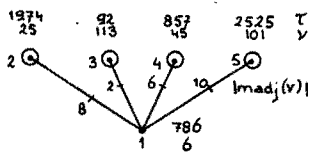
Table (5.1.1) Characteristics of the investigated associated graphs and the e-tree results.

CLASS	1	2	3	4	5	6	7	8	9	10	11	12	13-20
DIMENSION	86	38	16	12	9	7	5	4	2	2	2	2	1
WORKLOAD per class	-	-	-	-	-	-	-	-	360	276	342	272	308

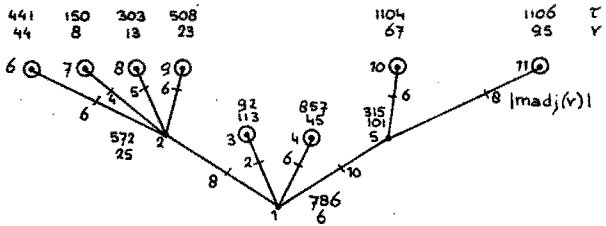
Table (5.1.2) Dimension of the label classes and workload of example 1.

Fig.(5.1-1) μA 758

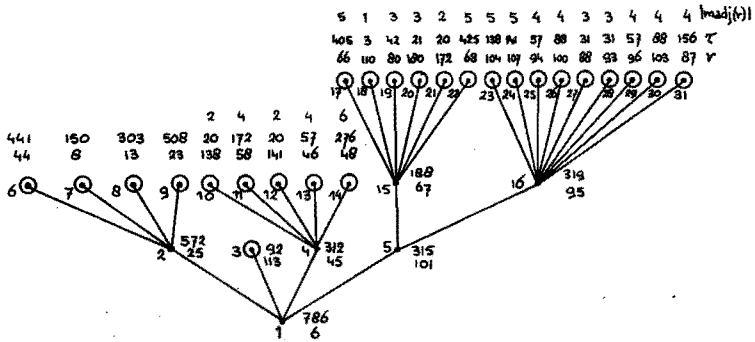
Fig.(5.1-2) e-tree (V,B)



(C,B̄) obtained by max = 3117



(C,B̄) obtained by max = 1558



(C.B̄) obtained by max = 779

Fig.(5.2-1) Cluster graphs for various values of max.

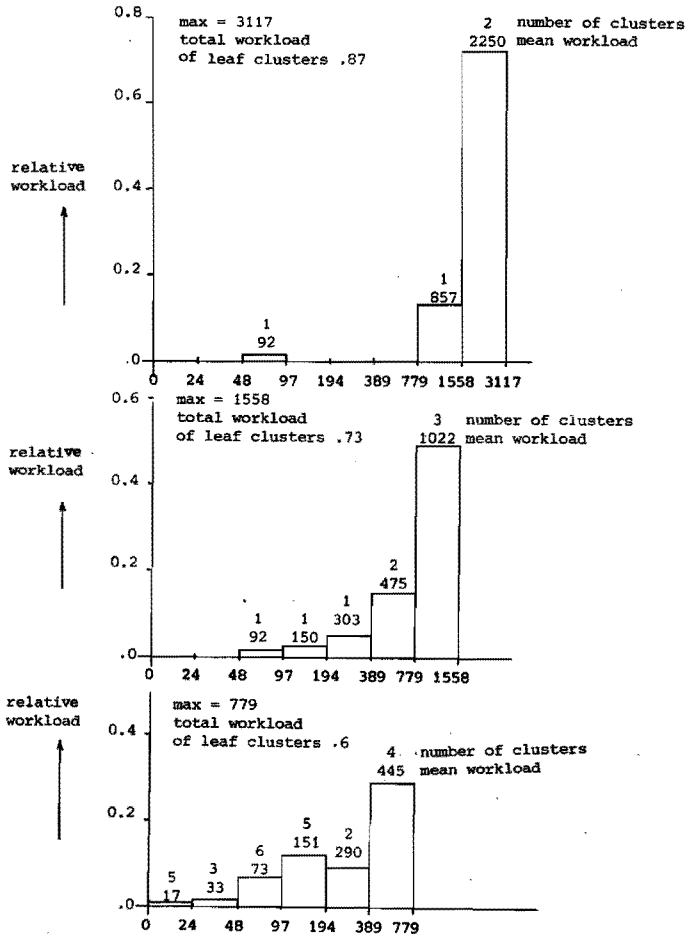


Fig.(5.2-4) Histograms workload for various values of max.

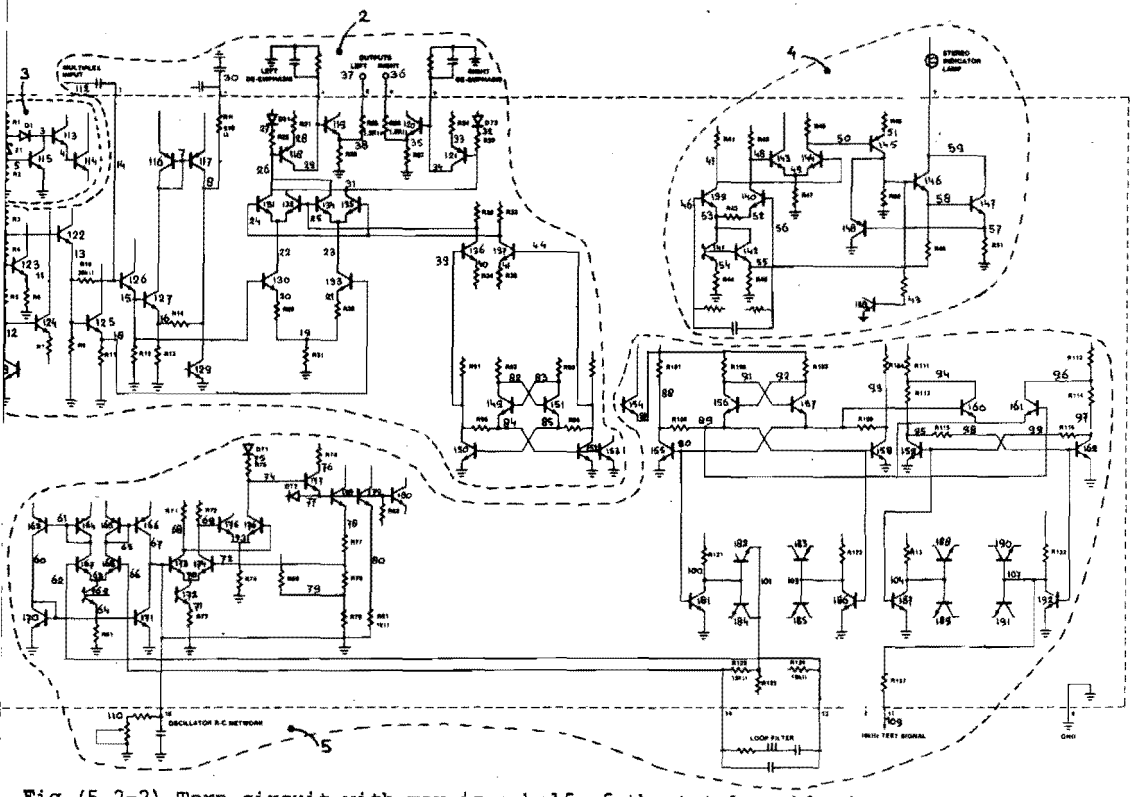


Fig.(5.2-2) Torn circuit with max is a half of the total workload.

