Monograph:
Zomaya, A.Y. and Morris, A.S. (1989) Fast Forward Dynamics Algorithm for Robot Arms Using Multi-Processing. Research Report. Acse Report 367 . Dept of Automatic Control and System Engineering. University of Sheffield

---

Fast Forward Dynamics Algorithm for Robot Arms Using Multi-Processing

by

A. Y. ZOMAYA

A. S. MORRIS

Department of Control Engineering
University of Sheffield
Mappin Street
Sheffield S1 3JD
U.K.

Research Report No. 367

June 1989

Abstract

The computation of the direct dynamics problem (forward dynamics) plays a major role in the real-time computer modelling and simulation of robot manipulators. The efficient and computationally inexpensive solution of this problem facilitates the design of real-time robot simulators. In addition, it allows for a better understanding of the key elements affecting robot operations.

This work proposes to solve this problem by employing parallel and distributed processing techniques. First, a parallel implementation of a simplified Lagrange-Euler formulation is used to solve for the dynamics. Second, a resulting system of linear equations is solved using Gaussian-Elimination with simple row interchange. Both algorithms are distributed over a multiple-instruction multiple-data stream (**MIMD**) computer architecture. The system is constructed from (**VLSI**) building blocks called the (**TRANSPUTER**). Quantification of the speed and utilisation measures are given to demonstrate the cost-effectivness of the parallel approach. The problem is solved for a 6 dof robot arm.

Key words: robot dynamics, forward dynamics, parallel processing, Transputer, Occam, MIMD.

1. Introduction

A precise mathematical model of the robot arm is an indispensable part of any dynamic computer simulation strategy (Fig.1). The accurate and meticulous formulation of the dynamic model will lead to a better understanding of the different components of the system [15, 34, 44].

The model can be used to simulate real-time motions of the robot arm. In addition, it provides a powerful tool for the study of different control techniques and for evaluating system performance under conditions that might be dangerous and expensive if examined *in situ* [38].

However, the dynamic equations that influence robot performance are complicated, highly coupled, and non-linear. General simplifying assumptions have to be made to reduce the computational burden. This is accomplished by ignoring part of the forces which affect the dynamics [3]. The results obtained are valid for a limited range of operations which will lead to an overall sub-optimal performance.

The mathematical formulation of the dynamics divides into two distinct areas; the *Direct and Inverse Dynamic* formulations. Most of the research has been focused on the latter problem. In this work the former problem is addressed which is equally

important and computationally more expensive.

The formulation of computationally efficient dynamics has been an active area of research for the last two decades and several methods have been developed. The Lagrange-Euler (LE) [3, 37, 45] has high computational complexity but is a very well structured and systematic representation. The Recursive Lagrangian [16] gives good computational results but destroys the structure of the equations. The Newton-Euler (NE) [1, 33, 36, 47] has the most efficient computational formulation but has untidy recursive equations. Other approaches include the tabulation techniques [41], Kane's dynamic equations [23], and the Generalized D'Alembert [29]. The most commonly used of these methods are the (LE) and (NE). The interaction and equivalence of these schemes has been shown by Silver [42]. In this paper the (LE) is addressed. The results and discussions are presented in the following order; Section (2) surveys recent research in this area. Section (3) presents the computer architecture used to implement this work. The dynamic model is explained and analyzed in section (4). Parallelism is introduced to solve the problem in section (5). Conclusions and further comments are given in section (6).

2. Previous Work

A typical robot arm consists of an open-chain of (N+1) rigid links. The links can be arranged such that link (i) is connected to a preceeding link (i−1) and a following link (i+1). The links are connected by joints and each joint has one degree of freedom (dof). In robot arms, two types of joints exist, translational and revolute.

As mentioned earlier, the direct dynamics problem is computationally expensive. Nevertheless, only a few attempts have been made to solve it. Walker and Orin [47] proposed four methods by using the (NE) approach. Swartz [43] described a method to depict the arm by its rotational characteristics. A set of equations based on the (NE) called the *Inverse Arm* and an algorithm called the *Forward Arm* were presented. Featherstone [12] proposed a different technique based on the so-called articulated-body method.

In contrast, several algorithms have been developed which apply parallel processing techniques to solve the *Inverse Dynamics* problem [2, 6, 7, 24, 28, 30, 40, 46, 49, 52]. Lately, an attempt has been made by Lee and Chang [31] to incorporate parallelism to speed up the computation. Their approach was based on that of Walker and Orin [47] and the algorithms were distributed on a single-instruction multiple-data stream (*SIMD*) computer.

The use of *Distributed Computing* techniques by employing several computing units, interacting in real-time to provide the required computational speed and power, finds an immediate application in robotics. There is a vast plethora of literature on distributed computing, and many, if not all, of the concepts described are directly applicable to robotics. In this paper a trial is made to introduce these concepts to robot dynamic simulation to emphasize the role which can be played by distributed architectures in enhancing real-time operations by distributing the whole of a task over several cooperating processors.

3. Multiple-Instruction Multiple-Data Stream Architecture (MIMD)

We are currently witnessing an enormous revolution in the mass manufacuring of low cost advanced VLSI components. This will enable the actual implementation of the theoretical parallel-orientated architectures and algorithms [5].

Parallelism is achieved by distributing the job over a number of processors, ideally in such a way that all the processors are fully utilised. As a consequence, highly parallel structures have evolved, and many have been built to meet the increasing demand for more computing power and higher processing speed [14, 17, 26, 48].

Until recently, most of the research has been dealing with solving problems from a parallel perspective by applying (*SIMD*) techniques. In this approach, a single machine instruction is able to compute over massive data structures (e.g. vector and array processors). However, the use of (*SIMD*) architectures was hindered by their technological constraints. As a result, the multiprocessing (*MIMD*) emerged to provide a solution. In this case a number of processors interact and co-operate to produce higher performance and computing power. The (*MIMD*) architecture is *tightly coupled* if the processors are highly interactive. Otherwise, it is considered to be *loosely coupled.*

3.1. Transputer and Occam

The T800 **TRANSPUTER**† (Fig.2) which is adopted in this work is a 32 bit microcomputer with 4 Kbytes of on chip RAM for high processing speed, a configurable memory interface, 4 bidirectional communication links, 64-bit floating point unit, and a timer. It achieves an instruction rate of 10 MIPS (millions of instructions per second) by running at a speed of 20 MHz. The Transputer is one of the first

† TRANSPUTER and OCCAM are trademarks of the INMOS group of companies.

designs that incorporate several hardware features to support parallel processing. This allows for any number of Transputers to be arranged together to build a parallel processing system, and permits massive concurrency without further complexity. To provide maximum speed with minimal wiring, the Transputer uses point to point serial communication links for direct connection to other Transputers.

OCCAM† is a high level language developed to run on the Transputer [19, 20, 25] and optimise its operation. It is simple, block structured, and supports both sequential (**SEQ**) and parallel (**PAR**) features on one or more Transputers which can be used to facilitate simulation, modelling and control of complicated physical systems [13, 21, 22].

3.2. Processor Farms

The processor farms are based on a simple concept which might be useful in a wide range of applications [18]. It involves a *master* processor which acts as a controller that optimises processor utilisation and farms out tasks to a set of *slave* processors in the network. When a task is completed successfully by a slave processor, the results are sent back to the *master* which then farms out another task to it.

In this work, a similar processor organization is used. This technique is motivated by the amount of tasks that can be executed independently in the proposed algorithm. The software portions running on the processors are replica of one another with minor modifications depending upon the task and the communication protocols.

4. Manipulator Dynamic Model

The importance of the (LE) evolves from its simple, algorithmic and highly structured formulation. In general, the (LE) equations of motion can be written in a compact state-space formulation:

$$\boldsymbol{\tau}(t) = \mathbf{D}(\boldsymbol{\Theta})\,\ddot{\boldsymbol{\Theta}}\,(t) + \mathbf{C}(\boldsymbol{\Theta},\dot{\boldsymbol{\Theta}}) + \mathbf{h}(\boldsymbol{\Theta}) \tag{1a}$$

or alternatively,

$$\tau_i(t) = \sum_{j=1}^{n} d_{ij}\,\ddot{\Theta}_j + \sum_{j=1}^{n}\sum_{k=1}^{n} \dot{\Theta}_j\, c_{jk}(i)\,\dot{\Theta}_k + h_i \qquad i = 1,\ldots,n \tag{1b}$$

where $\boldsymbol{\tau}(t)$ is an n × 1 applied force/torque vector for joint actuators; $\Theta(t)$, $\dot{\Theta}(t)$, and $\ddot{\Theta}(t)$ are n × 1 vectors representing position, velocity and acceleration respectively; $\mathbf{D}(\Theta)$ is an n × n effective and coupling inertia matrix; $\mathbf{C}(\Theta,\dot{\Theta})$ is an n × 1 Coriolis and Centripetal effects vector; and $\mathbf{h}(\Theta)$ is an n × 1 gravitational force vector, where (n) is the no. of degrees of freedom (dof).

