Monograph:
Zomaya, A.Y. and Morris, A.S. (1989) Robot Inverse Dynamics Computation Via VLSI Distributed Architects. Research Report. Acse Report 350 . Dept of Automatic Control and System Engineering. University of Sheffield

Robot Inverse Dynamics Computation Via VLSI Distributed Architectures

by

A. Y. ZOMAYA

A. S. MORRIS

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

Research Report No. 350

Feburary 1989

Abstract

The computation of the highly coupled dynamic equations has always posed a bottleneck in real-time dynamic control of robot manipulators. Recent advances in VLSI technology make it possible to implement new algorithms that compute these equations and meet real-time constraints.

Parallel processing techniques can now be used to reduce the computation time for models of a highly mathematical nature such as the dynamical modelling of robot manipulators. In this work a semi-customized symbolic form of the Lagrange-Euler is divided into subtasks and distributed on a parallel processing sysem. The development system used consists of an **INMOS TRANSPUTER** (a VLSI single chip computer) running the **OCCAM** concurrent programming language. Further, this network is used to introduce parallelism by using different task allocation stratgies which flow naturally from the Lagrange-Euler formulation. The cost-effectivness and speed of the scheme is demonstrated by applying it to a case study (Stanford arm). Comparisons are made between uniprocessing (Von Neumann) and parallel implementations of the algorithm. Several measures such as *Utilization*, *Efficiency*, and *Speed up* are used to evaluate the performance of the employed networks and task-allocations.

1. Introduction

The last few years have seen remarkable achievements in the field of Robotics. The control of most existing robot manipulators is relatively simple and well defined: it neglects system dynamics and is based on a servo mechanism at each joint. However, sophisticated control algorithms are needed to facilitate the design of the next generation robot arms which can interact efficiently with unstructured and poorly defined environments. Several state of the art schemes had been proposed recently that require heavy computations which hinder their practical implementation [8].

To achieve an accurate real-time application of the control algorithm, it must be computed within a sampling rate of no less than 60 Hz. Furthermore, the quality of the control is greatly affected by its ability to accommodate the dynamic behaviour of the manipulator [46]. Hence, the dynamic equations must be evaluated repeatedly during the control loop sampling interval to avoid undesirable and serious degrading performance of the robot arm. So, the execution time for computing the dynamics partially enhances the feasibility of real-time implementations of the controller.

The dynamics consist of a set of differential, coupled, non-linear, and matrix oriented equations which governs the applied forces/torques values. Many researchers have proposed several simplified forms of the dynamics based on Lagrangian and Newtonian energetic principles [6,44].

The Lagrange-Euler (LE) [3,39,40] has high computational complexity but is a very well structured and systematic representation. The Recursive Lagrangian [11] gives good computational results

but destroys the structure of the equations. The Newton-Euler (NE) [31,37,48] has a very efficient computational formulation but with untidy recursive equations. Other approaches include the tabulation techniques [42] which suffers from serious difficulties owing to the enormous computer memory requirements; Kane's dynamic equations [19], and the Generalized D'Alembert [27]. Of the previous methods the most commonly used are the (LE) and (NE). The interaction and equivalence of these schemes has been shown by [43]. In this paper the (LE) is addressed.

The problem is solved for a 6 dof Stanford arm. The results and discussions are presented in the following order; Section (2) describes previous work by other researchers. Section (3) presents the development system used to implement this work. Section (4) addresses the dynamic problem. Parallelism is introduced in section (5) and (6). Conclusions and summary are given in section (7).

2. Previous Work

Previously, there have been two main approaches to tackle the problem of computing the inverse dynamics. The first of these is to reduce the complexity of the model, recognising that the robot performance suffers as a consequence. Bejczy [3] employed this concept and neglected the coriolis and centripetal effects by assuming low speed operating conditions. Ignoring these terms will result in a notable "vibration" of the robot arm at high speeds due to large errors in computing the forces and torques [51]. The other alternative is to use a stand alone computer system, which might lead to an increased development cost [31]. In this work a third option of parallel processing is introduced. A semi-customized symbolic formulation of the (LE) [51] is distributed on a number of cooperating general purpose processors. The device being considered is the **INMOS TRANSPUTER** and its programming langauge **OCCAM**.

Several parallel architectures have been proposed by researchers to solve for the inverse dynamics problem. The pioneering work of Luh and Lin [32], based on a generalization of the branch-and-bound algorithm, exhibits several significant limitations. Most importantly, their proposed architecture does not fully consider the recursive structure of the (NE) and the sequential dependencies of the algorithm. Furthermore, the system suffers from load unbalances because some of the processors are under utilized and the interprocessor communication and synchronization of the (NE) are ignored.

Orin et. al. [38] proposed a pipeline design for the (NE) that eliminates some of the performance degradation problems associated with interprocessor communications which appear in the computation of the (NE) using parallel processing techniques. However, the performance of the proposed design was not analyzed and compared with the serial (uniprocessor) implementation.

Lathrop [24] proposed two parallel algorithms using special purpose processors. First, is a linear parallel algorithm which is related to the Luh and Lin method [32]. The second is a logarithmic-parallel-algorithm based on the partial sum technique. Both approaches exhibit massive buffering which degrades the performance and causes complicated intertask communication structures, and hence no

practical implementation had been made. Liao and Chern [29] used a cross-bus array processor (CBAP), which utilizes a long set of bit-parallel processors. The main disadvantages of this approach are the unfully-utilized processing elements and the vulnerability to hardware failures. Kasahara and Narita [20] proposed a parallel processing scheme which employs two scheduling algorithms; depth-first/implicit-heuristic-search and critical-path/most-immediate-successors-first. The algorithm was implemented on an actual multiprocessor system to prove its effectiveness. Lee and Chang [25] introduced a method based on the recursive doubling algorithm with a modified inverse perfect shuffle interconnection scheme between a set of parallel processors. Their approach may not be cost efficient and fault-tolerant due to the complexity and expensive interconnection structure among the processors. Vukobratovic et. al. [47] recently proposed an algorithm that employs a modified branch-and-bound (BB) method combined with the largest-processing-time-first algorithm (LPTF). An actual implementation had been made and good results were obtained, but the issues of intercommunication and intermediate buffering were neglected which degrades the performance in actual situations. Finally, some more work in this area can be found in the literature [2,5,36,41,50].

Most of the previous attempts did not involve implementation on an actual parallel processing system. Results are presented in terms of the number of multiplications/additions and their theoretical equivalent of processor clock cycles. The results obtained in this work are the outcome of the actual implementation of the algorithm. The different task allocations are executed by a *distributed processor development system*. Hence, these results not only represent the processing-time of multiplications/additions but also the delays caused by the communication between different processors and some other problems that might rise from hardware and software limitations.

3. The Transputer and Parallel Processing

The discipline of computer architecture is now in a transitional stage because of the rapid advances of VLSI technology. This advent is weighting all the arguments in favour of parallel processing techniques [12,22,49].

Parallelism is achieved by distributing the job over a number of processors, ideally in such a way that all the processors are fully utilized. To achieve that, highly parallel structures have evolved, and many have been built to meet the increasing demand for more computing power and higher processing speed [10].

The **INMOS TRANSPUTER** is a pioneering device that fills this gap, and it can be considered to be the ideal component for fifth generation computers. The T800 Transputer in (Fig.1) which is used in this work is a 32 bit microcomputer with 4 Kbytes 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. This makes the Transputer one of the first 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 minmial wiring, the Transputer uses point to point serial communication links for direct connection to other Transputers.

OCCAM is a high level language developed by **INMOS** to run on the Transputer [13,14,15,16,21] and optimise its operation. It is simple, block structured, and supports both sequential (**SEQ**) and parallel (**PAR**) features on one or more Transputer which can be used to facilitate simulation, modelling and control of complicated physical systems [9,17,18].

4. Manipulator Dynamics

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 form which is the final outcome of solving the dynamics :

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

where $\tau(t)$ is an n $\times$ 1 applied force/torque vector for joint actuators; $\Theta(t)$, $\dot{\Theta}(t)$, and $\ddot{\Theta}(t)$ are n $\times$ 1 vectors representing position, velocity and acceleration respectively; $\mathbf{D}(\boldsymbol{\Theta})$ is an n $\times$ n effective and coupling inertia matrix; $\mathbf{C}(\boldsymbol{\Theta},\dot{\boldsymbol{\Theta}})$ is an n $\times$ 1 Corioilis and Centripetal effects vector; and $\mathbf{h}(\boldsymbol{\Theta})$ is an n $\times$ 1 gravitational force vector, where (n) is the degree of freedom (dof).