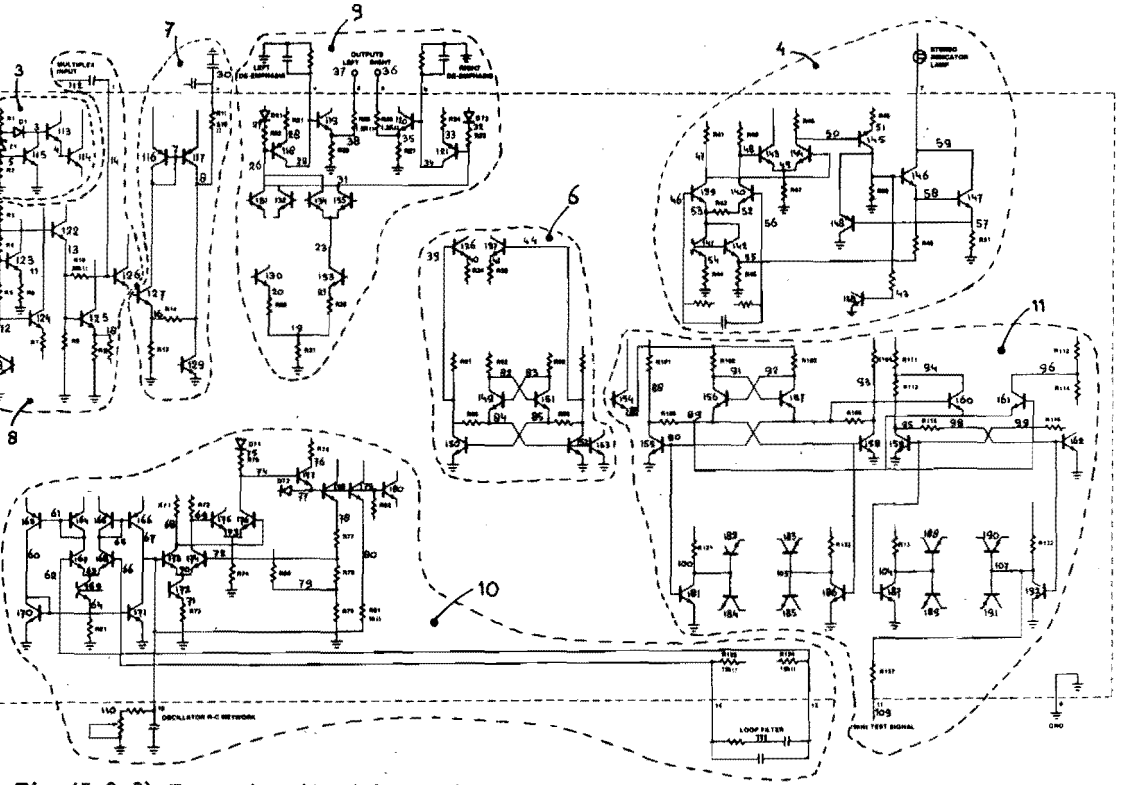


Fig.(5.2-3) Torn circuit with max is a quarter of the total workload.