The very general form of eq.(1) is important in state space and modern control applications; however, it can't be utilised unless simplified [3, 4, 8, 11, 32, 35, 50].

In this work a semi-customised symbolic form is used. The formulation was first introduced by Bejczy [3] and later by Paul [37] , then refined and further simplified by Zomaya and Morris [50]. The conventions used are the same as those of Bejczy [3] and Paul [37] , being based on the Denavit-Hartenberg (DH) [9] representation.

The dynamic formulation is divided into two main parts :

(I) The vectors $\boldsymbol{\delta}_l$, and $\mathbf{d}_l$ which describe the differential rotation, and differential transformation of link (l) respectively. These vectors are customised and symbolically expressed for a certain type of manipulator. The general description of $\boldsymbol{\delta}_i^l$, and $\mathbf{d}_i^l$ is given in Paul [37];

$$\mathbf{d}_i^l = \begin{cases} (-n_{lx}^{i-1}\, p_{ly}^{i-1} + n_{lx}^{i-1}\, p_{lx}^{i-1})\; i \\ (-o_{lx}^{i-1}\, p_{ly}^{i-1} + o_{lx}^{i-1}\, p_{lx}^{i-1})\; j \\ (-a_{lx}^{i-1}\, p_{ly}^{i-1} + a_{lx}^{i-1}\, p_{lx}^{i-1})\; k & \textit{revolute joint} \\ (n_{lz}^{i-1}\; i + o_{lz}^{i-1}\; j + a_{lz}^{i-1}\; k) & \textit{prismatic joint} \end{cases} \tag{2}$$

$$\boldsymbol{\delta}_i^l = \begin{cases} (n_{lz}^{i-1}\; i + o_{lz}^{i-1}\; j + a_{lz}^{i-1}\; k) & \textit{revolute joint} \\ 0 & \textit{prismatic joint} \end{cases} \tag{3}$$

(II) This part describes a general formulation of the inertial, coriolis, centripetal and gravitational effects.

(a) The *Effective and Coupling Inertias*:

$$\mathbf{D}_{ij} = \sum_{l=max(i,j)}^{n} tr\,(\,\Delta_j^l\, \mathbf{J}^l\, \Delta_i^{lT}\,) = \sum_{l=max(i,j)}^{n} \sum_{m=1}^{3} e_{mm} \tag{4}$$

where $\sum_{m=1}^{3} e_{mm}$ is given as,

$$= J_{11}^l \begin{bmatrix}\delta_{iy}\\ \delta_{iz}\end{bmatrix}_l \begin{bmatrix}\delta_{jy}\\ \delta_{jz}\end{bmatrix}_l + J_{22}^l \begin{bmatrix}\delta_{ix}\\ \delta_{iz}\end{bmatrix}_l \begin{bmatrix}\delta_{jx}\\ \delta_{jz}\end{bmatrix}_l + J_{33}^l \begin{bmatrix}\delta_{ix}\\ \delta_{iy}\end{bmatrix}_l \begin{bmatrix}\delta_{jx}\\ \delta_{jy}\end{bmatrix}_l + J_{44}^l \begin{bmatrix}d_{ix}\\ d_{iy}\\ d_{iz}\end{bmatrix}_l \begin{bmatrix}d_{jx}\\ d_{jy}\\ d_{jz}\end{bmatrix}_l$$

$$+ J_{24}^l \left(\begin{bmatrix}\delta_{jx}\\ \delta_{iz}\end{bmatrix} \begin{bmatrix}d_{iz}\\ -d_{jx}\end{bmatrix} + \begin{bmatrix}\delta_{ix}\\ \delta_{jz}\end{bmatrix} \begin{bmatrix}d_{jz}\\ -\delta_{jz}\end{bmatrix} \right)_l + J_{34}^l \left(\begin{bmatrix}\delta_{iy}\\ \delta_{jx}\end{bmatrix} \begin{bmatrix}d_{jx}\\ -d_{iy}\end{bmatrix} + \begin{bmatrix}\delta_{jy}\\ \delta_{ix}\end{bmatrix} \begin{bmatrix}d_{ix}\\ -d_{jy}\end{bmatrix} \right)_l \tag{5}$$

(b) The *Coriolis and Centripetal Forces*:

$$\mathbf{C}_{ijk} = \sum_{l=max(i,j,k)}^{n} tr\,(\,\Delta_j^l\, \Delta_k^l\, \mathbf{J}^l\, \Delta_i^{lT}\,) = \sum_{l=max(i,j,k)}^{n} \sum_{m=1}^{3} u_{mm} \tag{6}$$

where $\sum_{m=1}^{3} u_{mm}$ is given as,

$$= J_{11}^{l}\ \delta_{jx}^{l}\ \left[\begin{bmatrix}\delta_{ky}\\ \delta_{iy}\end{bmatrix}\begin{bmatrix}\delta_{iz}\\ -\delta_{kz}\end{bmatrix}\right]_l + J_{22}^{l}\ \delta_{ix}^{l}\delta_{jy}^{l}\ \left[\begin{bmatrix}\delta_{kz}\\ 1\end{bmatrix}\begin{bmatrix}1\\ -\delta_{kx}\end{bmatrix}\right]_l + J_{33}^{l}\ \delta_{jz}^{l}\ \left[\begin{bmatrix}\delta_{kx}\\ \delta_{ix}\end{bmatrix}\begin{bmatrix}\delta_{iy}\\ -\delta_{ky}\end{bmatrix}\right]_l$$

$$+ J_{44}^{l}\left(d_{ix}\begin{bmatrix}\delta_{jy}\\ \delta_{jz}\end{bmatrix}\begin{bmatrix}d_{kz}\\ -d_{ky}\end{bmatrix} + d_{iy}\begin{bmatrix}\delta_{jz}\\ \delta_{jx}\end{bmatrix}\begin{bmatrix}d_{kx}\\ -d_{kz}\end{bmatrix} + d_{iz}\begin{bmatrix}\delta_{jx}\\ \delta_{jy}\end{bmatrix}\begin{bmatrix}d_{ky}\\ d_{kx}\end{bmatrix}\right)_l$$

$$+ J_{24}^{l}\left(\delta_{ix}\begin{bmatrix}\delta_{jz}\\ \delta_{jy}\end{bmatrix}\begin{bmatrix}d_{ky}\\ d_{kz}\end{bmatrix}\begin{bmatrix}\delta_{jx}\\ \delta_{jy}\end{bmatrix}\begin{bmatrix}d_{ky}\\ -d_{kx}\end{bmatrix} + \delta_{jy}\begin{bmatrix}\delta_{kx}\\ \delta_{kz}\end{bmatrix}\begin{bmatrix}d_{ix}\\ d_{iz}\end{bmatrix} - d_{iy}\begin{bmatrix}\delta_{jx}\\ \delta_{jz}\end{bmatrix}\begin{bmatrix}\delta_{kx}\\ \delta_{kz}\end{bmatrix}\right)_l$$

$$+ J_{34}^{l}\left(\delta_{iy}\begin{bmatrix}\delta_{jy}\\ \delta_{jz}\end{bmatrix}\begin{bmatrix}d_{kz}\\ -d_{ky}\end{bmatrix} + \delta_{jz}\begin{bmatrix}\delta_{kx}\\ \delta_{ky}\end{bmatrix}\begin{bmatrix}d_{ix}\\ d_{iy}\end{bmatrix} + \delta_{ix}\begin{bmatrix}\delta_{jx}\\ \delta_{jz}\end{bmatrix}\begin{bmatrix}d_{kz}\\ -d_{kx}\end{bmatrix} - d_{iz}\begin{bmatrix}\delta_{jy}\\ \delta_{jx}\end{bmatrix}\begin{bmatrix}\delta_{ky}\\ \delta_{kx}\end{bmatrix}\right)_l \quad (7)$$

where J_{ij}^{l}'s are the inertial parameters.

(c) The *Gravitational Effects* are given by:

$$\mathbf{h}_i = g \sum_{l=i}^{n} m_l\ \Psi_l\ \mathbf{r}_l^l \quad (8)$$

where m_l and $\mathbf{r}_l^l$ are the mass and the centre of mass of link (l) respectively, and Ψ_l is a vector of the following form,

$$\Psi_l = \begin{bmatrix} s\alpha\ \delta_{iz} - c\alpha\ \delta_{iy} \\ c\alpha\ \delta_{ix} \\ s\alpha - \delta_{ix} \\ s\alpha\ d_{iy} + c\alpha\ d_{iz} \end{bmatrix}_l \quad (9)$$

where $s\alpha$ and $c\alpha$ are $\sin(\alpha)$ and $\cos(\alpha)$ respectively.

The previous dynamic equations eq.(5,7,8) will be assumed throughout this work.