The very general form of eq.(1) is important in state space and modern control applications [26,34,45], however, it can't be utilized unless simplified [3,4,28,30,33,35,51].

In this work a semi-customized symbolic form is used. The formulation was first introduced by Bejczy [3] and later by Paul [40], then refined and further simplified by Zomaya and Morris [51]. The conventions used are the same as those of Bejczy [3] and Paul [40], being based on the Denavit-Hartenberg (DH) [7] representation.

The dynamic formulation is divided into two main parts :

(**I**) The vectors δ_l, and $\mathbf{d}_l$ which describe the differential rotation, and differential transformation of link (l) respectively. These vectors are customized and symbolically expressed for a certain type of manipulator. The general description of δ_i^l, and $\mathbf{d}_i^l$ is given in Paul [40];

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

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

(II) This part describes a general formulation of the inertial, coriolis, centripetal and gravitational effects. However, it can be further customized if a specific manipulator is used.

As given by Paul [40] :

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

More simplifications can be achieved by expanding eq.(4) to remove the multiplication by zero operations. In the following discussion the Stanford manipulator is considered. Assume a matrix (**E**) such that:

$$\mathbf{E} = \begin{bmatrix} \mathbf{e} & 0 \\ 0 & 0 \end{bmatrix} \tag{5}$$

where (**e**) is a 3×3 matrix. Using the trace operator,

$$\mathbf{D}_{ij} = \sum_{l=max(i,j)}^{n} \sum_{m=1}^{3} e_{mm} \tag{6}$$

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

$$\begin{aligned} &= J_{11}^l \begin{bmatrix} \delta_{iy} \\ \delta_{iz} \end{bmatrix}_l \begin{bmatrix} \delta_{jy} \\ \delta_{jz} \end{bmatrix}_l + J_{22}^l \begin{bmatrix} \delta_{ix} \\ \delta_{iz} \end{bmatrix}_l \begin{bmatrix} \delta_{jx} \\ \delta_{jz} \end{bmatrix}_l + J_{33}^l \begin{bmatrix} \delta_{ix} \\ \delta_{iy} \end{bmatrix}_l \begin{bmatrix} \delta_{jx} \\ \delta_{jy} \end{bmatrix}_l + J_{44}^l \begin{bmatrix} d_{ix} \\ d_{iy} \\ d_{iz} \end{bmatrix}_l \begin{bmatrix} d_{jx} \\ d_{jy} \\ d_{jz} \end{bmatrix}_l \\ &+ J_{24}^l \left(\begin{bmatrix} \delta_{jx} \\ \delta_{iz} \end{bmatrix} \begin{bmatrix} d_{iz} \\ -d_{jx} \end{bmatrix} + \begin{bmatrix} \delta_{ix} \\ \delta_{jz} \end{bmatrix} \begin{bmatrix} d_{jz} \\ -\delta_{jz} \end{bmatrix} \right)_l + J_{34}^l \left(\begin{bmatrix} \delta_{iy} \\ \delta_{jx} \end{bmatrix} \begin{bmatrix} d_{jx} \\ -d_{iy} \end{bmatrix} + \begin{bmatrix} \delta_{jy} \\ \delta_{ix} \end{bmatrix} \begin{bmatrix} d_{ix} \\ -d_{jy} \end{bmatrix} \right)_l \end{aligned} \tag{7}$$

In a similar approach to describe the coriolis and centripetal effects ;

$$\mathbf{D}_{ij} = \sum_{l=max(i,j)}^{n} tr\ (\ \Delta_j^l\ \Delta_k^l\ \mathbf{J}^l\ \Delta_i^{lT}\ ) \tag{8}$$

assuming a matrix (**U**) such that :

$$\mathbf{U} = \begin{bmatrix} \mathbf{u} & 0 \\ 0 & 0 \end{bmatrix} \tag{9}$$

where (**u**) is a 3×3 matrix. Using the trace operator will yield,

$$\mathbf{C}_{ijk} = \sum_{l=max(i,j,k)}^{n} \sum_{m=1}^{3} u_{mm} \tag{10}$$

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

$$\begin{aligned}
&= J^l_{11}\ \delta^l_{jx} \left(\begin{bmatrix} \delta_{ky} \\ \delta_{iy} \end{bmatrix} \begin{bmatrix} \delta_{iz} \\ -\delta_{kz} \end{bmatrix} \right)_l + J^l_{22}\ \delta^l_{ix}\delta^l_{jy} \left(\begin{bmatrix} \delta_{kz} \\ 1 \end{bmatrix} \begin{bmatrix} 1 \\ -\delta_{kx} \end{bmatrix} \right)_l + J^l_{33}\ \delta^l_{jz} \left(\begin{bmatrix} \delta_{kx} \\ \delta_{ix} \end{bmatrix} \begin{bmatrix} \delta_{iy} \\ -\delta_{ky} \end{bmatrix} \right)_l \\
&\quad + J^l_{44} \left(d_{ix} \begin{bmatrix} \delta_{jy} \\ \delta_{jz} \end{bmatrix} \begin{bmatrix} d_{kz} \\ -d_{ky} \end{bmatrix} + d_{iy} \begin{bmatrix} \delta_{jz} \\ \delta_{jx} \end{bmatrix} \begin{bmatrix} d_{kx} \\ -d_{kz} \end{bmatrix} + d_{iz} \begin{bmatrix} \delta_{jx} \\ \delta_{jy} \end{bmatrix} \begin{bmatrix} d_{ky} \\ d_{kx} \end{bmatrix} \right)_l \\
&\quad + J^l_{24} \left(\delta_{ix} \begin{bmatrix} \delta_{jz} \\ \delta_{jy} \end{bmatrix} \begin{bmatrix} d_{ky} \\ d_{kz} \end{bmatrix} \begin{bmatrix} \delta_{jx} \\ \delta_{jy} \end{bmatrix} \begin{bmatrix} d_{ky} \\ -d_{kx} \end{bmatrix} + \delta_{jy} \begin{bmatrix} \delta_{kx} \\ \delta_{kz} \end{bmatrix} \begin{bmatrix} d_{ix} \\ d_{iz} \end{bmatrix} - d_{iy} \begin{bmatrix} \delta_{jx} \\ \delta_{jz} \end{bmatrix} \begin{bmatrix} \delta_{kx} \\ \delta_{kz} \end{bmatrix} \right)_l \\
&\quad + J^l_{34} \left(\delta_{iy} \begin{bmatrix} \delta_{jy} \\ \delta_{jz} \end{bmatrix} \begin{bmatrix} d_{kz} \\ -d_{ky} \end{bmatrix} + \delta_{jz} \begin{bmatrix} \delta_{kx} \\ \delta_{ky} \end{bmatrix} \begin{bmatrix} d_{ix} \\ d_{iy} \end{bmatrix} + \delta_{ix} \begin{bmatrix} \delta_{jx} \\ \delta_{jz} \end{bmatrix} \begin{bmatrix} d_{kz} \\ -d_{kx} \end{bmatrix} - d_{iz} \begin{bmatrix} \delta_{jy} \\ \delta_{jx} \end{bmatrix} \begin{bmatrix} \delta_{ky} \\ \delta_{kx} \end{bmatrix} \right)_l
\end{aligned} \tag{11}$$

where

$$\begin{aligned}
J^l_{24} &= y_l\, m_l \\
J^l_{34} &= z_l\, m_l \\
J^l_{11} &= (-I^l_{11} + I^l_{22} + I^l_{33})/\ 2 \\
J^l_{22} &= (I^l_{11} - I^l_{22} + I^l_{33})/\ 2 \\
J^l_{33} &= (I^l_{11} + I^l_{22} - I^l_{33})/\ 2 \\
J^l_{44} &= m_l
\end{aligned}$$

and the gravitational effects are given by,

$$\mathbf{h}_i = g \sum_{l=i}^{n} m_l\, \Psi_l\, \mathbf{r}^l_l \tag{12}$$

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,

$$\Psi_l = \begin{bmatrix} s\alpha\ \delta_{iz} - c\alpha\ \delta_{iy} \\ c\alpha\ \delta_{ix} \\ s\alpha - \delta_{ix} \\ s\alpha\ d_{iy} + c\alpha\ d_{iz} \end{bmatrix}_l \tag{13}$$

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

The previous dynamic equations eq.(7,11,12) will be assumed throughout this work.