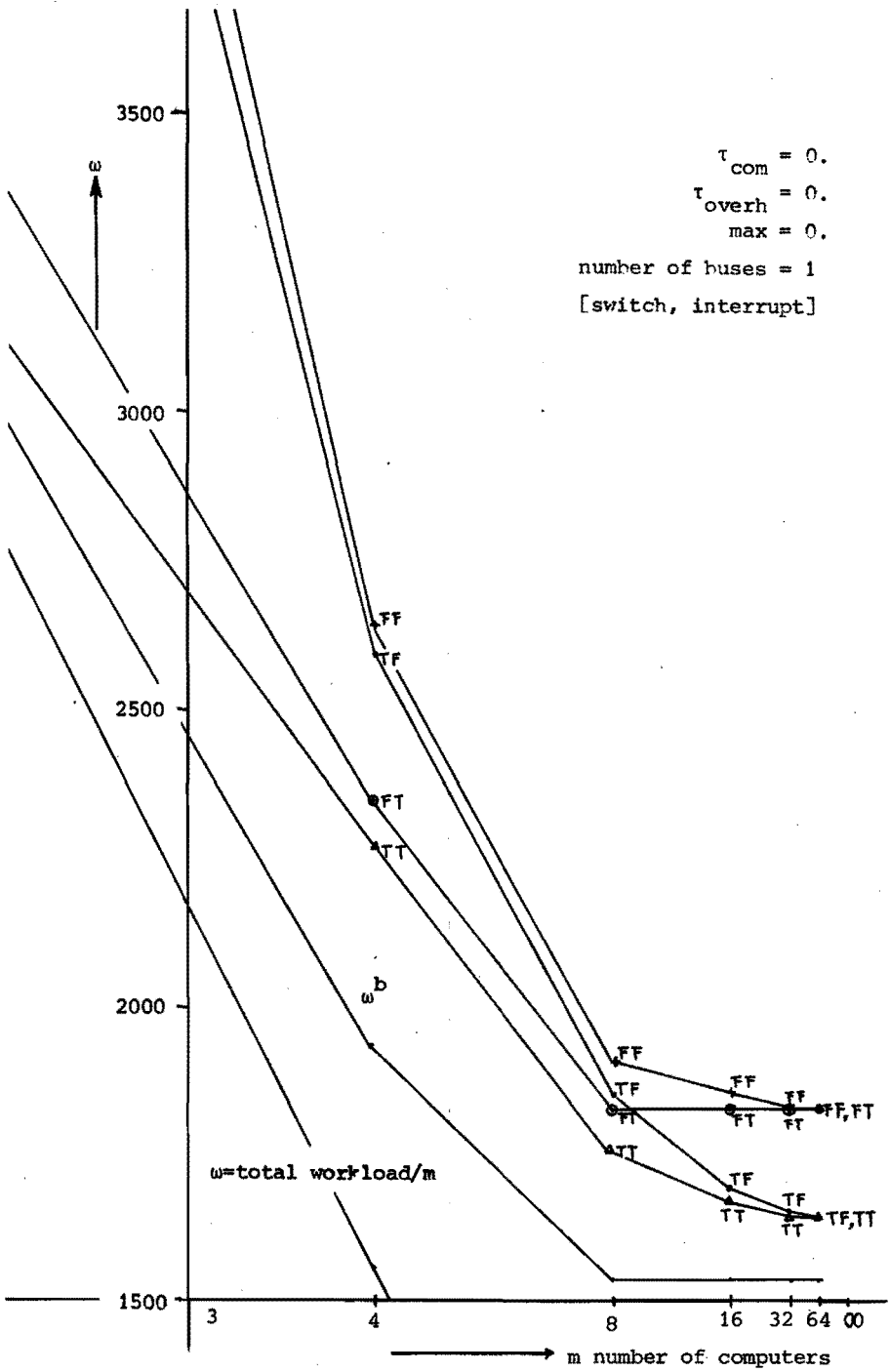


Fig. (5.3-1) $\omega = \omega(m)$

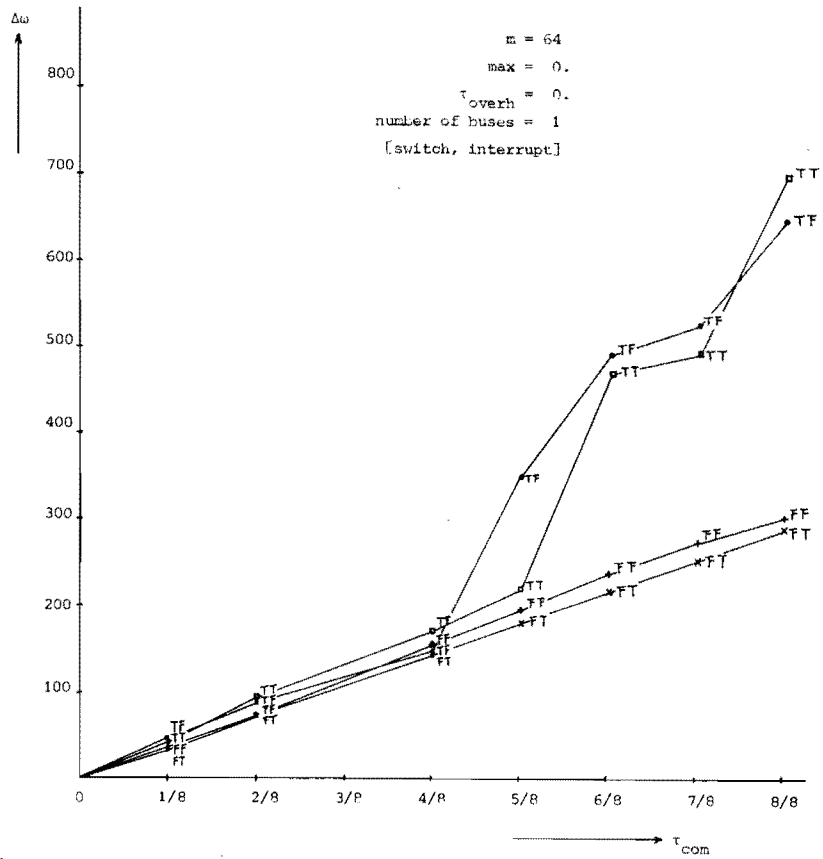


Fig.(5.3-2) Increase of the schedule length as a function of τ_{com}.

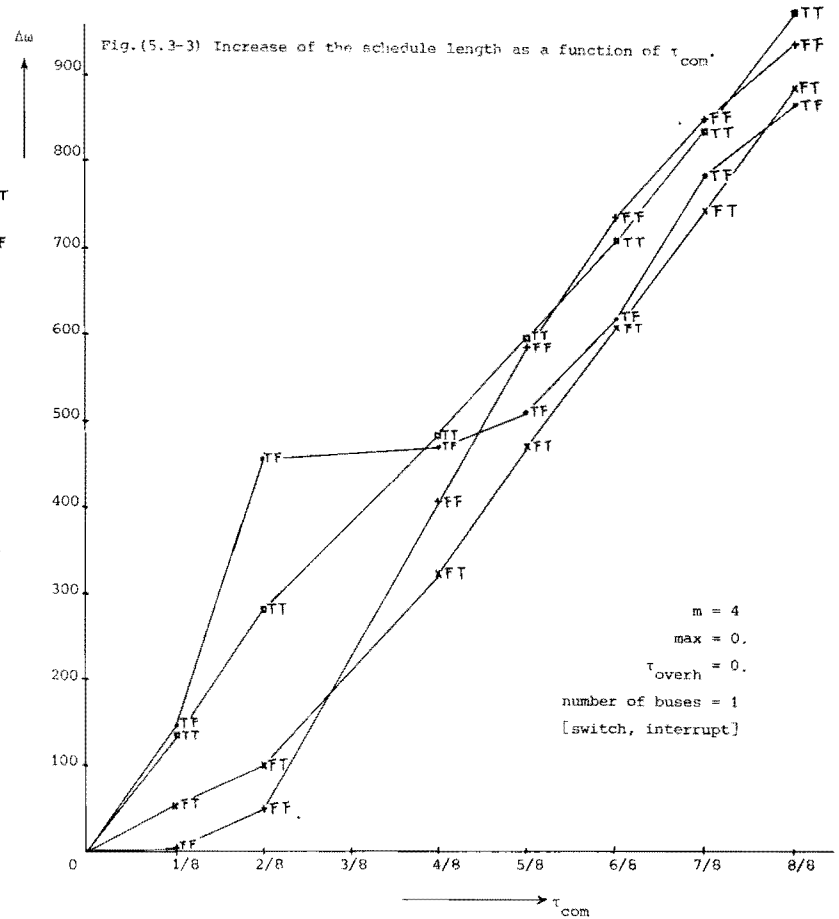


Fig.(5.3-3) Increase of the schedule length as a function of τ_{com}.

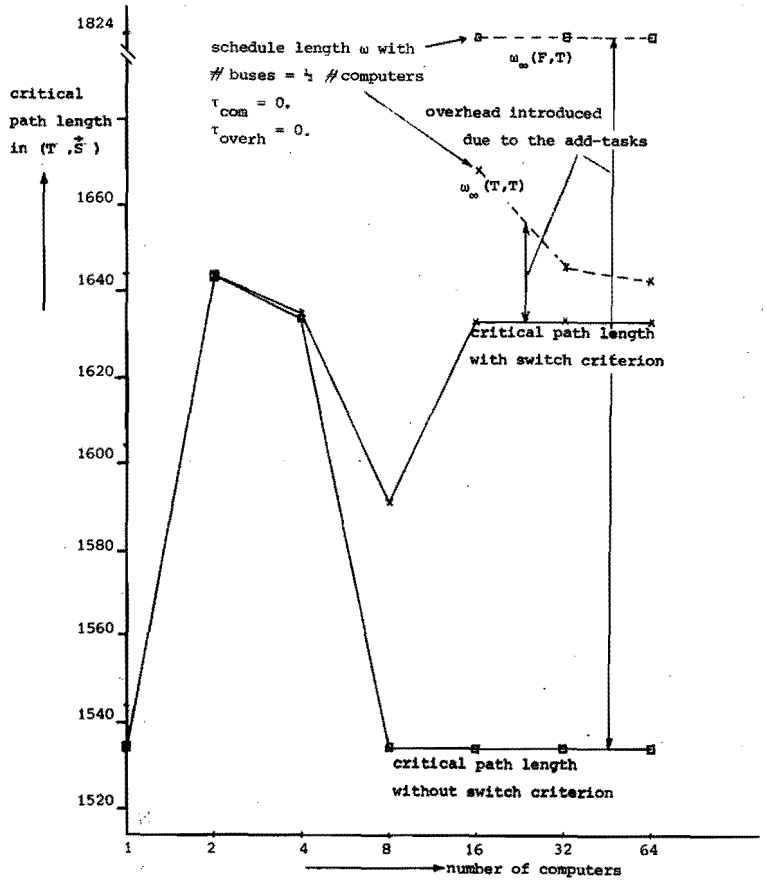


Fig. (5.3-4) Influence of the switch criterion.