The dynamic simulation problem of a robot manipulator can be tackled by solving it as a forward dynamics problem. By rearranging eq.(1):

$$\mathbf{D}(\Theta)\ \ddot{\Theta}\ (t) = \tau(t) - \mathbf{M} \quad (10a)$$

$$\mathbf{M} = C(\Theta,\dot{\Theta})\ \dot{\Theta}(t) + h(\Theta) \quad (10b)$$

or alternatively,

$$\sum_{j=1}^{n} d_{ij}\ \ddot{\Theta}_j = \tau(t) - \mathbf{M} \quad (11a)$$

$$\mathbf{M} = \sum_{j=1}^{n}\sum_{k=1}^{n} \dot{\Theta}_j\ c_{jk}(i)\ \dot{\Theta}_k + h_i \quad (11b)$$

Therefore, it can be stated as follows:

(i) Given an input force/torque vector $\tau(t)$, the joint position $\Theta(t)$ and the joint velocity $\dot{\Theta}(t)$, calculate the $\mathbf{D}(\Theta)$ matrix and the bias or offset vector $\mathbf{M}$.

(ii) Then, the joint acceleration vector $\ddot{\Theta}(t)$ is calculated by solving a system of linear equations given in eq.(10a,11a).

5. Real-Time Simulation of Robot Motion

The simulation procedure consists of two main parts. First, solving eq.(1) by dividing the dynamics into a set of subtasks (processes) which in this case consist of the different terms of eq.(1). Second, calculating $\ddot{\Theta}(t)$ eq.(1a) by solving the system of linear equations. Parallelism is introduced to both parts to enhance the speed and efficiency of the algorithms. This is achieved by using different network configurations and task allocations. Quantification of speed and processor utilisation with real-time implementation results are included.

5.1. Parallel Computation of the Dynamics

The Stanford arm is used as an example, and in this case there are three main tasks (i.e. calculating $\mathbf{D}$, $\mathbf{C}$, and $\mathbf{h}$):

Task1: divided into 17 subtasks representing the effective and coupling inertia terms.

Task2: divided into 43 subtasks representing the coriolis and centripetal effects.

Task3: divided into 4 subtasks representing the gravitational effects.

This sums up to a total of 64 subtasks to be computed. Two different scheduling strategies are used. The basic structure of the network configurations is the same, that is, a main processor (Schedular or Controller) and a cluster of slave processors responsible for the computation and number crunching. Several factors have to be considered in the analysis and task distribution phase . First, the sequential dependency between the different subtasks. Second, minimising the interaction between the different slave processors as much as possible by enabling each processor to execute its job without the need for data from other processors. Lastly, avoiding the case of two slave processors communicating with each other through a third slave processor. To avoid (I/O) bottleneck, the overall (I/O) of a processor must be reduced by increasing the size of its memory [27]. Furthermore, redundant calculations must be avoided to minimize the communication overhead.

5.1.1. One-Processor (Sequential) Case

Prior to the actual real-time implementation, a thorough off-line analysis is performed. All the sources of overhead and communication deadlocks are located and avoided. The whole task is computed using one processor (Transputer). The total processing-time for computing the forces vector was found to be (25.6 msec).

5.1.2. Three-Processors Case:

For this case a tree-structured network is used (Fig.3) where (P_0) is the master processor and the other three processors (P_1, P_2, P_3) are slave processors. The master processor is connected to a personal computer (PC) which works as a link between the user and the network of processors. The total processing-time was found to be (8.96 msec). Note that the value of the total processing time (T_p) includes both the computation time of the task and the time needed to send and receive any data items from the processor, i.e.

$$T_{p\ (total\ processing\ time)} = t_{1\ (computation\ time)} + t_{2\ (communication\ time)}$$

The subtasks are distributed as shown in (table 1)

PROCESSOR		
P_1	P_2	P_3
C_{25}^3	$\mathbf{C}_1$	**D**
$\mathbf{C}_2$	C_{44}^3	$\mathbf{C}_5$
$\mathbf{C}_4$	C_{45}^3	C_{14}^3
h	C_{55}^3	C_{24}^3
-	-	C_{15}^3

Table 1. Three–Processors Task Allocation

5.1.3. Six-Processors Case:

The same procedure is followed here with a different network configuration (Fig.4). This architecture gives more independence to each processor and increases the computing power to achieve better processing time.

In this configuration, the first level of the network is a simple tree structure, but each slave processor in (level 1) is a master for another slave processor in (level 2). Hence, (level 1) slave processors communicate directly with the controller (P_0) but slave processors in (level 2) "talk" to (P_0) through their master processor.

The total processing time is (4.7 msec). The scheduling strategies are shown in (table 2). While increasing the number of processors in a network, care must be taken that the additional processors do not lower the processing time and lead to under-utilised resources.

PROCESSOR					
P_1	P_2	P_3	P_4	P_5	P_6
C_{12}^1	D_{14}	C_{14}^2	C_{14}^1	D_{11}	C_4
C_{13}^1	D_{15}	C_{15}^2	C_{15}^1	D_{12}	C_{23}^2
C_{22}^1	D_{16}	C_{16}^2	C_{24}^1	D_{13}	C_{14}^3
C_{23}^1	D_{23}	C_{24}^2	C_{25}^1	D_{22}	C_{44}^3
C_{55}^3	D_{24}	C_{25}^2	C_{26}^1	C_{13}^2	-
C_5	D_{25}	C_{44}^2	C_{44}^1	C_{24}^3	-
-	D_{26}	C_{45}^2	C_{45}^1	h	-
-	D_{33}	C_{46}^2	C_{46}^1	-	-
-	D_{35}	C_{55}^2	C_{55}^1	-	-
-	D_{44}	C_{56}^2	C_{56}^1	-	-
-	D_{46}	C_{15}^3	C_{45}^3	-	-
-	D_{55}	-	-	-	-
-	D_{66}	-	-	-	-
-	C_{25}^3	-	-	-	-

Table 2. Six–Processors Task Allocation

5.1.4. Nine-Processors Case:

A nine-processors network enhances the performance and gives higher processing power while maintaining the desirable cost-effectiveness and fault-tolerance (Fig.5). A total processing-time of (2.95 msec) is obtained. The task scheduling is shown in (table 3).

PROCESSOR								
P_1	P_2	P_3	P_4	P_5	P_6	P_7	P_8	P_9
D_{11}	D_{16}	C_{16}^{4}	C_{12}^{1}	C_{22}^{1}	C_{26}^{1}	C_{13}^{2}	C_{24}^{2}	C_{14}^{3}
D_{12}	D_{22}	C_{25}^{4}	C_{13}^{1}	C_{23}^{1}	C_{44}^{1}	C_{14}^{2}	C_{25}^{2}	C_{15}^{3}
D_{13}	D_{23}	C_{26}^{4}	C_{14}^{1}	C_{24}^{1}	C_{45}^{1}	C_{15}^{2}	C_{44}^{2}	C_{25}^{3}
D_{14}	D_{24}	C_{45}^{4}	C_{15}^{1}	C_{25}^{1}	C_{46}^{1}	C_{16}^{2}	C_{45}^{2}	C_{44}^{3}
D_{15}	D_{25}	C_{55}^{4}	h_2	h_5	C_{55}^{1}	C_{23}^{2}	C_{46}^{2}	C_{45}^{3}
h_4	D_{26}	C_{56}^{4}	C_{16}^{4}	-	C_{56}^{1}	-	C_{55}^{2}	C_{55}^{3}
-	D_{33}	C_{24}^{3}	-	-	C_{16}^{5}	-	C_{56}^{2}	D_{55}
-	D_{35}	D_{46}	-	-	C_{26}^{5}	-	-	-
-	D_{44}	-	-	-	C_{46}^{5}	-	-	-
-	D_{66}	-	-	-	h_3	-	-	-

Table 3. **Nine–Processors Task Allocation**

5.1.5. Comparison of Performance:

The different values of (T_p) are reported in (table 4) and (Fig.6).

PROCESSING TIME (T_p) (msec)	
No. of Processors	*Described Scheme*
1	25.6
3	8.96
6	4.46
9	2.95

Table 4. **Total Processing Time** ($\mathbf{T_p}$)

The *Utilisation* is given by the ratio of the total processing time of each processor to the total processing time of the network, i.e.

$$U = T_p \text{ (one processor)} \; / \; T_p \text{ (network)}$$

The (U) rate shows the percentage contribution of each processor in the execution of the whole job (table 5).

PROCESSOR UTILISATION (%)									
No. of Processors	P_1	P_2	P_3	P_4	P_5	P_6	P_7	P_8	P_9
1	100	-	-	-	-	-	-	-	-
3	100	100	96.8	-	-	-	-	-	-
6	100	98.7	98.7	96.2	97.5	95	-	-	-
9	100	100	100	95.7	95.7	95.7	95.7	95.7	100

Table 5. Processor Utilisation for
Different Network Configurations

5.2. Parallel Solution of System of Equations

In this section the Gaussian Elimination (GE) algorithm [39] is used to solve the linear system of equations representing the dynamics developed in the last section (5.1). The (GE) algorithm is distributed on the network shown in (Fig.7). The configuration used evolves from the basic structure of the algorithm of (GE) with simple row interchange [51]

LEVEL 1:

The processor (T) prepares the matrix (**D**) by augmenting it with the ($\ddot{\Theta}$). A check is performed to avoid a zero pivoting element, and a row interchange is performed as necessary to avoid this situation. Then, normalisation is performed by dividing the whole row by the first entry in that row (i.e. d_{ii}). The remaining rows are sent to the array of processing elements in (level 2). If the number of rows exceeds the number of processing elements in (level 2), (T) will schedule the operation by sending only (M) rows to the (M) processors and the rest will be stored in the local memory (LM) from where they can be restored and sent to any free processor.