5. Parallel Computation of the Dynamics

This section describes the procedure used to compute the inverse dynamics on a distributed processor system. The task (algorithm) is decomposed into a set of subtasks (processes). In this case the subtasks represent the different terms of eq.(1). The objective of this work is to find a cost-effective architecture, by distributing the task over several network configurations with different number of processors (the name processor and transputer are used interchangably), to reach an optimum configuration that computes the input joint forces and torques. The main difference between the configurations used is in respect of how the task is divided, that is, the amount of work carried out by each processor and the overhead caused by the communication between the different processors in the network. The networks are evaluated according to the total processing time and processor utilization measures. Quantification of speed up, utilization, and efficiency with real-time implementation results are included.

5.1 The Analysis

The analysis is divided into two sub levels :

Off–Line Level :

A specific manipulator has been chosen (the Stanford arm). The customized part of the algorithm is prepared and the zero-value subtasks are located and excluded from the on-line implementation. These simplifications are carried out using different assumptions [3,28] such as symmetry and reflexive coupling ($C_{ijk} = -C_{kji},\ i,\ k \geq j$). Then the whole algorithm is executed using one processor to identify the processing-time of each subtask, so that all the sources of overhead and communication bottleneck are avoided in the real-time situation.

In the case of the Stanford arm there are three main tasks :

task 1 : divided into 17 subtasks representing the effective and coupling inertia terms.

task 2 : divided into 43 subtasks representing the coriolis and centripetal effects.

task 3 : divided into 4 subtasks.

which sums up to a total of 64 subtasks to be computed. A close examination of the dynamics shows a certain amount of parallelism with a large amount of sequentialism in the natural flow of the computations. The analysis shows that a main source of bottleneck is in computing the coupling inertia matrix (**D**) and matrix (1) and (2) of the coriolis and centripetal effects ($\mathbf{C}_{1jk}$, $\mathbf{C}_{2jk}$), approximately 24%, 28% and 21% respectively, of the total processing time.

On–Line Level :

This level deals with the actual implementation of the algorithm, and is discussed in the next section.

6. Scheduling Strategies

The networks used in this paper have the same basic structure, that is, a main processor (Scheduler or Controller) and a cluster of slave processors responsible for the computations and the number crunching. Two main points are kept in mind while doing the analysis and the task distribution; first, the proportion of subtasks that must be computed sequentially; second, allowing enough time for the communication of data between slave processors and trying to minimize this time as much as possible by enabling each processor to execute its job without the need for data from other processors. To avoid (I/O) bottleneck, the overall (I/O) of a processor must be reduced by increasing the size of its memory [23]. Hence, no redundant calculations are performed and the (DH) parameters, the link masses, inertias, and centre of mass reside on each transputer in the network to minimize the communication overhead.

The main processor (P_0) will supervise the network and send the required data to the slave processors (P_i, $i > 0$), that is, sending Θ, $\dot{\Theta}$, and $\ddot{\Theta}$; and receiving $\overline{f_i}$ which constitutes a partial sum of the force/torque vector τ (t) .

6.1 One-Processor Case (Von Neumann):

This is a trivial case because 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).

6.2 Three-Processors Case :

For this case a tree-structured network is used (Fig.2.) 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 subtasks are distributed as shown in (table 1). The total processing-time was found to be (8.96 msec). It's very important to 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.

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

Theoretically, in this case, the lower bound time (T_{lb}) or the ideal processing time that can be achieved by using more than one processor is given as

$$T_{lb} = T_{p(one\ processor\ case)} / m$$

where (m) is the number of processors. However, in actual hardware implementations there is an offset time (T_{of}) which is the difference between the actual (T_p) and the ideal (T_{lb})

$$T_{of} = T_p - T_{lb}$$

This deviation is due to (t_2) and hardware limitations.

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

6.3 Six and Seven-Processors Case :

The same procedure is followed here with different network configurations (Fig.3 and Fig.4). These architectures give more independance to each processor and increase the computing power to achieve better processing time.

In these configurations 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 times are (4.46 msec) and (3.82 msec) for the six-processors and seven-processors networks respectively. The scheduling strategies are shown in (table 2 and table 3).

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}	$\mathbf{C}_4$
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	-
$\mathbf{C}_5$	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

PROCESSOR						
P_1	P_2	P_3	P_4	P_5	P_6	P_7
C_{12}^1	D_{11}	C_{15}^2	C_{13}^2	D_{14}	$\mathbf{C}_4$	C_{14}^1
C_{13}^1	D_{12}	$\mathbf{C}_3$	C_{14}^2	D_{15}	$\mathbf{C}_5$	C_{15}^1
C_{22}^1	D_{13}	-	C_{23}^2	D_{16}	**h**	C_{24}^1
C_{23}^1	D_{22}	-	C_{24}^2	D_{24}	C_{46}^2	C_{25}^1
C_{16}^2	D_{23}	-	C_{44}^2	D_{25}	C_{56}^2	C_{44}^1
-	D_{33}	-	C_{55}^2	D_{26}	-	C_{45}^1
-	C_{26}^1	-	-	D_{35}	-	C_{55}^1
-	C_{46}^1	-	-	D_{44}	-	C_{25}^2
-	C_{56}^1	-	-	D_{46}	-	-
-	-	-	-	D_{55}	-	
-	-	-	-	D_{66}	-	
-	-	-	-	C_{45}^2	-	

Table 3. Seven–Processors Task Allocation

Some points about the values of (T_p) quoted above should be noted. Firstly, the goal of enhancing the performance of the network is attained by making sure that information can flow to and from the processsing elements with sufficient speed, rather than by maximizing the computational bandwidth of the processing elements (the number of operations per second that the processor can deliver). This is accomplished by organizing the processors which need to communicate with one another in such a way as to make the transfer of data smooth and cost effective. Hence, the situation of two processors communicating with each other through a third processor is avoided.

Secondly, the value of (T_p) is always higher than the lower bound (T_{lb}) because of the communication between the different processors. Fig.(5) shows the relation between the (T_{of}) and the number of processors (m).

Thirdly, the different subtasks must be divided so that the processors spend most of the time on useful computations whilst at the same time minimizing the communication between the different processors as much as possible.

6.4 Eight-Processors Case :

Satisfactory results were obtained by using the previous networks. Furthermore, an eight-processors network is employed to enhance the performance and to give higher processing power while maintaining desirable cost-effectivness and fault-tolerance (Fig.6). A total processing-time of (3.34 msec) is

obtained. The task scheduling is shown in (table 4). The flow of the algorithm in this case is given in a block diagram representation (Fig.7).

PROCESSOR							
P_1	P_2	P_3	P_4	P_5	P_6	P_7	P_8
D_{14}	D_{11}	C_4	C^1_{46}	C^2_{13}	C^1_{44}	C^1_{12}	C^1_{22}
D_{15}	D_{12}	C_5	C^3_{14}	C^2_{14}	C^2_{16}	C^1_{13}	C^1_{23}
D_{16}	D_{13}	h_2	C^3_{24}	C^2_{15}	C^2_{25}	C^1_{14}	C^1_{24}
D_{24}	D_{22}	h_3	C^3_{25}	C^2_{23}	C^2_{44}	C^1_{15}	C^1_{25}
D_{25}	D_{23}	h_5	C^3_{44}	C^2_{24}	C^2_{45}	C^1_{55}	C^1_{26}
D_{26}	D_{33}	-	C^3_{45}	-	C^2_{46}	C^1_{56}	C^1_{45}
D_{35}	h_4	-	C^3_{55}	-	C^2_{55}	-	-
D_{44}	-	-	-	-	C^2_{56}	-	-
D_{46}	-	-	-	-	C^3_{15}	-	-
D_{55}	-	-	-	-	-	-	-
D_{66}	-	-	-	-	-	-	-

Table 4. Eight–Processors Task Allocation

Ideally, a better (T_p) can be achieved if the number of processors is 64 under the assumption of assigning a processor per task (for the case of Stanford arm). However, a 64-processors network will degrade the performance because of the low cost-effectivness and fault-tolerance. Undoubtedlly, this will lead to lower processing time, and under-utilized processors in the network. The different values of (T_p) are reported in (table 5).