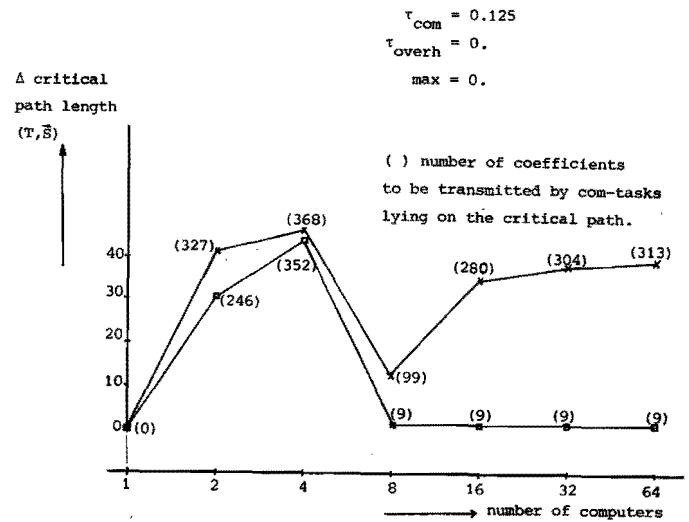


Fig. (5.3-5) Increase of the critical path length in (T,S⃗) due to the com-tasks lying on this path.

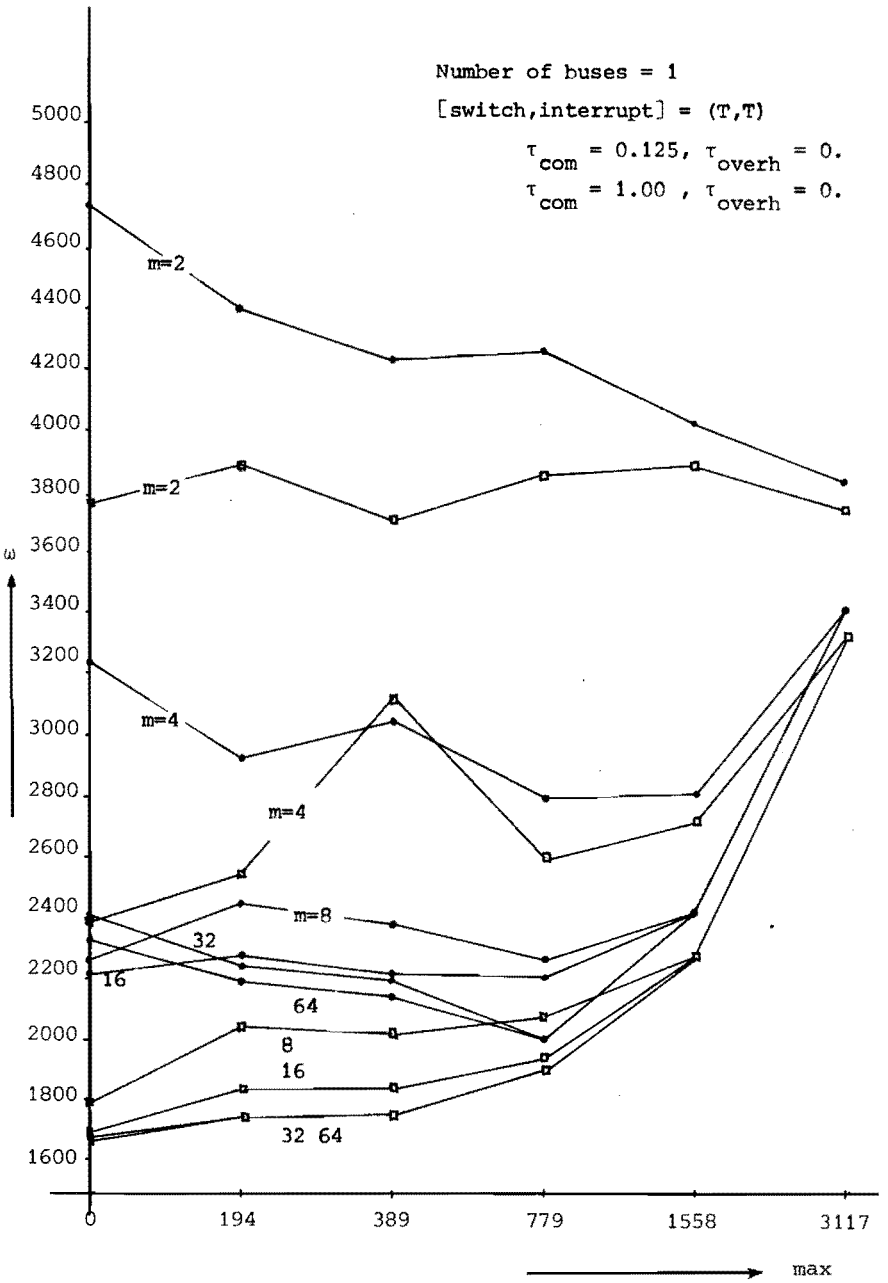


Fig.(5.3-6) The schedule length ω as a function of parameter max.

6. FINAL REMARKS

6.0 Introduction

The first section of this chapter deals with the implementation of the N-R convergence test and the initialization of a new time step. The second section deals briefly with the scheduling of the remaining tasks and finally in the last section the conclusions and concluding remarks are given.

6.1 Implementation of the N-R convergence test and the new time step initialization

The implementation of the tasks with labels 8, 12, 13 and 14 of table (1.3.1) is considered in this section. The task with label 8 tests whether the N-R iteration converged or not. The set of tasks, determined by the labels 12, 13 and 14, determines the new time step and order of the PBD formulas.

The convergence test is accomplished by all computers together. To this purpose computer C_i has to execute the procedure "N-R convergence test(V_i)" where the set V_i denotes the vertices which have been assigned to computer C_i, for $i \in \{1,\ldots,m\}$. The time instant t8 has been determined by the scheduling.

1. procedure N-R convergence test(V_i);
2. begin
   comment This procedure resides in computer C_i, ε is the allowed deviation;
3. if ($\exists_{u \in V_i} [\Delta x(\alpha(u)) > \varepsilon]$) set(or);
4. test(time,t8);
   comment All computers are synchronized to give them the opportunity to set the or signal;
5. nonconvergence $\leftarrow$ or;
6. reset(or);
7. if (nonconvergence) go to N-R iteration;
8. end;

The duration $\tau(8)$ is equal to the maximum time which is required by any of the computers C_i to execute the procedure N-R convergence test(V_i). If the statement in line 3 determines almost completely the required execution time then the task duration $\tau(8)$ is proportional to

1/m (with the assumption that the same amount of vertices has been assigned to all computer modules).