LEVEL 2:

This array of processors is operating in parallel. Each processor is loaded with a row of the system matrix from (T) in (level 1). The role of these processors is to make the first element in each loaded row equal to zero. This is accomplished by employing the following formula:

$$J_k = J_k - M_{ki} * J_i$$

where $M_{ki} = J_{ki} / J_{ii}$, J_k is the processed row, J_i is the first row which is used in common with the array of processors. The processed rows are sent to (level 3) and the array is ready now to receive some more rows (if there are any).

LEVEL 3:

This processor (T) will receive the processed rows from (level 2) and store them in its (LM) and checks if all the rows are received. Then all the rows will be recovered and a new matrix constructed and sent back to processor (T) in (level 1) to repeat the whole procedure again. Also (T) will check whether the operation is completed successfully. If not, the fault is located and corrected as fast as quickly as possible.

LEVEL 4:

Back substitution is performed on the resulting matrix and then the values of the joint accelerations ($\ddot{\Theta}$) are sent to the output unit. Thereafter, ($\ddot{\Theta}$) can be integrated using numerical integration techniques (e.g. Runge Kutta methods) to compute (Θ,$\dot{\Theta}$).

The whole procedure will be repeated again if a new (**D**) is received. The size of the (**D**) matrix for the case of the Stanford arm is (6×6). The number of processors used for (level 2) of the network is varied to find an optimum solution. The total processing time is given in (table 6).

PROCESSING TIME (T_p) (msec)	
No. of Processors(m)	*(GE) Algorithm*
1	1.22
2	0.72
3	0.55
4	0.46
5	0.36

Table 6. Total Processing Time (T_p)

Sufficiently-fast real-time results are obtained (table 6) (Fig.8). In order to minimise the cost and complexity of the solution, the number of processors should be limited. In this work ($m = 3$) seems to be a reasonable trade off between speed and efficiency

[10].

6. CONCLUSION AND SUMMARY

In this paper the problem of the real-time dynamic simulation of a robot manipulator has been addressed. The procedure was divided into two main parts. First, solving for the dynamics which was based on a simplified form of the Lagrangian formulation. Second, solving the system of linear equations resulting from the first stage by employing a parallel implementation of the Gaussian Elimination with row interchange.

The whole of the algorithm was distributed over a parallel processing system consisting of the INMOS TRANSPUTER. The programs were written in OCCAM. Real-time results have been produced to demonstrate how the recent advances in VLSI technology can be used together with parallel processing techniques to significantly speed up the dynamic simulation and modelling of robot manipulators.

References

[1]. ARMSTRONG, W. M., (1979). "Recursive Solution to the Equations of Motion of an N-Link Manipulator," in *Proc. 5th World Congress on Theory of Machines and Mechanisms*, vol. 2, pp. 1343-1346.

[2]. BARHEN, J., (1987). "Hypercube Ensembles: An Architecure for Intelligent Robots," in *Computer Architectures for Robotics and Automation*, ed. J. H. Graham, pp. 195-236, Gordon and Breach Science Pub, NewYork.

[3]. BEJCZY, A. K., (1974). "Robot Arm Dynamics and Control," *NASA-JPL Technical Memorandum, 33-669.*

[4]. BEJCZY, A. K. AND PAUL, R. P., (1981). "Simplifed Robot Arm Dynamics For Control," in *Proc. 20the IEEE Conf. Decision and Control, San Diego*, pp. 261-262.

[5]. BERTSEKAS, B. P. AND TSITSIKLIS, J. N., (1989). *Parallel and Distributed Computation: Numerical Methods,* Prentice-Hall, Englewood cliffs, N.J.

[6]. BINDER, E. E. AND HERZOG, J. H., (1986). "Distributed Computer Architecture and Fast Parallel Algorithms in Real-Time Robot Contol," *IEEE Trans. on Systems, Man, and Cybernetics*, vol. 16, no. 4, pp. 543-549.

[7]. CHEN, C. L., LEE, C. S. G., AND HOU, E. S. H., (1988). "Efficient Scheduling Algorithms for Robot Inverse Dynamics Computation on a Multiprocessor System," *IEEE Trans Systems, Man, and Cybernetics*, vol. 18, no. 5, pp. 729-743.

[8]. CHENG, P., WENG, C., AND CHEN, C., (1988). "Symbolic Derivation of Dynamic Equations of Motion for Robot Manipulators Using Piogram Symbolic Method," *IEEE J. Robotics and Automation*, vol. 4, no. 6, pp. 599-609.

[9]. DENAVIT, H. AND HARTENBERG, R., (1955). "A Kinematic Notation for Lower Pair Mechansims Based on Matrices," *J. Applied Mechanics*, no. 22, pp. 215-221.

[10]. EAGER, D. L., ZAHORJAN, J., AND LAZOWSKA, E. D., (1989). "Speed up Versus Efficiency in Parallel Systems," *IEEE Trans. on Computers*, vol. 38, no. 3, pp. pp. 408-423.

[11]. FAESSLER, H., (1986). "Computer-Assisted Generation of Dynamical Equations for Multibody Systems," *Int. J. Robotics Research*, vol. 5, no. 3, pp. 129-141.

[12]. FEATHERSTONE, R., (1987). *Robot Dynamics Algorithms,* Kluwer Academic Publishers, Cambridge, Mass.

[13]. HAMBLEN, J. O., (1987). "Parallel Continuous System Simulation Using the Transputer," *Simulation*, vol. 49, no. 6, pp. 249-253.

[14]. HAYNES, L. S., LAU, R. L., SIEWIOREK, D. P., AND MIZELL, D. W., (1982). "A Survey of Highly Parallel Computing," *IEEE Computer*, pp. 9-24.

[15]. HEMAMI, H., JASWA, V. C., AND MCGHEE, R. B., (1975). "Some Alternative Formulations of Manipulator Dynamics for Computer Simulation Studies," in *Proc. 13th Allerton Conf. Circuit and System Theory, University of Illinois*, pp. 124-140.

[16]. HOLLERBACH, J. M., (1980). "A Reccursive Lagrangian Formulation of Manipulator Dynamics and a Comparative Study of Dynamics Formulation Complexity," *IEEE Trans. on Systems, Man, and Cyberntics*, vol. smc-10, no. 11, pp. 730-736.

[17]. HWANG, K. AND BRIGGS, F. A., (1985). *Computer Architecture and Parallel Processing,* McGraw-Hill, New York.

[18]. IEE,, (1986). *Colloquium on the Transputer Applications and Case Studies,* Professional Group C2.

[19]. INMOS,, (1984). *OCCAM Programming Manual,* Prentice-Hall, Englewood Cliffs, N.J.

[20]. INMOS,, (1988). *OCCAM-2 Reference Manual,* Prentice-Hall, Englewood Cliffs, N.J.

[21]. JONES, D. I., (1985). "OCCAM Structures in Control Applications," *Trans. Inst. of Measurements and Control*, vol. 7, no. 5, pp. 222-227.

[22]. JONES, D. I. AND ENTWISTLE, P. M., (1988). "Parallel Computation of An Algorithm in Robotic Control," in *Int. Conf. on Control 88, Oxford, U.K*, pp. 438-443.

[23]. KANE, T. AND LEVINSON, D., (1983). "The Use of Kane's Dynamical Equations in Robotics," *Int J. Robotics Res.*, vol. 2, no. 3, pp. 3-21.

[24]. KASAHARA, H. AND NARITA, S., (1985). "Parallel Processing of Robot-Arm Control Computation on a Multi-microprocessor System," *IEEE J. Robotics and Automation*, vol. 1, no. 2, pp. 104-113.

[25]. KERRIDGE, J., (1987). *OCCAM Programming: A Practical Approach,* Blackwell.

[26]. KUNG, H. T., (1982). "Why Systolic Architectures," *IEEE Computer*, pp. 37-46.

[27]. KUNG, H. T., (1985). "Memory Requirements for Balanced Computer Architectures," *J. of Complexity*, vol. 1, no. 1, pp. 147-157.

[28]. LATHROP, R. H., (1985). "Parallelism in Manipulator Dynamics," *Int. J. Robotics Res.*, vol. 4, no. 2, pp. 80-102.

[29]. LEE, C. S. G., LEE, B. H., AND NIGAM, R., (1983). "Development of the Generalized D'Alembert Equations of Motion for Mechanical Manipulators," in *Proc. 22nd Conf. Decision and Control, San Antonio, Tex.*, pp. 1205-1210.