PROCESSING TIME (T_p) (msec)	
No. of Processors	*Described Scheme*
1	25.6
3	8.96
6	4.46
7	3.82
8	3.34

Table 5. Total Processing Time

The *Utilization* which 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 of the processors being actively involved in the execution of the whole job (table 6).

PROCESSOR UTILIZATION (%)								
No. of Processors	P_1	P_2	P_3	P_4	P_5	P_6	P_7	P_8
1	100	-	-	-	-	-	-	-
3	100	100	96.8	-	-	-	-	-
6	100	98.7	98.7	96.2	97.5	95	-	-
7	100	94.1	98.5	100	97	98.5	97	-
8	100	96.6	96.6	95	96.6	100	100	100

Table 6. Processor Utilization for Different Network Configurations

The improvement in speed *Speed up* (table 7) can be defined as:

$$S_m = T_{ps} \; / \; T_{pm}$$

where T_{ps} is the total processing-time of the sequntial algorithm, and T_{pm} is the total processing-time required to finish the execution of a job by an m-processors network.

SPEED UP (S_m)	
No. of Processors	*Described Scheme*
1	1.0
3	2.86
6	5.74
7	6.7
8	7.67

Table 7. Speed Up

The *Efficiency* (ϕ) of an m-processors network is displayed by the utilization rate of the available processors (Transputers):

$$\phi = S_m / m$$

in the ideal case $\lim_{S_m \to m} \phi = 1$ (table 8).

ϕ	
No. of Processors	*Described Scheme*
3	0.952
6	0.957
7	0.957
8	0.96

Table 8. Efficiency

Graphical illustrations of the results are given in (Fig.8,9,10).

7. Conclusion and Summary

The dynamical description of a typical 6-dof robot arm such as the Stanford arm is complicated and computationally expensive, which hinders the inclusion of the dynamics in real-time control applications.

This paper addressed the parallel-processing approach to solve for the inverse dynamic problem. A simplified semi-customized form of the dynamics based on the Lagrange-Euler formulation has been distributed over a parallel-processing system. The system was constructed by using the INMOS TRANSPUTER as its basic building element running the OCCAM programming language.

Different configurations have been suggested and very good real-time results obtained. The results have been compared with the sequential implementation of the algorithm and the superiority of using parallel-processing techniques are emphasised by using several measures (e.g. efficiency, speed up). As mentioned earlier, the achievement of an optimum network configuration is determined by the computation and communication structure of the task, as well as the computation and communication organization of the system components (Transputers).

Similar scheduling strategies are equally applicable for other types of robot control problems. It has already been shown that the application of the proposed configurations can provide an efficient solution for the problems of *Direct and Inverse Jacobian* formulations [52].

The work described has demonstrated how recent advances in VLSI technology can be used together with parallel-processing techniques to significantly speed up the dynamic modelling of robot manipulators. Suitable foundations have been set for the development of a wide range of control algorithms, unhindered by their excessive computational requirements.

Acknowledgement

The authors wish to thank Dr. G. Virk and Mr. I. Durkacz for their help in using the different facilities of the parallel processing laboratory in the Control Engineering Dept.- University of Sheffield. Particular thanks to Dr. G. Manson form the Computer Science Dept. for his help with the use of the (T800) Transputers and the debugging of the software.

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] 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.

[6] D'SOUZA, A. F. AND GARG, V. K., (1984). *Advanced Dynamics: Modeling and Analysis*, Prentive-Hall, Englewood Cliffs, N.J.

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

[8] FU, K. S., GONZALES, R. C., AND LEE, C. S. G., (1987). *Robotics: Control, Sensing, Vision, and Intelligence*, McGraw-Hill, New York.

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

[10] 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.

[11] 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.

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

[13] IEE,(1988), *Parallel Processing in Control- the Transputer and other Architectures, IEE Workshop, UCNW*, Bangor, Wales, U.K..

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

[15] INMOS,(1986), *IMS T800 Transputer*, Product Overview.

[16] INMOS,(1988), *IMS T800 Architecture*, Technical Note 6.

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

[18] 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.

[19] 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.

[20] 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.

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

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

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

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

[25] 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.

[26] LEE, C. S. G., (1983). "On the Control of Robot Manipulators," in *Proc. 27th of the Society of Photo-Optical Instrumentation Engineers, San Diego*, vol. 442, pp. 58-83.

[27] 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.

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

[29] LIAO, F. Y. AND CHERN, M. Y., (1985). "Robot Manipulator Dynamics Computation on a VLSI Processor," in *Proc. 1st Conf. on Supercomputing Systems, st. Petersburg, Florida*, pp. 116-121.

[30] LIN, C. S. AND CHANG, P. R., (1984). "Automatic Dynamics Simplification for Robot Manipulators," in *Proc. 23rd conf. on Decision and Control, Las Vegas*, pp. 752-759.

[31] 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.

[32] LUH, J. Y. S. AND LIN, C. S., (1982). "Scheduling of Parallel Computation for a Computer Controlled Mechanical Manipulator," *IEEE Trans. on Systems, Man, and Cybernetics*, vol. 12, no. 2, pp. 214-234.

[33] MEGAHED, S. AND RENAUD, M., (1982). "Minimization of the Computation Time Necessary for the Dynamic Control of Robot Manipulators," in *Proc. Conf., on Industrial Robot Technology 6th Int. Symp. Industrial Robots, Paris*, pp. 469-478.

[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., (1987). "Customized Computational Robot Dynamics," *J. Robotic Systems*, vol. 4, no. 4, pp. 503-526.

[36] NIGAM, R. AND LEE, C. S. G., (1985). "A Multiprocessor-Based Controller for the Control of Mechanical Manipulators," *IEEE J. Robotics and Automation*, vol. 1, no. 4, pp. 173-182.

[37] 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.

[38] ORIN, D. E., CHAO, H. H., OLSON, K. W., AND SCHRADER, W. W., (1985). "Pipeline/Parallel Algorithms for the Jacobian and Inverse Dynamics Computations," in *Proc. IEEE Int. Conf. on Robotics and Automation*, pp. 785-789.

[39] PAUL, R. P., (1972). "Modelling, Trajectory Calculation and Servoing of a Computer Controlled Arm," *Stanford Artificial Intelligence Laboratory, Stanford University, AIM 177.*

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

[41] 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.

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

[43] 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.

[44] TORBY, B. J., (1984). *Advanced Dynamics for Engineers*, Holt-Saunders, NewYork.

[45] 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.

[46] VUKOBRATOVIC, M. AND STOKIC, D., (1983). "Is Dynamic Control Needed in Robotic Systems, and, if so, to What Extent ?," *Int J. Robotics Res.*, vol. 2, no. 2.

[47] 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.

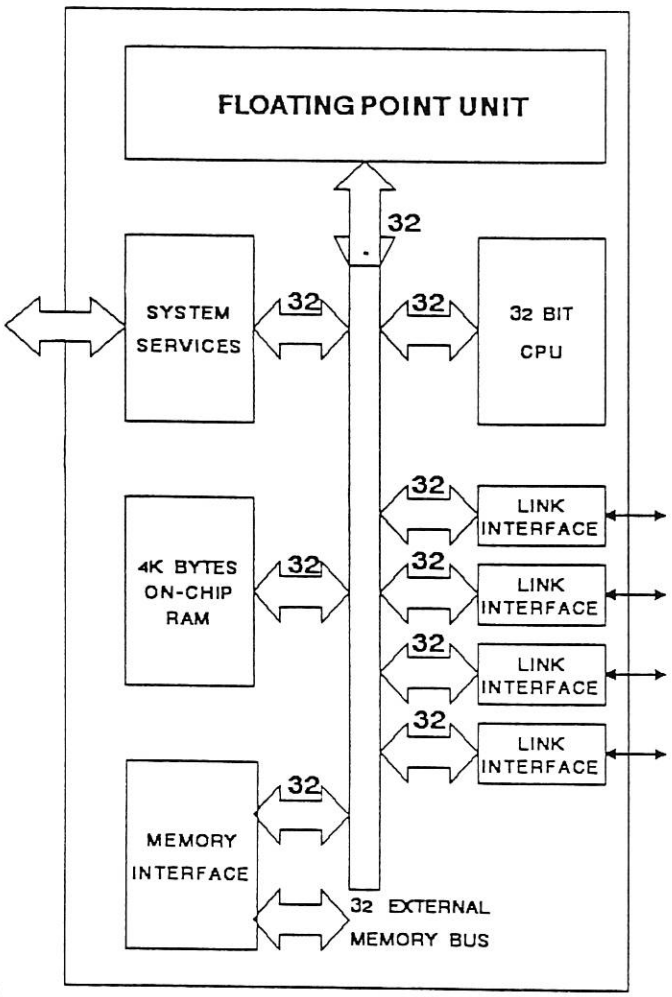
[48] WALKER, M. W. AND ORIN, D. E., (1982). "Efficient Dynamic Computer Simulation of Robotic Mechanisms," *Trans. ASME J. Dynamic Systems, Measurements, and Control*, vol. 104, pp. 205-211.