The determination of the new time step and order will also be accomplished by all computers together. The tasks with labels 12, 13 and 14 are combined into a simple task with label 17. The task with label 17 is the successor of all tasks with label 11. To execute the task with label 17 computer C_i has to execute procedure "new time step(I_{xci})", where the set I_{xci} denotes a subset of I_{xc} the set of controlled components, for $i \in \{1,\ldots,m\}$.

```
1. procedure new time step(I_xci);
2. begin
   comment This procedure resides in computer C_i, k is the order of
   the PBD, h is the time step, time step is a label;
3. K(1),K(2),K(3),L(1),L(2),L(3) ← 0; I(1)←1; I(2)←2; I(3)←3;
4. for j=1 step 1 until 3 do K(j)←min({h^m(v) | (m=k-2+j)∧v∈I_xci});
5. for j=1 step 1 until m do;
6.     begin
7.     if (j=i) then broadcast(K,I,bus) else receive(L,I,bus);
8.     for ℓ=1 step 1 until 3 do if (K(ℓ)>L(ℓ)) then K(ℓ) ← L(ℓ);
9.     end;
10. pick some p∈{ℓ|K(ℓ) = max({K(1),K(2),K(3)})};
11. h ← K(p);
12. k ← k+p-2;
13. t ← t+h;
14. if (t<t_e) then go to time step;
15. end;
```

The duration $\tau(17)$ will be equal to the maximum time which is required by any of the computers to execute its procedure new time step. If the statement in line 4 determines almost completely the required execution time then the duration is again proportional to 1/m (with the assumption that $|I_{xci}|$ is equal for all computer modules).

6.2 Scheduling of the total computation phase

In the preceding chapters only the scheduling of the L\U-decomposition job has been treated, here the scheduling of the whole computation phase will be considered.

The determination of the schedule is again split into the determina-

tion of the assignment of the tasks to the computers and the determination of the time intervals during which the tasks must be processed.

The assignment of the tasks to the computers.

A straightforward way is to apply again list scheduling. The list is determined by the levels of the tasks. (Note that the durations $\tau(7)$, $\tau(8)$ and $\tau(17)$ are not neccesarily to be known). However, a large number of communication tasks may arise. By recognizing that the tasks with labels 3 and 5 have the largest time duration it will be acceptable to let these tasks determine the assignment of the remaining tasks.

Thus the following assignment rules are used:

- tasks, which are associated with the same component of $\underline{x}$, are assigned to the same computer. A task with a label 1,2,9,10 or 11, which evaluates variables for component x of vector $\underline{x}$, is called to be associated with component x;
- tasks with the labels 3 and 4 and evaluating functions which are associated with the same element ℓ are assigned to the same computer.

The assignment starts with assigning the tasks with label 5 to the computers. The tasks are considered as independent tasks. The list scheduling strategy is applied. The list is determined by the task levels which are equal to the task durations.

In exactly the same way the tasks with label 3 are assigned to the computers. The partial assignment, obtained by the assignment of the tasks with label 5 and the assignment rule, is made complete.

The tasks with labels 3 and 4 which are assigned to computer C_i require the prediction of a set of components of $\underline{x}$. This set is the already mentioned set I_{xci}, for $i \in \{1,\ldots,m\}$. Note $I_{xci} \cap I_{xcj}$ is not necessarily empty, for $i,j \in \{1,\ldots,m\}$. The tasks with labels 1,2,9,10 and 11, which are associated with I_{xci}, are assigned to C_i. Some redundant tasks are created by this strategy in order to avoid communication tasks.

The tasks with labels 4 and 6 which update the matrix entries $a(k,\ell)$ and $a(k,n+1)$ are associated with vertex $\alpha(k)$ and will be assigned to the same computer as to which vertex $\alpha(k)$ has been assigned.

The determination of the time intervals during which the tasks must be processed.

It is assumed that each computer contains all components of the $\underline{x}$ vector. This is assured if during the bs-job all components are broad-

casted. The only places where communication tasks can be inserted are given by the edges of the task graph of fig.(2.1-1). By the above assignment the only communication which may be necessary is between tasks witk the labels 3 and 4 and between tasks with the labels 5 and 6. In order to reduce the number of communication tasks the communication tasks between the computers C_i and C_j, arising from tasks with the same labels which are assigned to computer C_i, are packed together into one communication task between the computers C_i and C_j, for $i,j \in \{1,\ldots,m\}$.

Again list scheduling for the augmented basic scheduling model is applied in order to determine the time intervals. The start and finishing time of the time intervals are determined with respect to the tasks with the labels 15,16 and 8.

The program "computation phase C_i" gives the procedures and synchronization instructions which accomplish the tasks of the computation phase being assigned to computer C_i. The time instants $t_{k\ell}$ are determined by the scheduling during the setup phase, the index k denotes a task label and index ℓ denotes a computer.

```
1.              program computation phase C_i;
2.              begin
3.              reset(time);
4.              evaluation of the tasks with labels 1 and 2;
5.              evaluation of the tasks with label 3;
6.              test(time,t_3j);
7.1             broadcast results of the tasks with label 3 to
                computer C_j;
8.1             test(time,t_3k);
9.1             receive results of the tasks with label 3 of
                computer C_k;
                  :
10.             evaluation of the tasks with label 4;
11. N-R iteration reset(time);
12.             evaluation of the tasks with label 5;
13.1            test(time,t_5j);
14.1            broadcast results of the tasks with label 5 to
                computer C_j;
```

```
15.1        test(time,t_5k);
16.1        receive results of the tasks with label 5 of
            computer C_k;
              :
17.         evaluation of the tasks with label 6;
18.         solve the set of linear equations;  see chapter 4
19.         N-R convergence test(V_i);
20.         reset(time);
21.         evaluation of the tasks with label 9,10 and 11;
22.         new time step(I_cxi);
23.         set(ready);
24.         end;
```

Of the m possible broadcast and receive tasks in lines 6-9 and 13-16 only one broadcast task to computer C_j and one receive task of computer C_k is given in the above 'listing' of the program.

The proposed computer organization requires a tight upper bound for the task durations. Whether this is possible depends on the applied hardware. If the upper bounds for the task durations are not tight enough then the schedule strategy and synchronization must be adjusted.

The assignment of the tasks to the computers will remain the same but the time intervals during which they will be processed will be determined while the computation phase is executed. This will be called "on-line scheduling". Because the task durations are reasonable well known the following scheduling is applied.

After the assignment of the tasks has been determined the scheduling of chapter 4 is again applied. On basis of the mean task durations for each computer the sequence by which the tasks, assigned to this computer, will be processed is determined instead of the exact time interval. For each bus the communication tasks which use this bus are labeled according to the sequence they use the bus. The procedures which accomplish the tasks are stored according to the determined sequence in the memory of the computers.

To make on-line scheduling possible for each bus h a signal line "sequence-h" is provided. The sequence-h signal value is a non-nega-

tive integer. Three instructions can operate on the signal sequence-h: test(sequence-h,sx), increase(sequence-h) and reset(sequence-h).
Execution of test(sequence-h,sx) results in active waiting until the value of the signal line sequence-h ≥ sx, where sx is a non-negative integer.
Execution of increase(sequence-h) increases the value of sequence-h with one.
Execution of reset(sequence-h) makes the value of sequence-h zero.
The required synchronization is obtained if the test(time,x) instruction which proceeds a broadcast(A,IBA,h) is replaced by the first column of fig(6.2-1) and the test(time,x) instruction which precedes the receive(B,IRB,h) by the instructions of the last column. In fig. (6.2-1) it is assumed that i computers must be synchronized. If interrupt is allowed the computers have to test the sequence lines periodically with the value sx of the next to execute test(sequence-h,sx) instruction.

```
test(sequence-h,sx);
increase(sequence-h);          test(sequence-h,sx+1);
test(sequence-h,sx+i+1);       increase(sequence-h);
increase(sequence-h);          test(sequence-h,sx+i+2);
broadcast(A,IBA,h);            receive(B,IRB,h);
```

Fig.(6.2-1) synchronization.