[30]. LEE, C. S. G. AND CHANG, P. R., (1986). "Efficient Parallel Algorithm for Robot Inverse Dynamics Computation," *IEEE Trans. on Systems, Man, and Cybernetics*, vol. 16, no. 4, pp. 532-542.

[31]. LEE, C. S. G. AND CHANG, P. R., (1988). "Efficient Parallel Algorithm for Robot Forward Dynamics Computation," *IEEE Trans. on Systems, Man, and Cybernetics*, vol. 18, no. 2, pp. 238-251.

[32]. LEWIS, R. A., (1974). "Autonomous Manipulation on a Robot: Summary of Manipulator Software Functions," *Tech. Memo. 33-679, Jet Propulsion Laboratory, Pasadena, California.*

[33]. LUH, J. Y. S., WALKER, M. W., AND PAUL, R. P., (1980). "On-Line Computational Scheme for Mechanical Manipulators," *Trans. ASME J. Dynamic Systems,*

Measurements, and Control, vol. 102, pp. 69-76.

[34]. NEUMAN, C. P. AND TOURASSIS, V. D., (1983). "Robot Control: Issues and Insight," in *Proc. 3rd Yale Workshop on Applications of Adaptive Systems Theory*, pp. 179-189.

[35]. NEUMAN, C. P. AND MURRAY, J. J., (1985). "Computational Robot Dynamics: Foundations and Applications," *J. Robotic Systems*, vol. 2, no. 4, pp. 425-452.

[36]. ORIN, D. E., MCGHEE, R. B., VUKOBRATOVIC, M., AND HARTOCH, G., (1979). "Kinematic and Kinetic Analysis of Open-Chain Linkages Utilizing Newton-Euler Methods," *Math. Biosci.*, vol. 43, pp. 107-130.

[37]. PAUL, R. P., (1981). *Robot Manipulators: Mathematics, Programming, and Control,* MIT Press, Cambridge, Mass.

[38]. PFEIFER, M. S. AND NEUMAN, C. P., (Nov. 1984). "An Adaptable Simulator for Robot Arm Dynamics," *Computers in Mechanical Engineering*, pp. 57-64.

[39]. PRESS, W. H., FLANNERY, B. P., TEUKOLSKY, S. A., AND VETTERLING, W. T., (1986). *Numerical Recipes: The Art of Scientific Computing,* Cambridge University Press.

[40]. RAHMAN, M. AND MEYER, D., (1987). "A Cost-Efficient High Performance Bit-Serial Architecture for Robot Inverse Dynamics Computation," *IEEE Trans. on Systems, Man, and Cybernetics*, vol. 17, no. 6, pp. 1050-1058.

[41]. RAIBERT, M. H. AND HORN, B. K., (1978). "Manipulator Control Using the Configuration Space Method," *Indusrial Robot*, vol. 5, no. 2, pp. 69-73.

[42]. SILVER, W. M., (1982). "On the Equivalence of Langrangian and Newton-Euler Dynamics for Manipulators," *Int. J. Robotics Res.*, vol. 1, no. 2, pp. 60-70.

[43]. SWARTZ, N. M., (1984). "Arm Dynamics Simulation," *Journal of Robotic Systems*, vol. 1, no. 1, pp. 83-100.

[44]. TOURASSIS, V. D. AND NEUMAN, C. P., (1985). "Properties and Structure of Dynamic Robot Models for Control Engineering Applications," *Mechansim and Machine Theory*, vol. 20, no. 1, pp. 27-40.

[45]. UIKER, J. J., (1965). "On the Dynamic Analysis of Spatial Linkages Using 4x4 Matrices," *Ph.D. Thesis, Dept. of Mechanical Engineering and Astronautical Sciences, Northwestern University.*

[46]. VUKOBRATOVIC, M., KIRCANSKI, N., AND LI, S. G., (1988). "An Approach to Parallel Processing of Dynamic Robot Models," *Int. J. Robotics Res.*, vol. 7, no. 2, pp. 64-71.

[47]. WALKER, M. W. AND ORIN, D. E., (1982). "Efficient Dynamic Computer Simulation of Robotic Mechanisms," *Trans. ASME J. Dynamic Systems, Measurements, and Control*, vol. 104, pp. 205-211.

[48]. ZAKHAROV, V., (1984). "Parallelism and Array Processing," *IEEE Trans. Computers*, vol. 33, no. 1, pp. 45-78.

[49]. ZHENG, Y. AND HEMAMI, H., (1986). "Computation of Multibody System Dynamics by a Multiprocessor Scheme," *IEEE Trans. on Systems, Man, and Cybernetics*, vol. 16, no. 1, pp. 102-110.

[50]. ZOMAYA, A. Y. AND MORRIS, A. S., (1988). "The Dynamic Performance of Robot Manipulators Under Different Operating Conditions," *Research Report No. 345, Dept. of Control Engineering, University of Sheffield, Sheffield S1 3JD, U.K.*

[51]. ZOMAYA, A. Y. AND MORRIS, A. S., (1988). "Distributed VLSI Architectures for Fast Jacobian and Inverse Jacobian Formulations," *Research Report No. 346, Dept. of Control Engineering, University of Sheffield, Sheffield S1 3JD, U.K.*

[52]. ZOMAYA, A. Y. AND MORRIS, A. S., (1989). "Robot Inverse Dynamics Computation Via VLSI Distributed Architectures," *Research Report No. 350, Dept. of Control Engineering, University of Sheffield, Sheffield S1 3JD, U.K.*

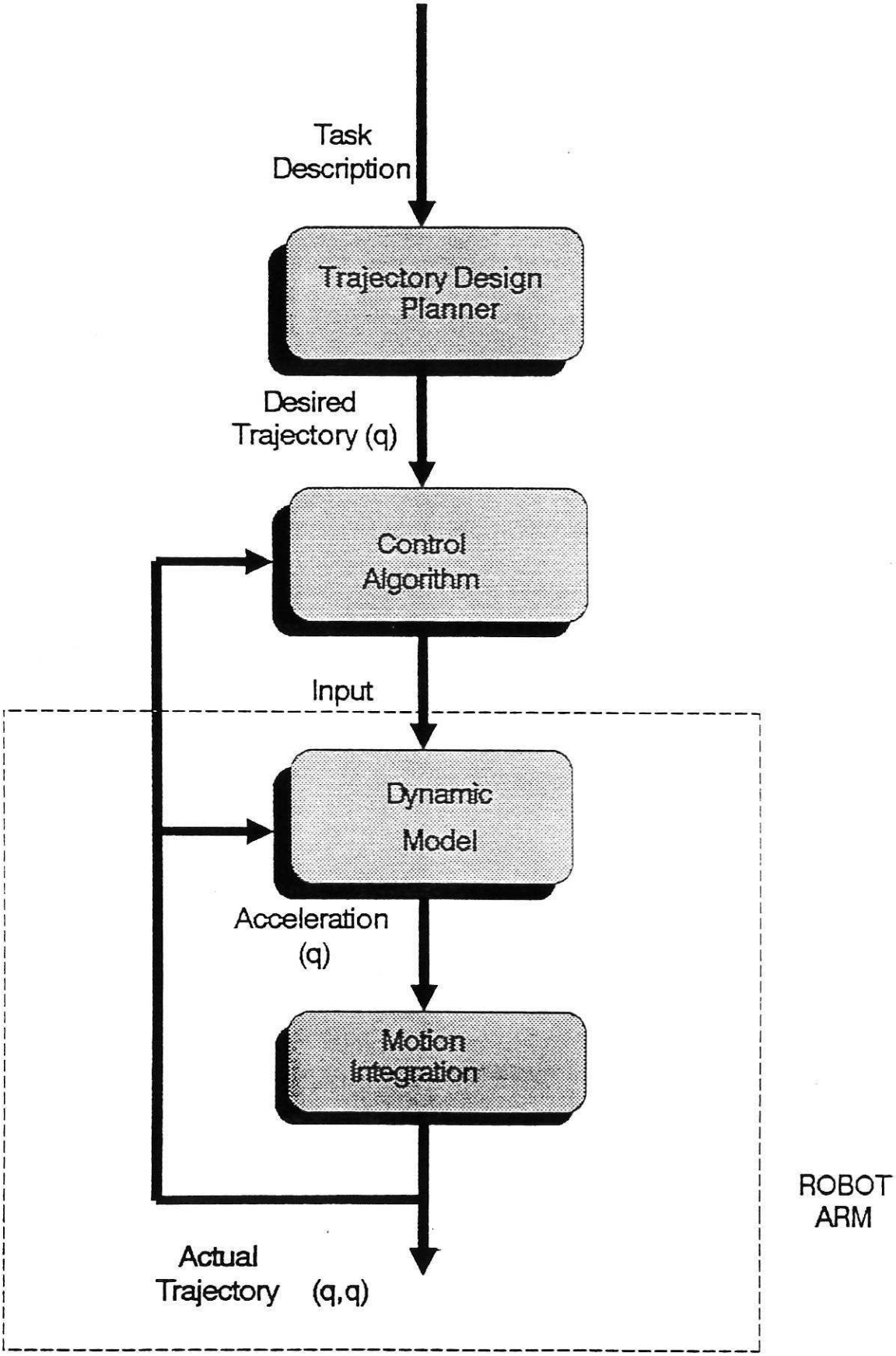

Figure 1. A Dynamic-Based Simulation Strategy.