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

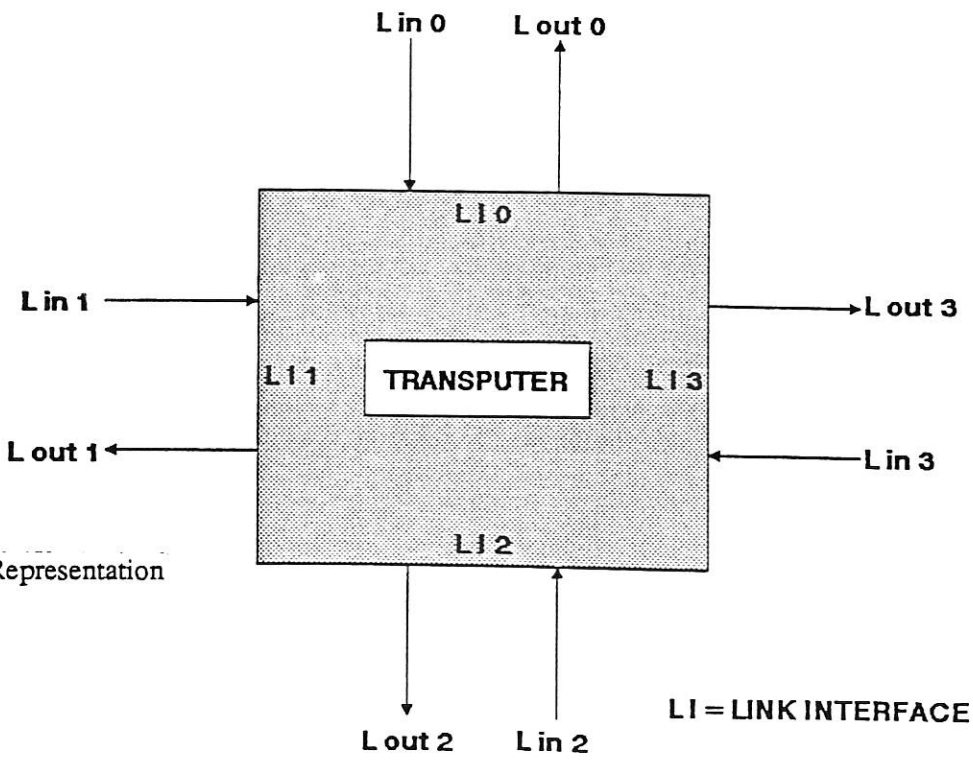
[50] ZHENG, Y. AND HEMAMI, H., (1986). "Computation of Multibody System Dynamics by a Multiprocessor Scheme," *IEEE Trans. on Systems, Man, and Cybernetics*, vol. 16, no. 1, pp. 102-110.

[51] 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.*.

[52] 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.*.