6.3 Conclusions and concluding remarks

In this thesis a parallel computer organization is outlined to process circuit analysis problems in parallel. The guidance of the parallel processing is done automatically without referring to the user.
The parallel computer system, the asynchronous array computer, consists of a general purpose computer which performs the setup phase and an array of computer modules which performs the computation phase. The normal setup phase is extended by the decomposition and scheduling procedures.
The proposed decomposition and scheduling strategies allow a speedup ratio of about 20 on the asynchronous array computer with 40 computer modules.
This performance is reasoned as follows. In section 2.1 it has been

shown that the success of the parallel processing of the circuit analysis job (computation phase) depends mainly on the speedup which can be obtained by a parallel solution of the set of linear equations. The parallel organization of the solution job (mainly L\U-decomposition) can be regarded as the most critical part of the total job with respect to the scheduling and communication demands. Due to this observation the parameters for the scheduling strategy and the connection network of the asynchronous array computer can be obtained from the simulation results for the L\U-decomposition method as presented in chapter 5. The decomposition of this job into tasks results in a critical path length of 1534 units (potential speedup of 4.06) with the given values of τ_{add}, τ_{mul} and τ_{div}. Due to the chosen organization the schedule length and speedup becomes 1644 units and 3.79 respectivily. This is given by $\omega_\infty=\omega_\infty(m)$ with allowed interrupt and switch criterion shown in fig.(5.3-4). By a one bus network with $\tau_{com}=\tau_{overh}=$ $=0$ the schedule length is only slightly increased as can be seen by comparing figures (5.3-1) and (5.3-4). The simulation results given by figures (5.3-1) and (5.3-2) show that this one bus connection network gives only an increase of the schedule length of about 10% if it allows data exchange at the rate of two coefficients in a time equal to τ_{mul}.

The above mentioned results allow to choose a single bus connection network which makes the parallel computer highly modular. Removing or adding a computer module requires only the change of the parameters in the scheduling procedures.

If the schedule length as a function of m is approximated by formula (2.1.2) with $c = 0.5$ and $m_0 = 7.6$ then fig.(2.1-2) shows the expected speedup for the computation phase which lies somewhere between the curves 3 and 4. Further if at least an effective use of the computer modules is required of 50% then the maximum speedup which can be expected is given by the intersection of the SR=SR(m) curve with the curve SR= m/2 in fig.(2.1-2).

The computer modules are very simple since they have only to perform floating point instructions and a few other instructions. To get an impression of the computing power the computer modules are thought to be built around the components of the Am2900 series [6.1] and a floating point processor of HP [6.2]. The floating point processor consists of three chips: a floating point add/substract chip (1,2μs),

a floating point multiplication chip (1,2μs) and a floating point division chip (2,4μs) which perform the operation on 64 bits operands in the time given within the parenthesis. The Am2900 series contains components which allow a μ-programmable computer organization.
The effective computing power for the 40 modules will be about 20 million 64 bits floating point operations per second (scalar operations).
The expected speedup can be degradated by the introduction of communication tasks and if the assumptions which are made in section 2.1 do not hold. As long as $m << N$, the number of components of $\underline{x}$, the assumptions will hold and the scheduling strategy is capable to keep the communication overhead low.
The obtained speedup is exclusively due to the parallel organization of the computation phase. The speedup for the circuit analysis job will be almost equal to the speedup of the computation phase because the preprocessing takes about the same time to accomplish one pass through the time loop and N-R iteration.
The setup phase is extended with the decomposition and scheduling procedures and loading and unloading of the computer modules. The decomposition procedure has the same complexity as the pivoting procedure $O(Nd_{gem}^2)$ while the scheduling has a $O(n^2)$ complexity, where n is the number of tasks which is less or equal to the number of components N. Hence the preprocessing time for the parallel organization will be of the same order as the original preprocessing time.
Some further remarks are made.

Firstly a few remarks are made about the e-tree construction. A suitable pivoting assured a dominant diagonal. To preserve this some vertices must be inserted into a label class with a higher label as would be possible by the selection criterion.
The matrix after the pivoting is considered as a structural symmetric matrix. If $a(i,j) \neq 0$ and $a(j,i) = 0$ this leads to a superfluous entry in hte data structure however due to the class of circuits considered the number of these empty entries in the data structure is limited.
The pivoting problem and the decomposition problem are treated independent of each other. The objective of reordering to minimize fill-ins usually competes with the objective of reordering to minimize the number of label classes [6.3]. Also the computation power of the applied computer may influence the pivoting strategy. If the computers

have indeed infinite processing power the objective is to minimize the number of classes.

Secondly a few remarks are made about the organization of the solution job on the asynchronous array computer.

The definition of the tasks.

A choice is made between the two solution procedures Gauss and Crout. Consider a full matrix with dimension N. The majority of the required operations are performed during the first pivot steps, given by the compound statement in lines 4-7 of procedure Gauss in section 3.0, if the Gauss procedure is applied. If the Crout procedure is applied then the majority of the required operations is performed during the pivot steps, given by the compound statement in lines 4-14 of procedure Crout in section 3.0, in and around the N/2-th pivot step. In a sparse matrix this is not so clear but by applying the Gauss procedure the number of operations which are associated with the first label classes will be larger than in case of applying the Crout procedure. The dimension of the label classes is independent of the applied procedure. Hence it may be expected that the Gauss procedure results in a shorter critical path in the e-tree if the computers have a limited computing power.

Thirdly the organization of the communication may require more attention. The IO ports can be equipped with an IO processor and a small memory. The computers of the communicating modules can continue with processing while their IO ports are communicating. The simultaneous access of the buffer by the IO processor and processor must be excluded.

The obtainable speedup can be increased if a mixed organization is applied. First an organization as described in chapter 4 can be applied. When those label classes which contain for instance three or four vertices are reached, then the organization can be changed to the rowwise organization as given in chapter 3.

Another way to increase the obtainable speedup is to use computer modules with different processing power. A few computer modules may be equipped with several arithmetic processors. These arithmetic processors can operate according to the first implementation scheme given in section 3.2.

Graph notations

A "graph" $G = (V,E)$ is defined by a set V of elements called "vertices" and a set E of unordered distinct vertex pairs called "edges" thus $E \subseteq \{(u,v) \mid u,v \in V;\ u \neq v\}$.

If $(u,v) \in E$, u and v are "adjacent" vertices and the edge (u,v) is "incident" to u and v.

The set "inc(v,E)" denotes the set of edges incident to vertex v.

Thus $\mathrm{inc}(v,E) := \{(u,v) \in E \mid u \in V\}$.

The set "adj(v,E)" denotes the set of vertices adjacent to vertex v.

Thus $\mathrm{adj}(v,E) := \{u \in V \mid (u,v) \in E\}$.

A set $W \subseteq V$ identifies a "subgraph" $G(W) = (W,E(W))$ of $G = (V,E)$ with $E(W) := \{(u,v) \in E \mid u,v \in W\}$ such that $G(W) \subseteq G$.

The set vertices given by $\{u \in V\backslash W \mid ((u,v) \in E) \wedge (v \in W)\}$, is adjacent to W. It will be denoted by $\mathrm{adj}(W,E)$.