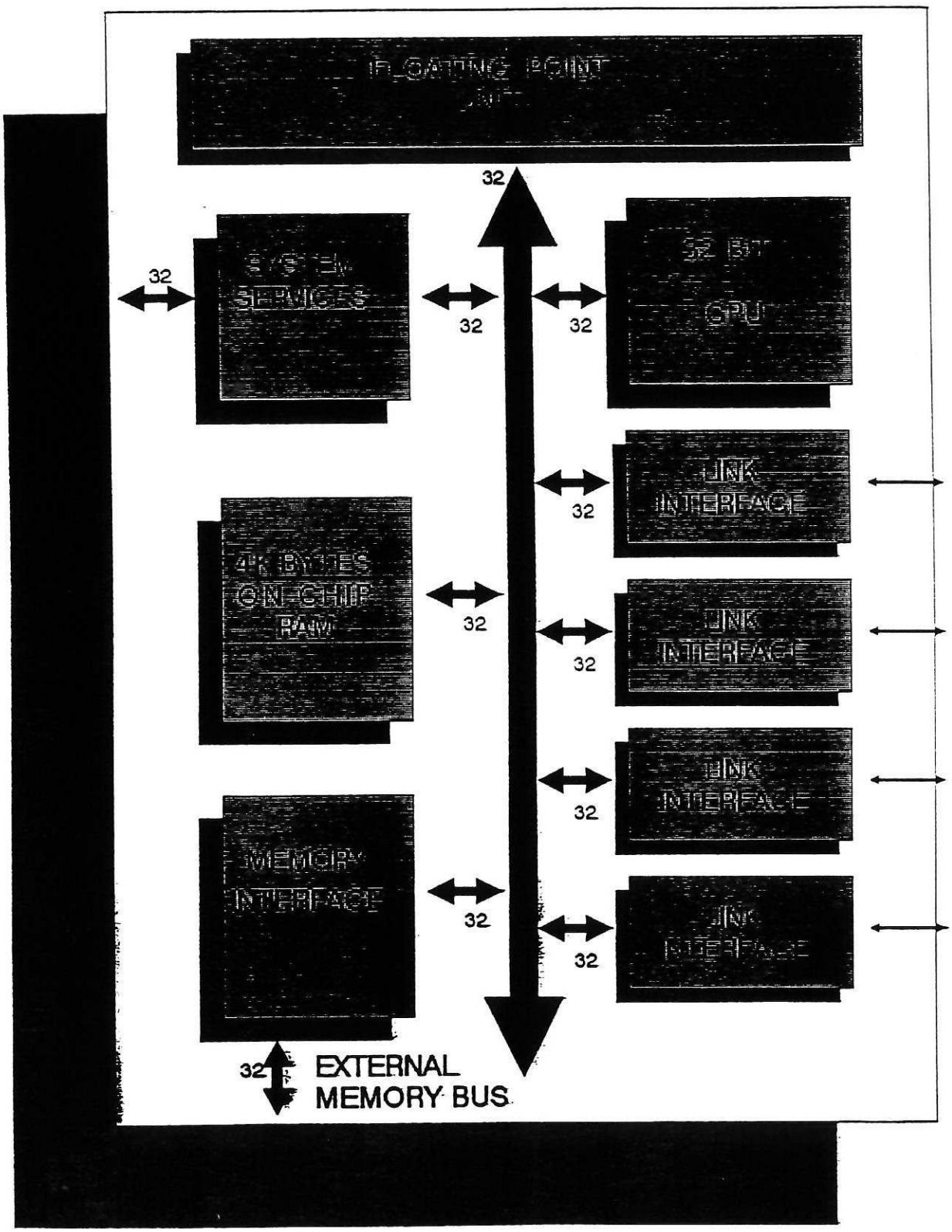

Figure 2. The well known INMOS T800 Transputer.

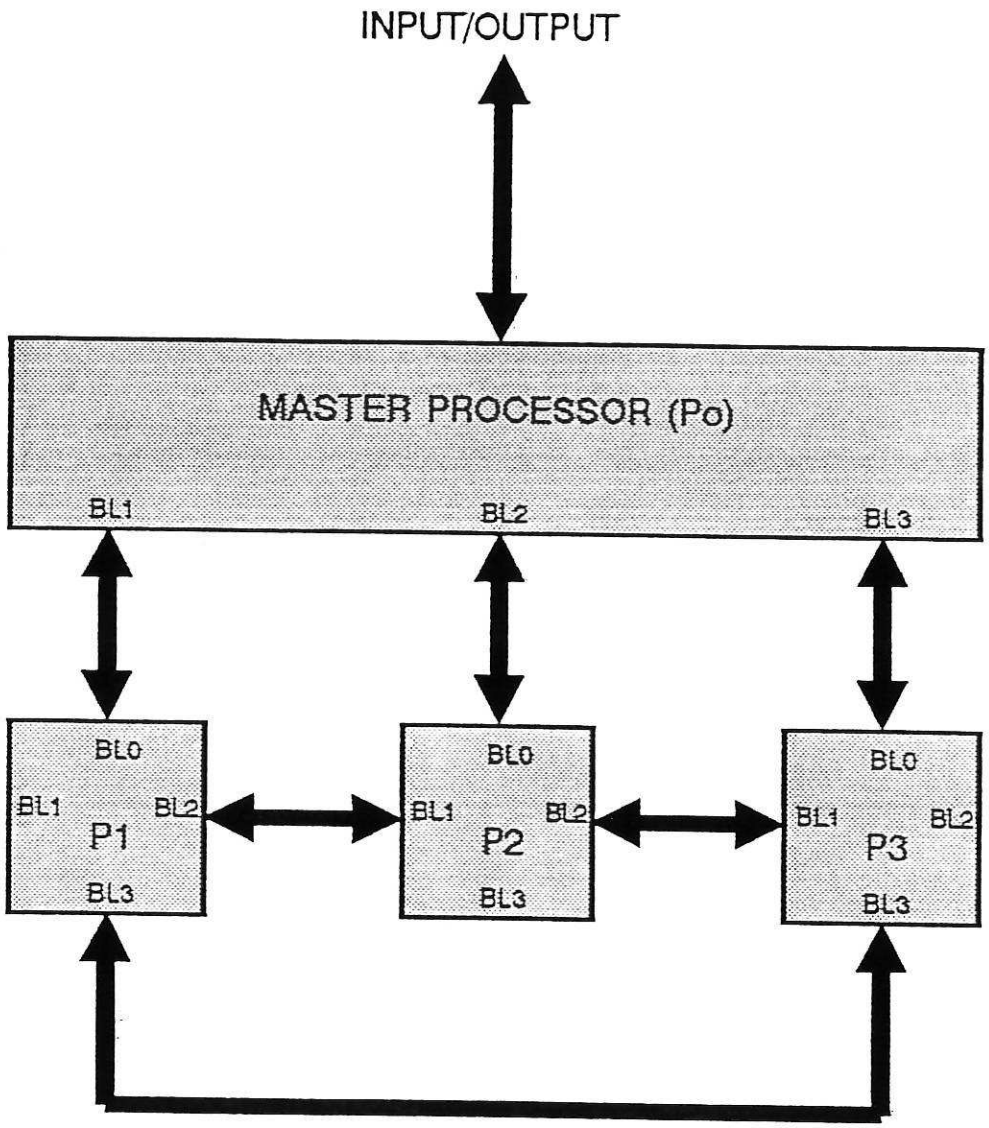

Figure 3. Three-Processors Network.