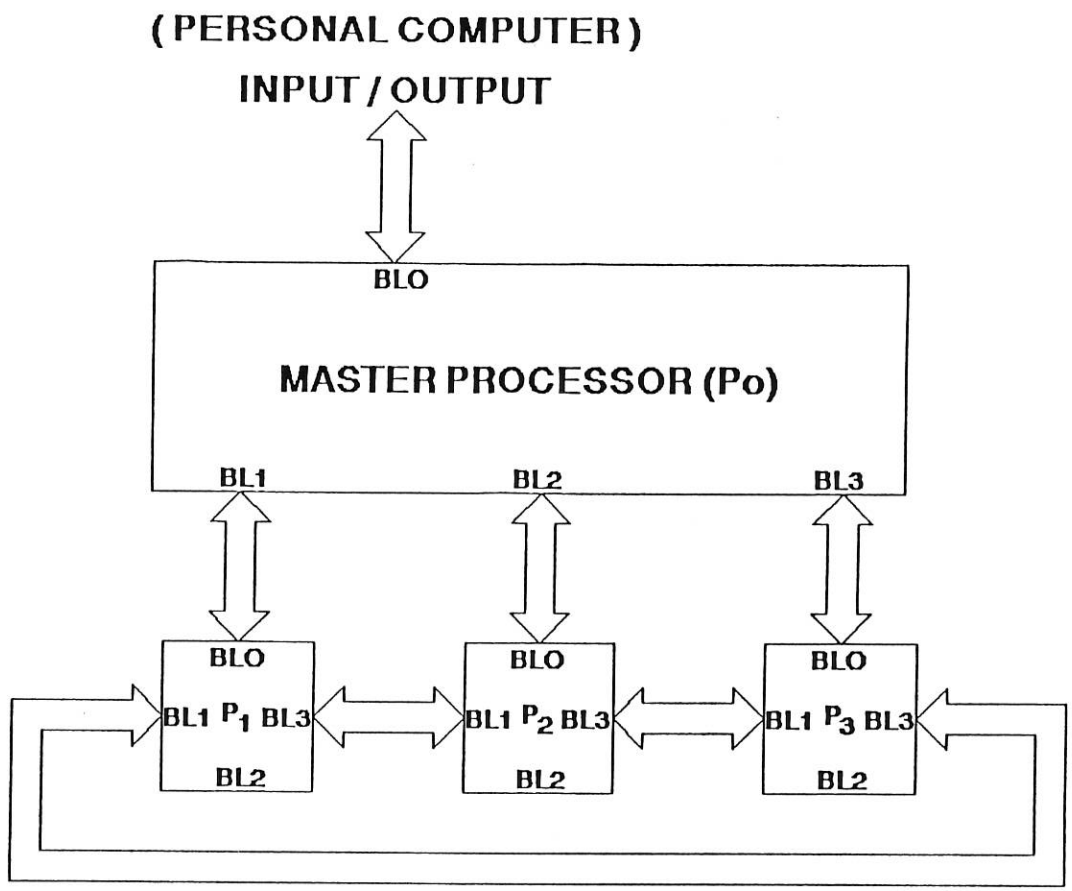


(a) General Representation



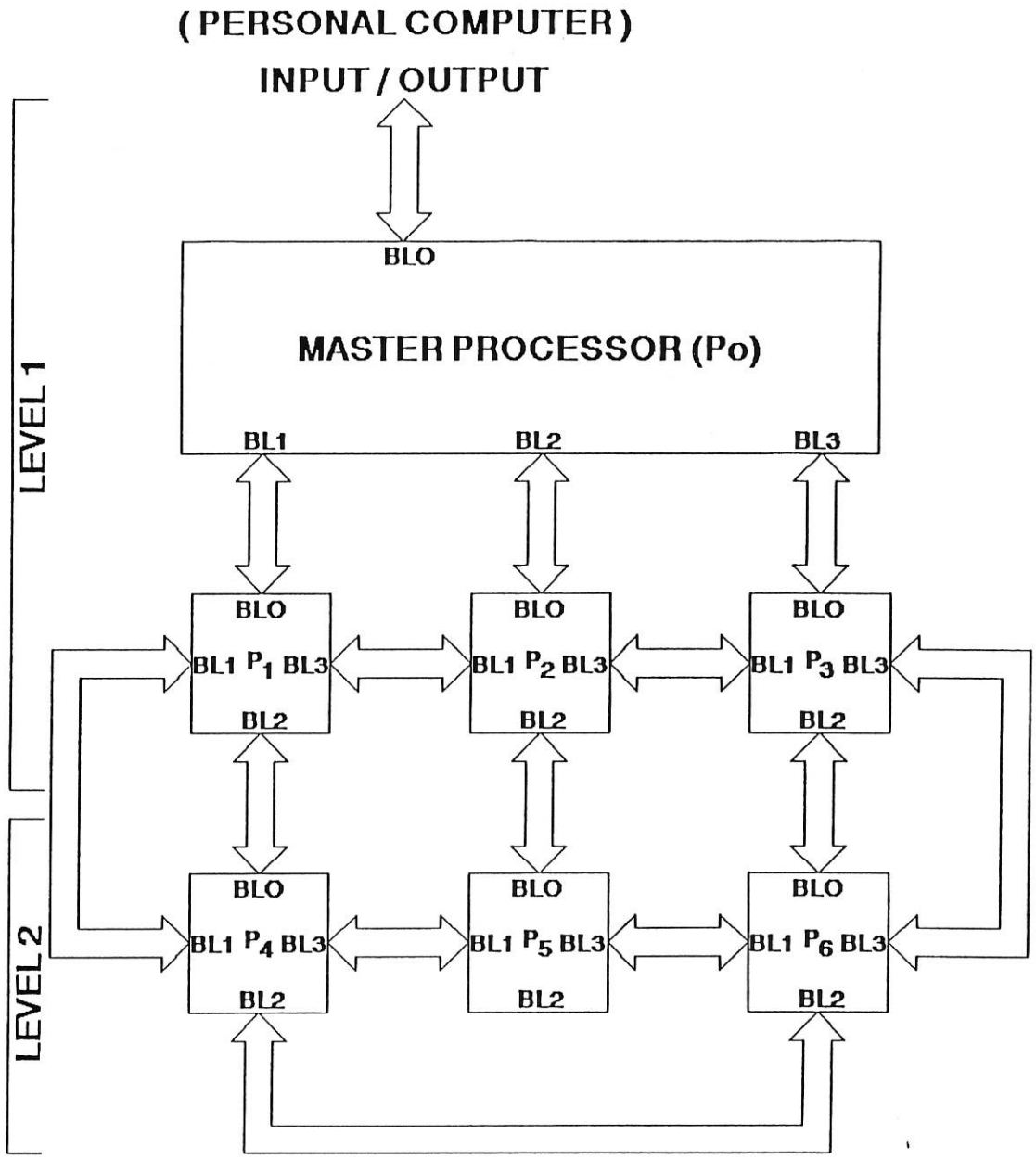
(b) Schematic Representation

Fig.1 The INMOS T800 Transputer



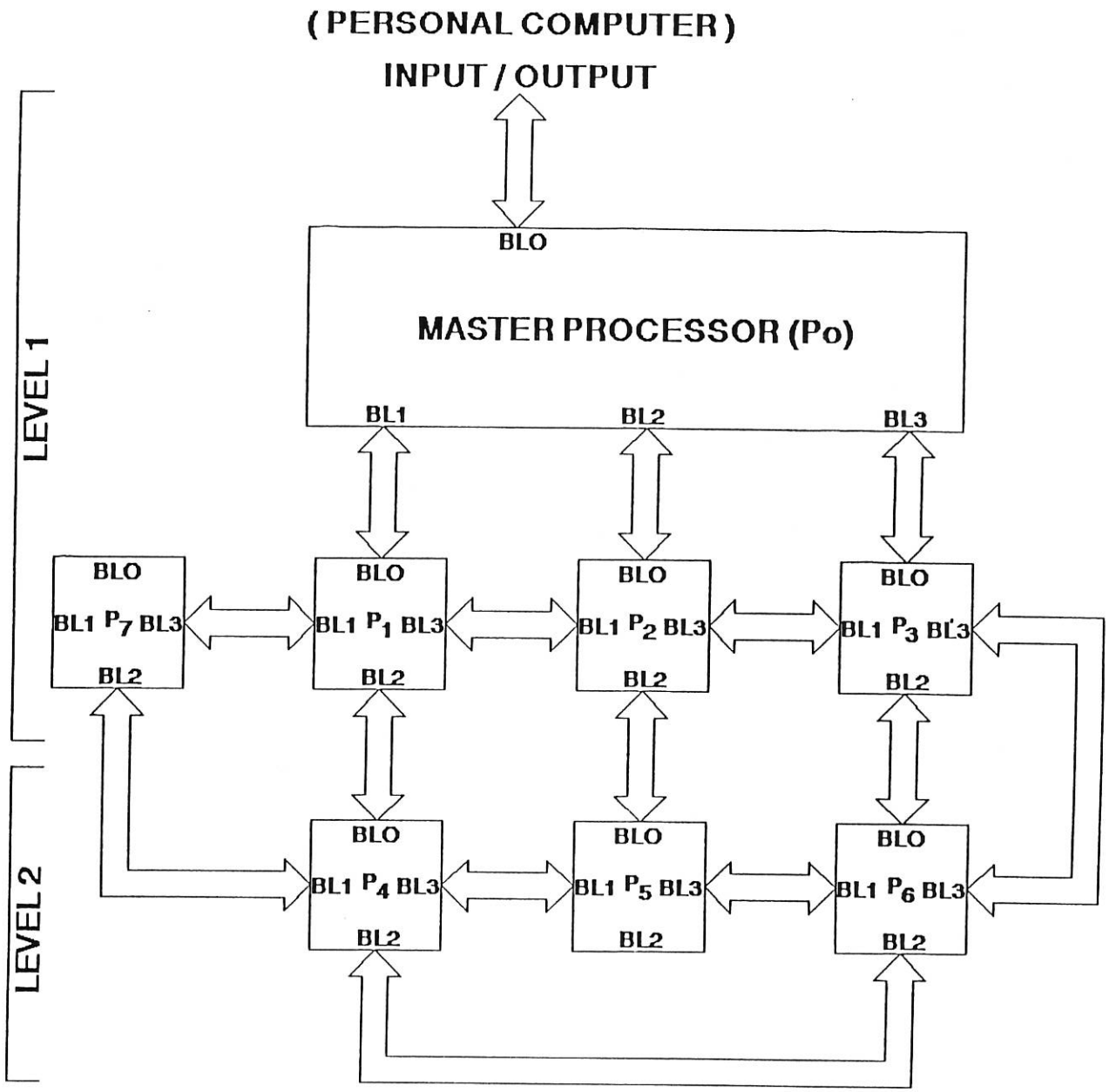
BL: BIDIRECTIONAL LINK
P : PROCESSOR OR TRANSPUTER

Fig.2 Three-Transputers Network