The set edges given by $\{(u,v) \in E \mid (u \in V\backslash W) \wedge (v \in W)\}$, is incident to W. This set will be denoted by $\mathrm{inc}(W,E)$.

A "chain" is a subgraph $(C,E(C)) \subseteq (V,E)$ with $C = \{v_1,v_2,\ldots,v_k\}$, where $v_i \neq v_j$ if $i \neq j$, ordered such that $v_{i+1} \in \mathrm{adj}(v_i,E)$ for $i \in \{1,2,\ldots,k-1\}$. The "length" of the chain is $|C|$.

A "cycle" is a chain such that $v_1 \in \mathrm{adj}(v_k,E)$. Its length is also $|C|$.

A "chord" of a chain $(C,E(C)) \subset (V,E)$ is an edge $(v_i,v_j) \in E\backslash E(C)$ with $v_i,v_j \in C$. A chord connects two nonadjacent vertices of a chain.

A "clique" $C \subseteq V$ with respect to (V,E) is a vertex set with the property that all its members are mutually adjacent, thus $[C \text{ is a clique}] \leftrightarrow [\forall_{u \in C}\ \forall_{v \in C}\ [(u,v) \in E]]$.

The "deficiency" of a vertex u, denoted by $\mathrm{def}(u,E)$, is the set of edges that lacks to make $\mathrm{adj}(u,E)$ a clique. Thus $\mathrm{def}(u,E) := \{(v,w) \mid (v,w \in \mathrm{adj}(u,E)) \wedge ((v,w) \notin E)\}$.

A graph (V,E) is called "connected" if there exists a chain between any two vertices of V.

A "separator" $S \subset V$ ((V,E) being a connected graph) is a set of vertices such that $(V\backslash S,\ E(V\backslash S))$ is not a connected graph, that is to say it consists at least of two distinct nonempty subgraphs called "components". Assume some separator S such that $u,v \in V$ are in different components. Then S is called an "u,v-separator". If S is an u,v-separator such that no subset of S has the same property then S is a "minimal u,v-separator".

A graph (V,E) is called a "tree" if (V,E) is a connected graph with-

out any cycles.

The graph (V,B) is called a "spanning tree" of (V,E) if (V,B) is a tree.

A set $W \subset V$ is called a "cluster" with respect to (V,E) if $(W,E(W))$ is connected.

A "triangulated graph" is a graph such that any cycle of length ≥ 4 has a chord.

A set of edges T, such that $(V,E \cup T)$ is a triangulated graph, is called a "triangulation" of (V,E).

The triangulation is "minimal" if no subset of it is also a triangulation.

The triangulation is "minimum" if T contains the minimum number of edges.

A graph (V,E) is called an "ordered graph" if a bijection $\alpha : \{1,2,\ldots, |V|\} \to V$ is defined. The bijection α is called the "ordering" of (V,E).

Two graphs (V,E) and (V',E') are "isomorphic" if there is a bijection $\lambda : V \to V'$ with $E' = \{(\lambda(u), \lambda(v)) \mid (u,v) \in E\}$.

Given a graph (V,E) and an anti reflexive relation $R(u,v)$ in V, such that $\forall_{(x,y)\in E}[R(x,y) \vee R(y,x)]$ holds. If $(x,y) \in E$ and $R(y,x)$ hold then the edge (x,y) is called "directed" from y to x.

The relation $R(u,v)$ will be called the "precedence relation" of $(u,v) \in E$. To denote the precedence relation (u,v) is noted as an ordered pair such that $R(u,v)$ holds; with this convention the graph will be denoted by $(V,\vec{E})$.

The set $\{v \in adj(u,E) \mid R(u,v)\}$ is called the set of "descendants of u". The descendants of u are denoted by $madj(u,E,R)$ and let $madj(u_1,\ldots,u_n,E,R)$ denote $\bigcap_{i=1}^{n} madj(u_i,E,R)$.

A "path" is a chain $(C,E(C))$ in (V,E) with $C = \{v_1,\ldots,v_k\}$ such that $R(v_i,v_{i+1})$, for $1 \leq i < k$.

A "loop" is a path such that $v_1 \in adj(v_k,E)$ and $R(v_k,v_1)$ holds.

Let $w(u)$ denote the "weight" of $u \in V$. The "path length", based on the weights, of a path $(P,E(P))$ in (V,E) is given by: $\sum_{u \in P} w(u)$.

A path called a "critical path", based on the weights, if no other in (V,E) has a larger path length.

A tree (V,B) is called an "in-tree" if each $v \in V$ has at most one descendant.

If no confusion is possible the vertex set E will be dropped in $adj(u,E)$ and also the relation will be dropped in $madj(u_1,\ldots,u_n,E,R)$.

Notations

$M(V)$	power set of V.
$\underline{x}$	vector with p_x components.
$\underline{x}^T$	transposed vector $\underline{x}$.
$(\underline{x}^T \mid \underline{y}^T)$	transposed vector with the first p_x components given by $\underline{x}$ and the last p_y components given by $\underline{y}$.
#	number.

INDEX

REFERENCES

[0.1] G.H. Barnes, R.M. Brown, M. Kato, D.J. Kuck, D.L. Slotnick and R.A. Stoker, "The Illiac IV computer", IEEE Trans. Computers, vol. C-17, pp. 746-757, Aug. 1968.

[0.2] J. Rumbaugh, "A data flow multiprocessor", IEEE Trans. Computers, vol. C-26, pp. 138-146, Febr. 1977.

[0.3] R.J. Swan, S.H. Fuller and D.P. Siewiorek, "CM* - a modular, multi-microprocessor", Proc. AFIPS 1977 NCC, vol. 46, AFIPS Press, Arlington Va., 1977, pp. 637-644.

[0.4] K.E. Batcher, "STARAN parallel processor system hardware", in 1974 Nat. Comput. Conf., AFIPS Conf. Proc., vol. 43, pp. 405-410.

[1.1] L.O. Chua and P. Lin, "Computer Aided Analysis of Electronic Circuits : Algorithms & Computational Techniques", Englewood Cliffs, New Jersey: Prentice-Hall, inc., 1975.

[1.2] W.M.G. van Bokhoven, "Linear implicit differentiation formulas of variable step and order", IEEE Trans. Circuits Syst., vol. CAS-22, pp. 109-115, Febr. 1975.

[1.3] Chung-Wen Ho, A.E. Ruehli and P.A. Brennan, "The modified nodal approach to network analysis", IEEE Trans. Circuits Syst., vol. CAS-22, pp. 504-509, June 1975.

[1.4] L.W. Nagel, "Spice 2 : A computer program to simulate semiconductor circuits", Memorandum No. ERL-M 520, 9 May 1975, Electronics Research Laboratory, College of Engineering University of California, Berkeley.

[2.1] H.S. Stone, "Problems of parallel computation", in "Complexity of Sequential and Parallel Numerical Algorithms", J.F. Traub, Ed. New York : Academic, 1973, pp. 1-16.

[2.2] H.T. Kung, "New algorithms and lower bounds for the parallel evaluation of certain rational expressions and recurrences", J. ACM, vol. 23, no. 2, pp. 252-261, April 1976.

[2.3] E.G. Coffman, "Computer and job-shop scheduling theory", New York: John Wiley & Sons, 1976.