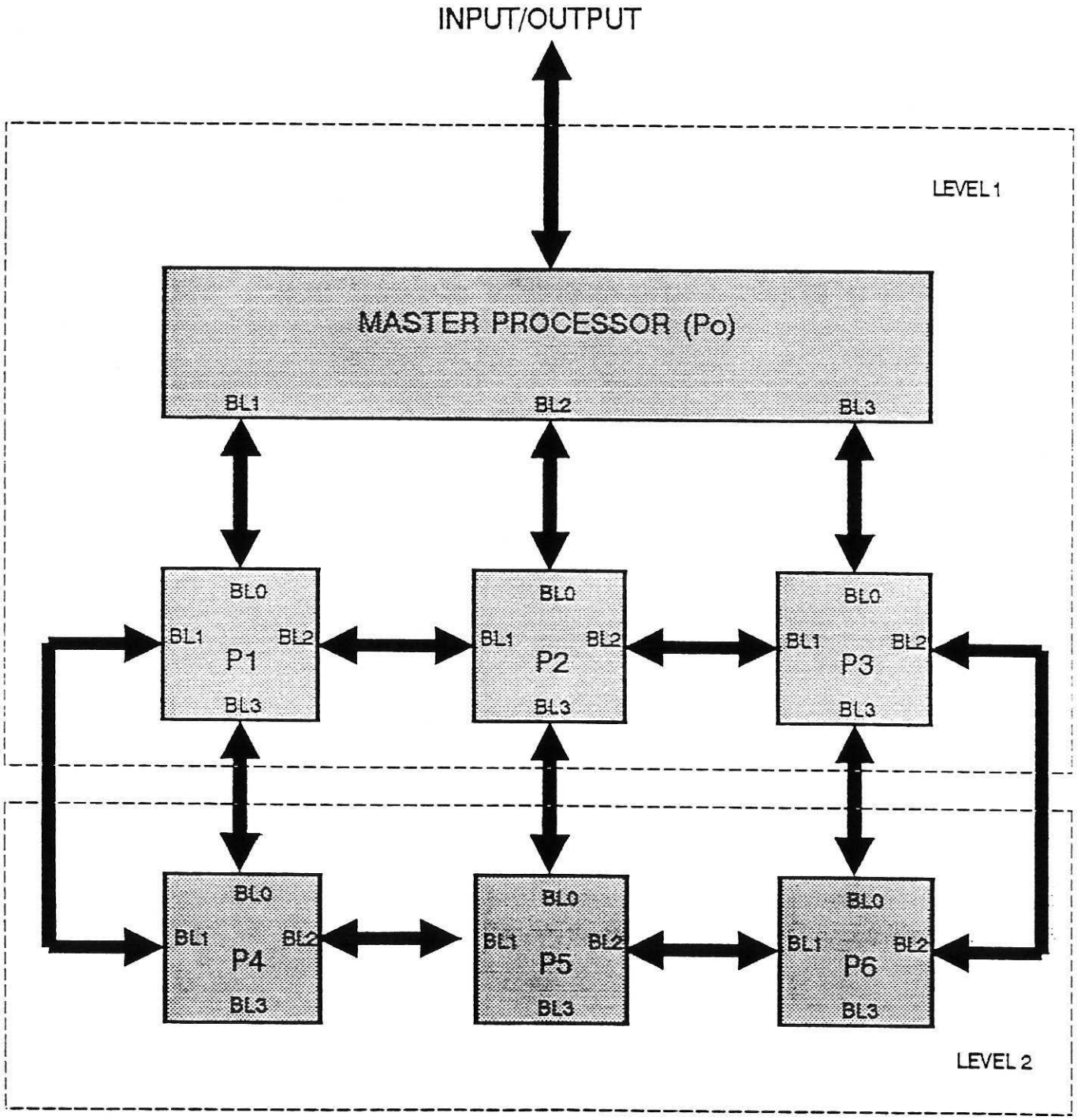

Figure 4. Six-Processors Network.

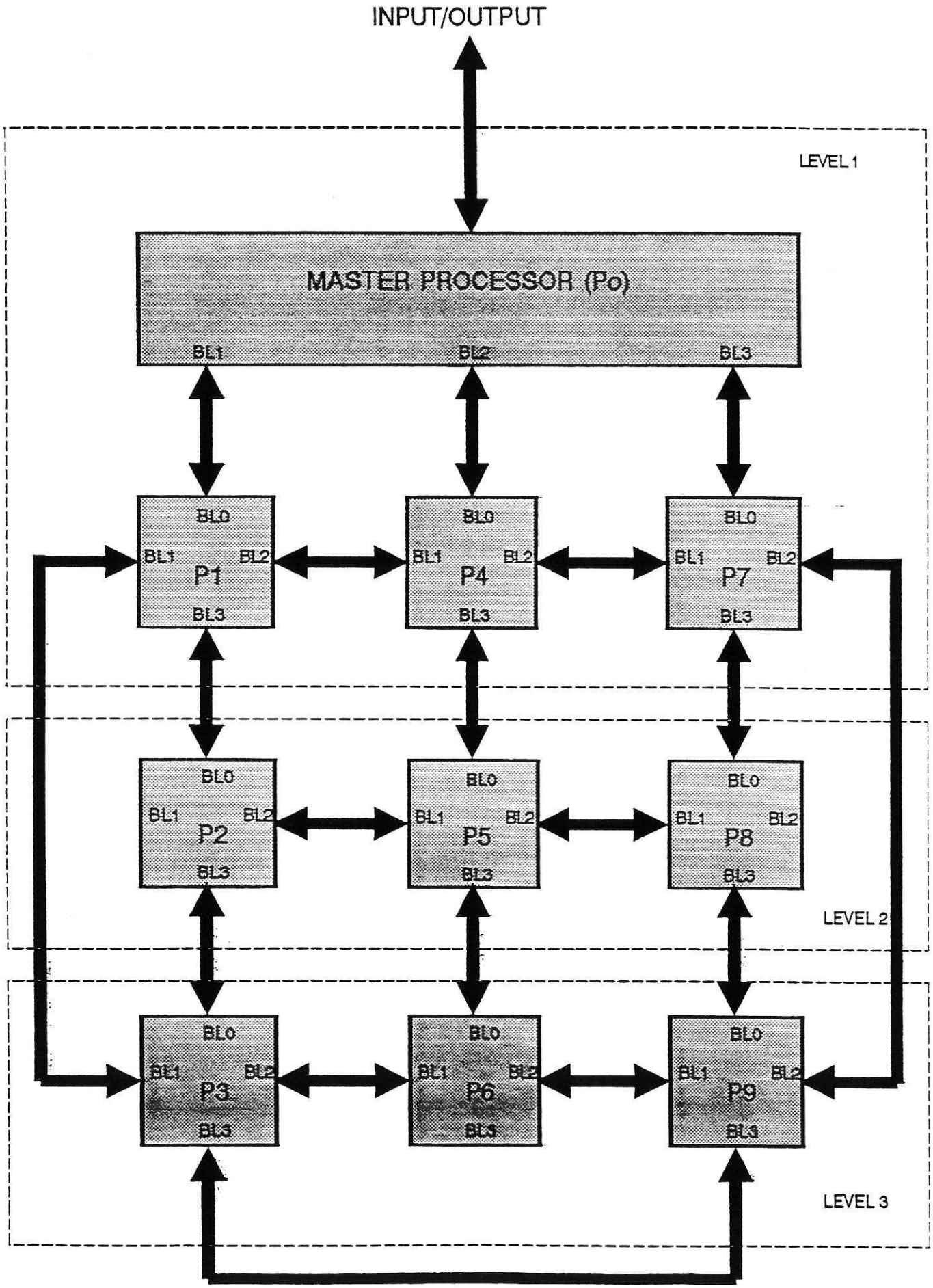

Figure 5. Nine-Processors Network.