BL: BIDIRECTIONAL LINK
P : PROCESSOR OR TRANSPUTER

Fig.3 Six-Transputers Network



BL: BIDIRECTIONAL LINK
P : PROCESSOR OR TRANSPUTER

Fig.4 Seven-Transputers Network

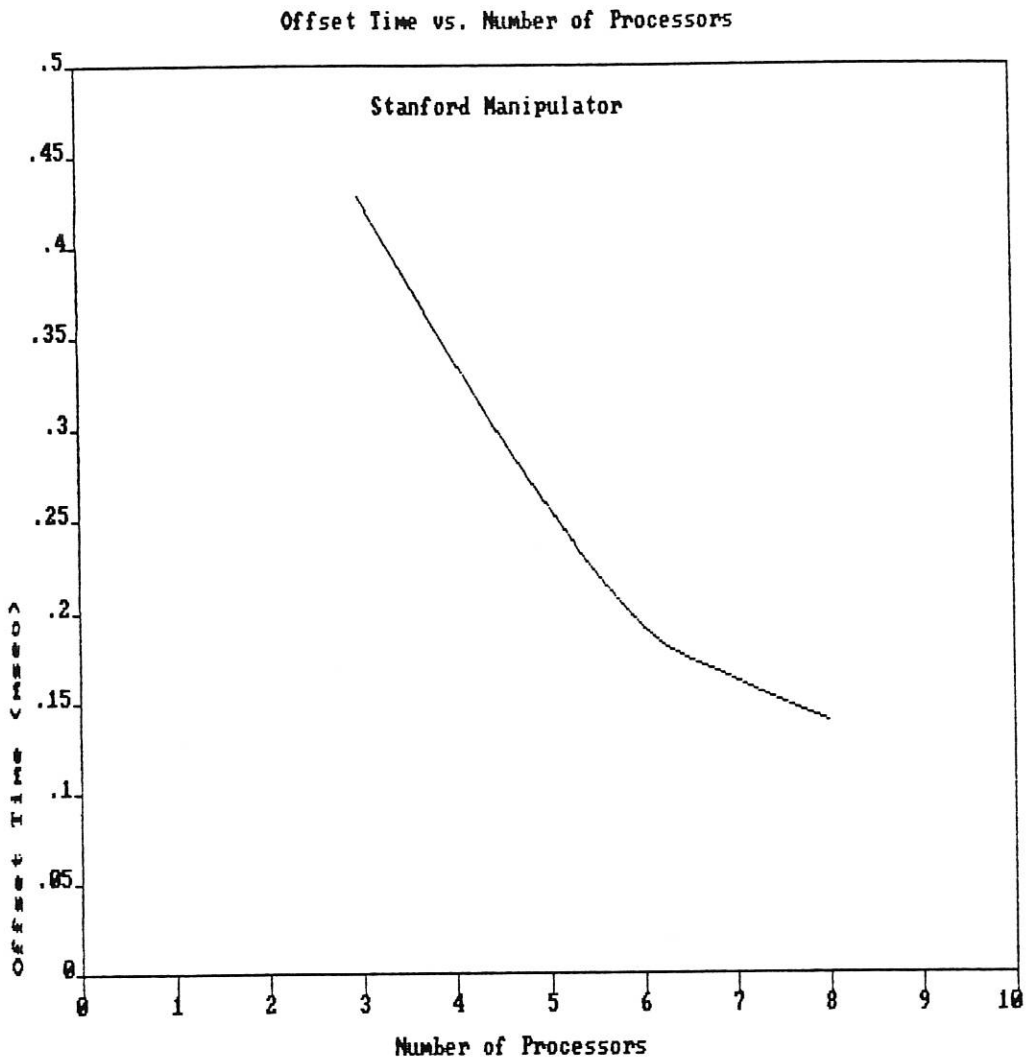


Fig.5 Offset-time Variation

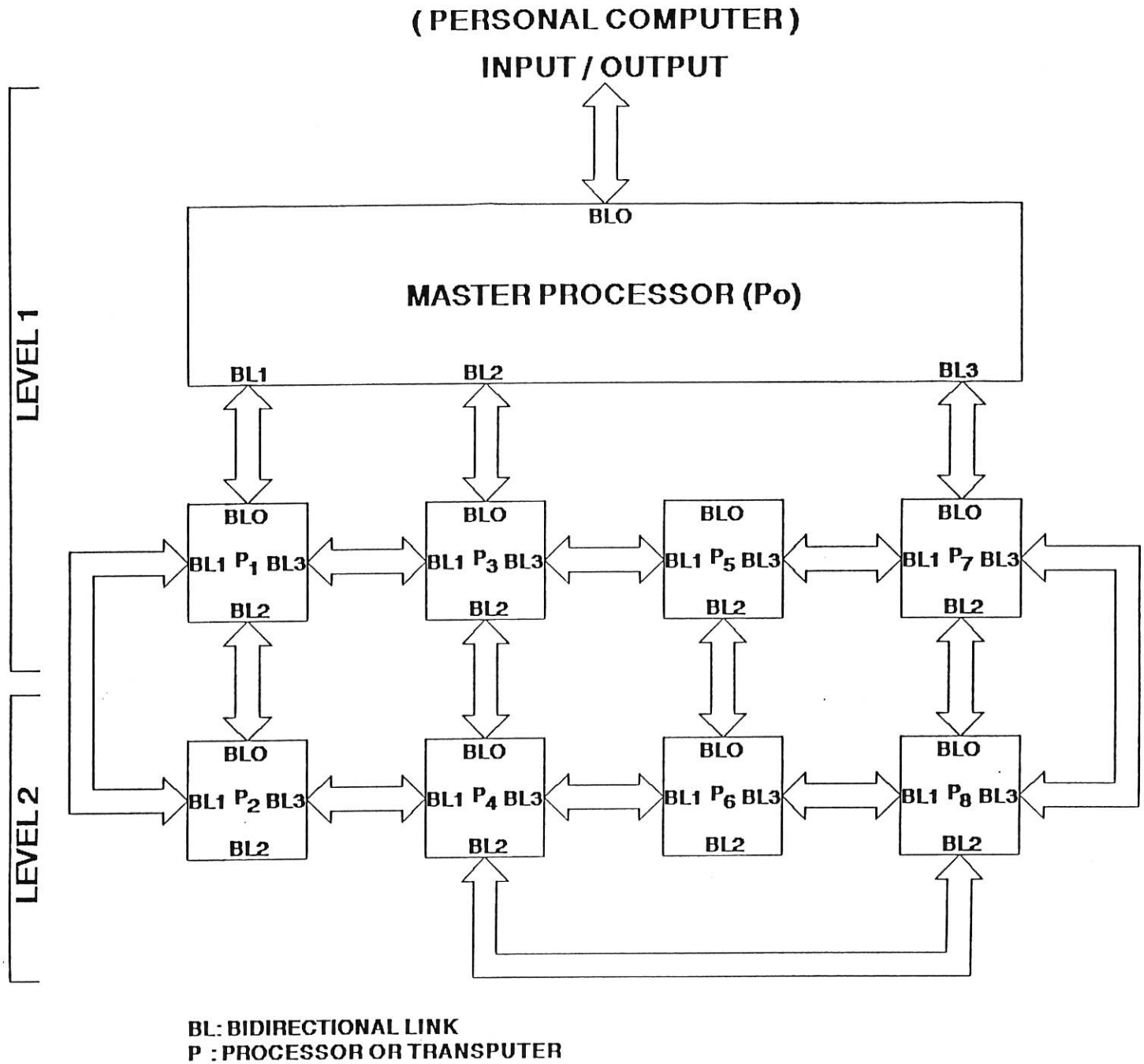