[2.4] M.J. Gonzalez, "Deterministic processor scheduling", ACM Computing Surveys, vol. 9, no. 3, pp. 173-204, Sept. 1977.

[3.0] Wilkinson, J.H.,"The algebraic eigenvalue problem", Oxford, Oxford University Press, 1965.

[3.1] J.L. Rosenfeld, "A case study in programming for parallel processors", Communications of the ACM, vol.12, pp. 645-655, Dec. 1969.

[3.2] P.A.Gilmore, "Matrix computations on an associative processor", in "Lecture Notes in Computer Science 24 Parallel Processing", G. Goos and J. Hartmanis, Ed. Berlin Heidelberg New York: Springer Verlag, 1975, pp. 272-290.

[3.3] S. Parter, "The use of linear graphs in Gauss elimination", SIAM Rev., vol. 3, pp. 119-130, April 1961.

[3.4] D.J. Rose, "A graph-theoretic study of the numerical solution of sparse positive definite systems of linear equations", in "Graph theory and computing", R. Read, Ed. New York: Academic, 1972, pp. 183-217.

[3.5] D.J. Rose, "Triangulated graphs and the elimination process", J. Math.Anal.Appl., vol. 32, pp. 597-609, 1970.

[3.6] C. Berge, "Some classes of perfect graphs", in "Graph Theory and Theoretical Physics", F. Harary, Ed. New York: Academic, 1967, pp. 155-166.

[3.7] I.S. Duff, "A survey of sparse matrix research", Proc. IEEE, Vol. 65, pp. 500-535, April 1977.

[3.8] R.D. Berry, "An optimal ordering of electronic circuit equations for a sparse matrix solution", IEEE Trans. Circuit Theory,vol.CT-18, pp. 40-50, Jan. 1971.

[3.9] H.M. Markowitz, "The elimination form of the inverse and its application to linear programming", Management Sci., vol. 3, pp. 255-269, 1957.

[3.10] L. Csanky, "Fast parallel matrix inversion algorithms", SIAM J. Comput., vol. 5, pp. 618-623, Dec. 1976.

[3.11] J.W. Huang and O. Wing, "On minimum completion time and optimal scheduling of parallel triangulation of a sparse matrix", IEEE Power Engineering Society Summer Meeting, Los Angeles, July 16-21, 1978. IEEE PES Abstract no. A78-567-0.

[3.12] C. Pottle and Y.M. Wong, "Nonlinear circuit simulation on a parallel microcomputer system", in Proc. 1976 IEEE Int. Symp. Circuits and Syst., April 1976, pp. 394-397.

[3.13] J.A.G. Jess, "Some new results on decomposition and pivoting of large sparse systems of linear equations", IEEE Trans. Circuits Syst., vol. CAS-23, pp. 729-738, Dec. 1976.

[3.14] D.A. Calahan, "Parallel solution of sparse simultaneous linear equations" in "Proc. Eleventh Annual Allerton Conf. on Circuit and Syst. Theory", October 1973, pp. 729-735.

[3.15] G. Kron, "A set of principles to interconnect the solutions of physical systems", J. Applied Phys., vol. 24, pp. 965-980, Aug. 1953.

[3.16] F.H. Branin, "A sparse matrix modification of Kron's method of piecewise analysis", in "Proc. 1975 IEEE Int. Symp. on Circuits and Syst., April 1975, pp. 383-386.

[3.17] L.O. Chua and L.K. Chen, "Diakoptic and generalized hybrid analysis", IEEE Trans. Circuits Syst., vol. CAS-23, pp. 694-705, Dec. 1976.

[3.18] A.K. Kevorkian and J. Snoek, "Decomposition in large scale systems: theory and application in solving large sets of nonlinear simultaneous equations", in "Decomposition of large scale problems", D.M. Himmelblau, Ed. Amsterdam, The Netherlands: North Holland, 1973.

[3.19] A.L. Sangiovanni-Vincentelli, "A graph theoretical interpretation of nonsymmetric permutation on sparse matrices", Int.J. Circuit Theory and Appl., vol. 5, pp. 139-147, 1977.

[3.20] J.T.M. Pieck, "Formele definitie van een e-tree", Memorandum 80-06 Technische Hogeschool Eindhoven, Onderafdeling der Wiskunde, April 1980.

[3.21] R. Tarjan, "Depth-first search and linear graph algorithms", SIAM J. Comput., vol. 1, pp. 146-160, June 1972.

[4.1] R.L. Graham, E.L. Lawler, J.K. Lenstra and A.H.G. Rinnooy Kan, "Optimization and approximation in deterministic sequencing and scheduling: a survey", Annals of Discrete Mathematics, vol. 5, pp. 287-326, 1979.

[4.2] R.L. Graham, "Bounds on multiprocessing timing anomalies", SIAM J. Appl. Math., vol. 17, pp. 416-429, 1969.

[4.3] T.C. Hu, "Parallel sequencing and assembly line problems", Operations Research, vol. 9, pp. 841-848, 1961.

[4.4] M.T. Kaufman, "An almost-optimal algorithm for the assembly line scheduling problem", IEEE Trans. Comp., vol. C-23, pp. 1169-1174, Nov. 1974.

[5.1] Philips Data handbook, Signetics Integrated circuits Part 8, May 1981, pp. 388-391.

[5.2] "AEP-118 Bus Test System", American Electric Power Service Corporation, New York.

[5.3] W.L. Powell, private communication, Bonneville Power Administration, Portland, Oregon.

[6.1] Advanced Micro Devices, The Am2900 Family Data Book With Related Support Circuits, 901 Thompson Place, P.O.Box 453, Sunnyvale, California 94086.

[6.2] F. Ware and W. McAllister, "C-MOS chip set streamlines floating-point processing", Electronics pp. 149-152, Febr. 10, 1982.

[6.3] O. Wing and J. Huang, "A parallel triangulation process of sparse matrices", in Proc. 1977 Int.Conf. Parallel Processing, Aug. 1977, pp. 207-214.

STELLINGEN

bij het proefschrift van

H.G.M. Kees

25 mei 1982

Technische Hogeschool Eindhoven

1. Bij de complexiteitsanalyse van parallel algoritmen dient met terdege rekening te houden met de struktuur van de toegepaste machine.

2. Naarmate de pakkingsdichtheid van de ic's toeneemt vergt het data transport in computers meer aandacht.

3. Bij parallel processing dienen twee doelstellingen onderscheiden te worden, nl. optimalisatie naar snelheid of naar kosten.

4. Alleen bij onafhankelijke taken is bij toepassing van parallel verwerking de verwerkingssnelheid van de processoren uitwisselbaar met het aantal processoren. (Uitwisselbaar in die zin dat het produkt van verwerkingssnelheid en aantal konstant blijft).

5. Naarmate de graad van specialisatie toeneemt zal ook de bereikbare versnelling en kostenperformance voor parallel processing toenemen.

6. De economische wetenschap is er nog niet in geslaagd (reken)modellen te ontwikkelen, welke in een groot gebied geldig zijn.
   E.J. Bomhoff, "De noodzaak van het bezuinigen", Intermediair 10, 12 maart 1982.

7. Het aantal werklozen wordt teveel als graadmeter voor de economie gebruikt.

8. We moeten er op bedacht zijn niet onze "eigen" neergang te financieren door de aankoop van geavanceerde wapens in het "buitenland".