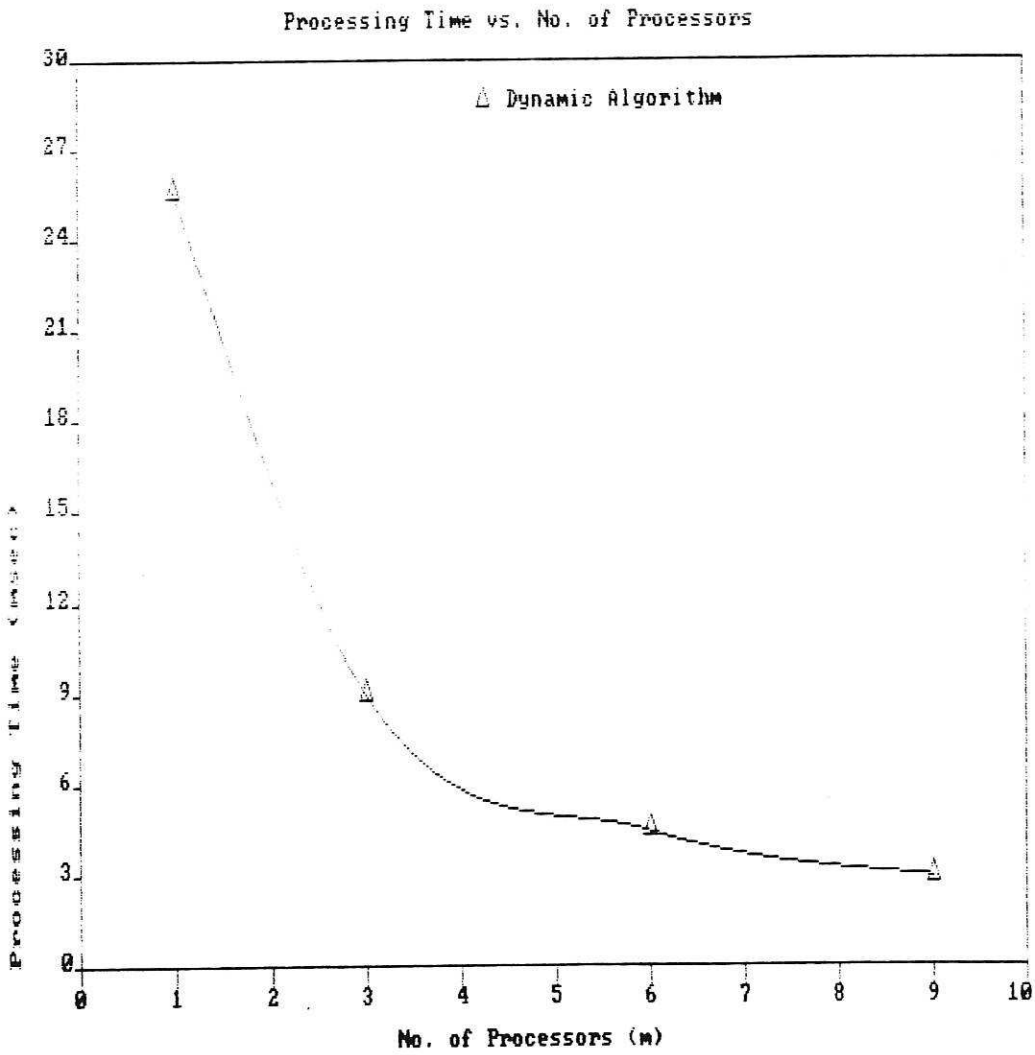

Figure 6. Processing-Time of the Dynamics.

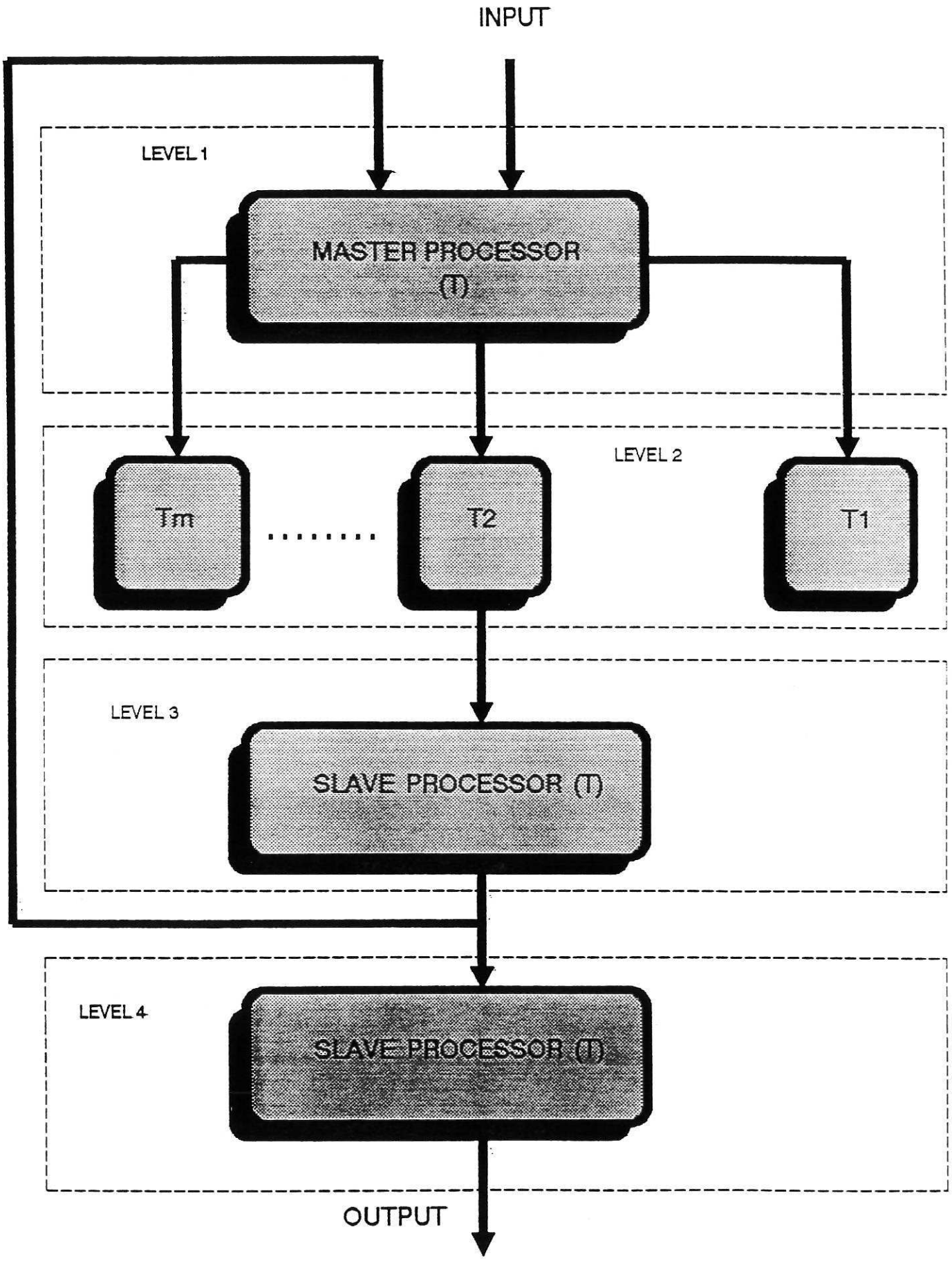

Figure 7. A Four-Levels Network for Solving the Gaussian-Elimination.

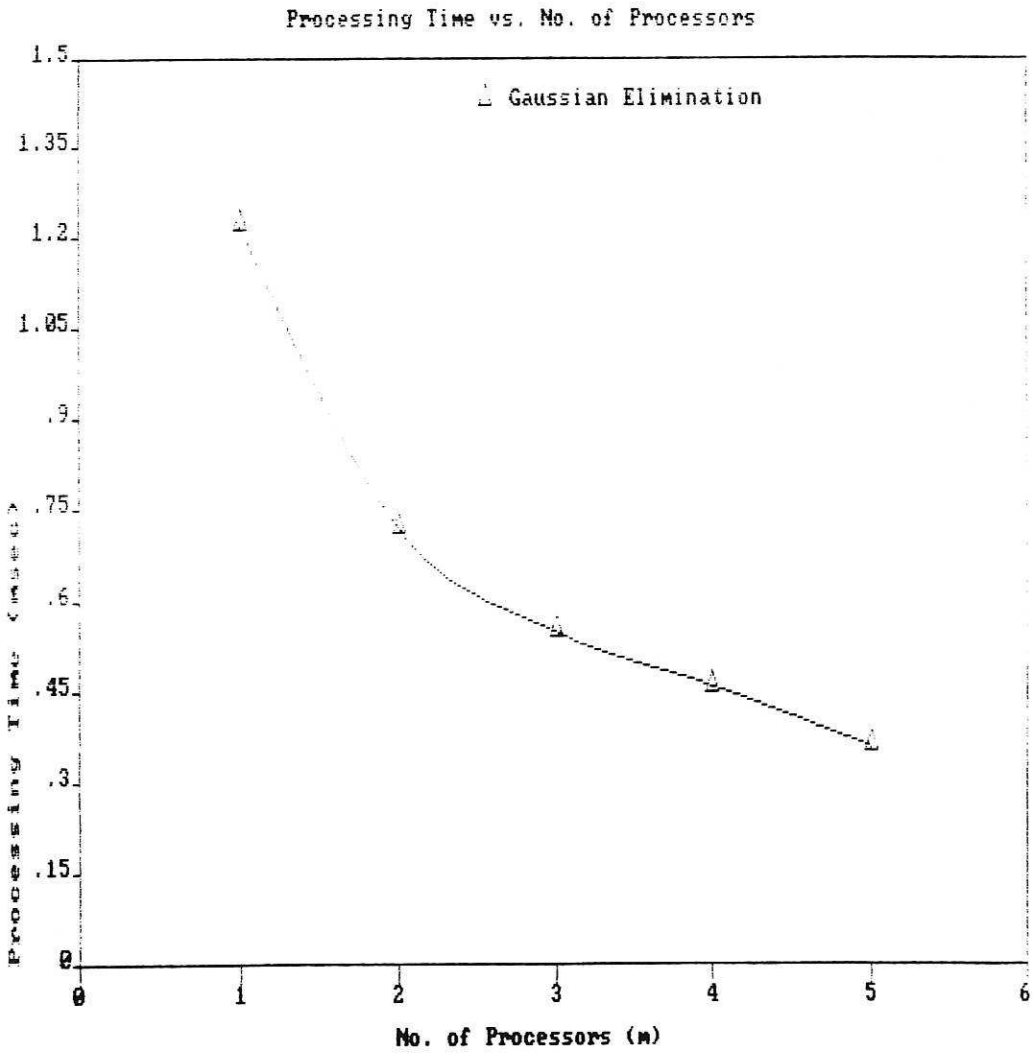

Figure 8. Processing-Time of the (GE).