Fig.6 Eight-Transputers Network

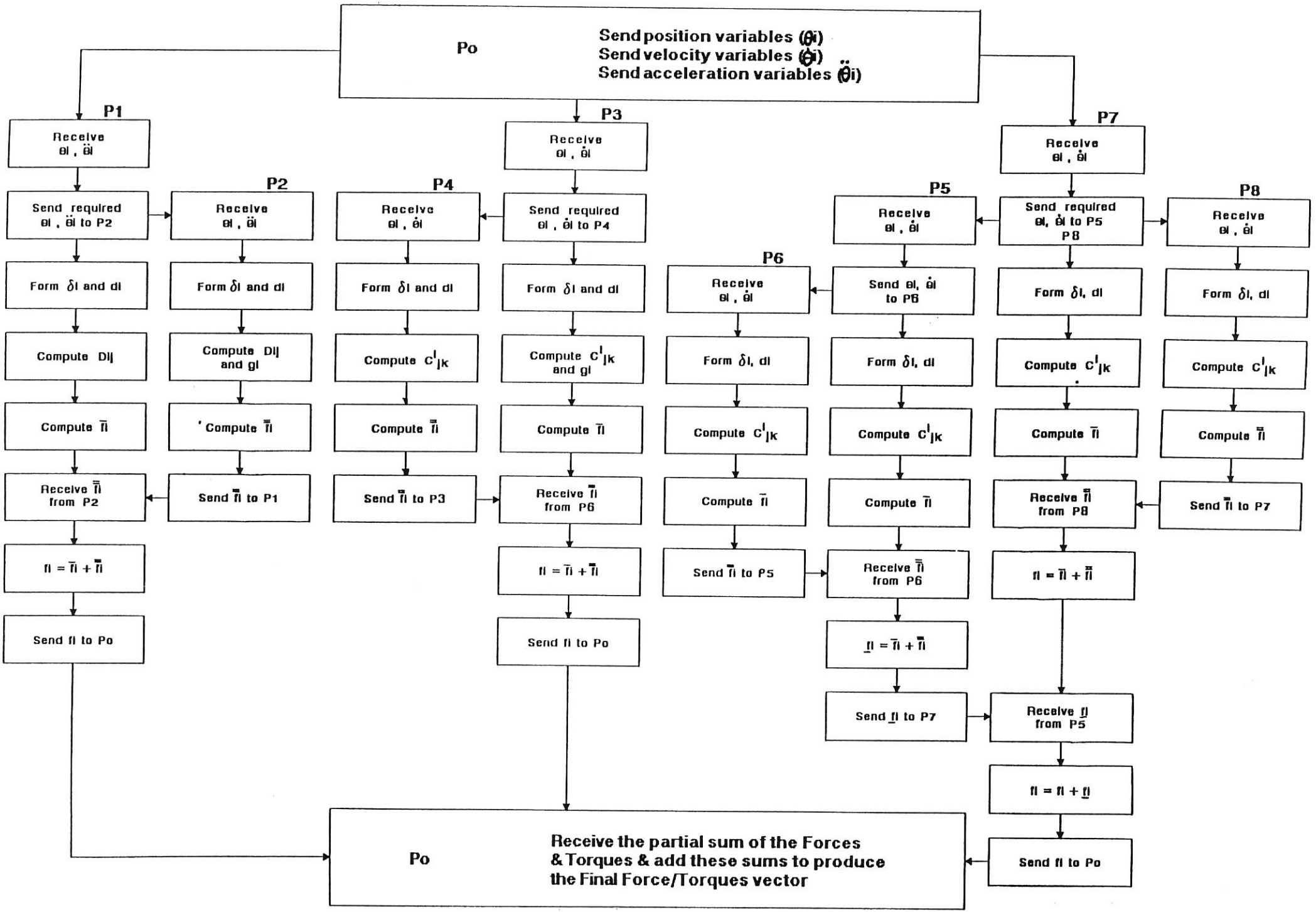


Fig.7 Block diagram representation of the Algorithm on an Eight-Transputers Network

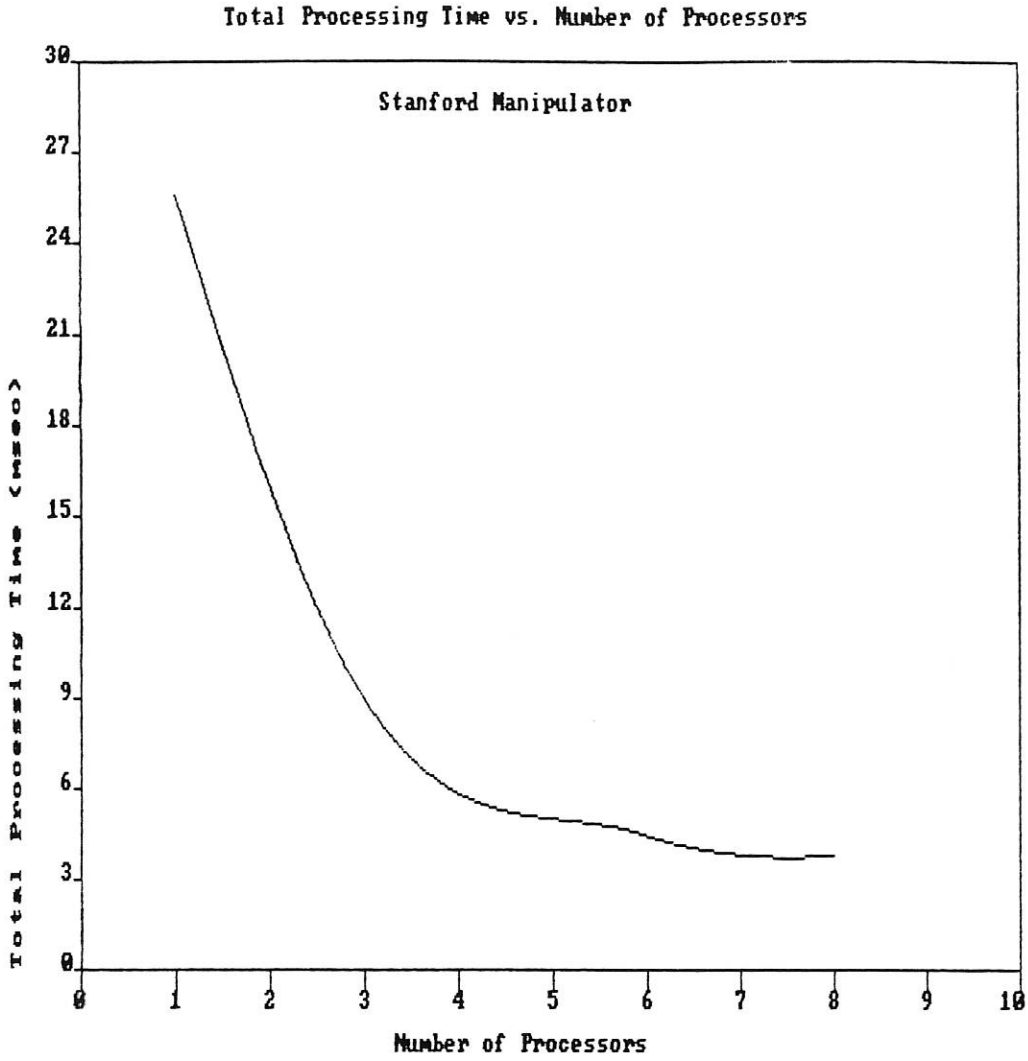


Fig.8 Total Processing Time

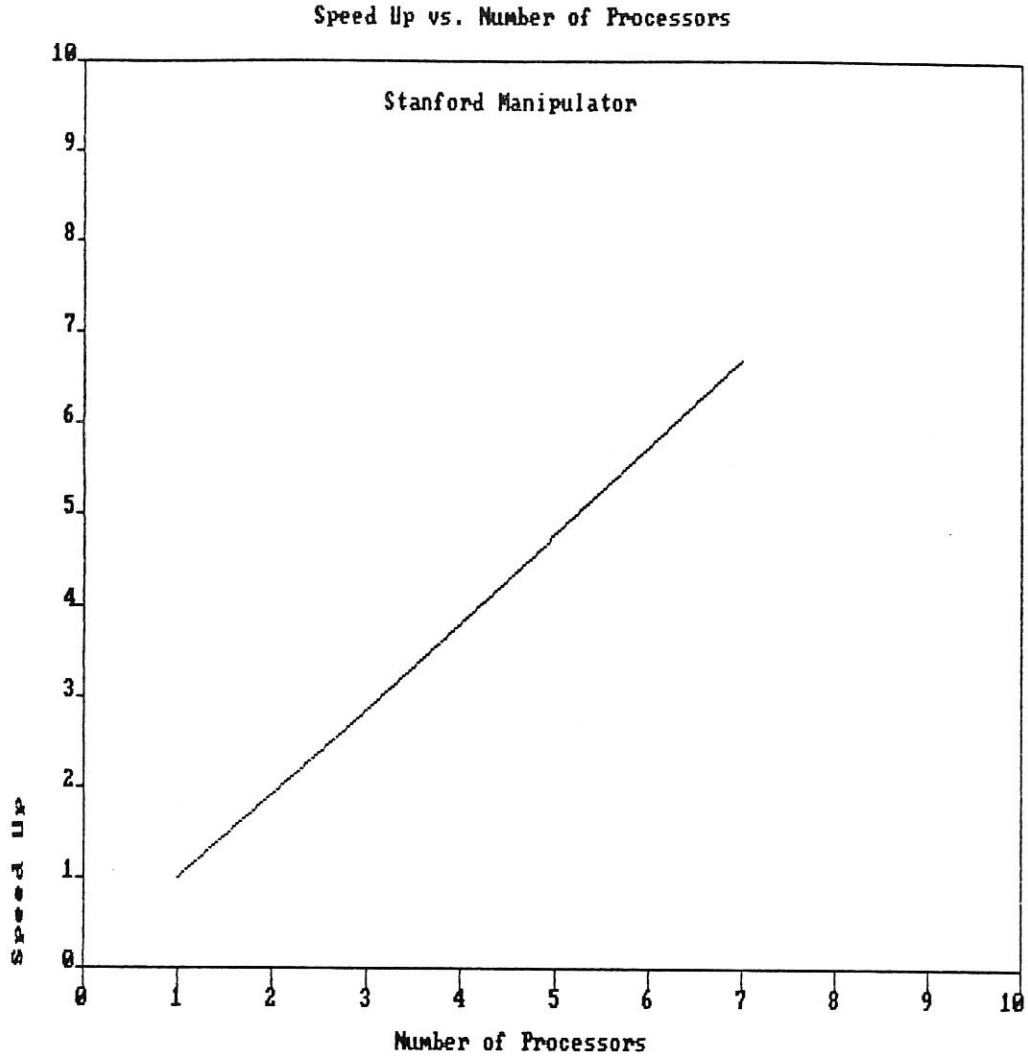


Fig.9 Speed Up in Computations