A Parallel Newton-Euler Formulation for Fast Dynamic Simulation of Robot Manipulators

by

A. Y. ZOMAYA

A. S. MORRIS

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

Research Report No. 368

July 1989

Abstract

Advanced control strategies require the inclusion of the dynamical model of the robot arm in the control law. However, the dynamics consist of a highly coupled and non-linear set of equations. Thus, this complexity has always presented a major obstacle in real-time dynamic control applications. The computationally efficient solution of this problem will lead to a better comprehension of the key factors affecting robot operations.

This work describes a solution of this problem by employing a parallel processing approach. The dynamics are computed by using a semi-customised Newton-Euler formulation. The algorithm is distributed over a *highly-coupled* multiple-instruction multiple-data steram (**MIMD**) computer architecture. The computer system is constructed from general purpose (**VLSI**) building blocks called the (**TRANSPUTER**). The cost-effectivness and speed of the scheme is demonstrated by a case study (PUMA 560 robot arm). The communication issues between the different processors are discussed. Speed-up results are included to show the superiority and advantages of the parallel approach.

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

1. Introduction

Present day robot manipulators are generally implemented by simple and well defined (PID) controllers. However, to allow these robots to operate under varying conditions, advanced control algorithms are needed to counteract for different changes and allow for wider diversity. This necessitates the inclusion of the system (robot) dynamics in the controller design [46].

Nevertheless, the computation of the dynamics is a very intensive task. In addition, it must be computed within a sampling rate of 60 Hz or more to avoid poor performance. This emphasise the fact that the efficient and inexpensive computation of the dynamics enhances the feasibility of real-time robot controllers.

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] proposed to neglect 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 [34].

The dynamics provide an important tool for simulation and feedforward computations. Therefore, it can be used in testing and designing controllers without the expense and hazards accompanied with actual systems. The formulation of computationally efficient dynamic models has been an active area of research for the last two decades and several methods have been developed. The Lagrange-Euler (LE) [3,41,45] has high computational complexity but is a very well structured and systematic representation. The Recursive Lagrangian [15] gives good computational results but destroyes the structure of the equations. The Newton-Euler (NE) [1,34,39,48] has the most efficient computational formulation but with untidy recursive equations. Other approaches include the tabulation techniques [43], Kane's dynamic equations [23], the Generalized D'Alembert [30]. and the use of dedicated microprocessor architectures [22,38]. The most commonly used of these methods are the (LE) and (NE) which were shown to be equivalent [44]. In this paper the (NE) is employed.

2. Robot Mechanism

An open chain robot mechanism consists of a chain of $(N+1)$ rigid links. The links (Fig.1) are arranged such that link (i) is connected to a preceding link $(i-1)$ and a following link $(i+1)$. In robot manipulators, two types of joints exist, translational and revolute joints. The translational joints are such that the adjacent links translate linearly to each other along the joint axis, while the revolute joints allow adjacent links to rotate with respect to each other about the joint axis. Therefore, the link (i) motion with reference to the link $(i-1)$ depends only on one variable, rotation Θ_i or translation d_i. Generally, the robot base is considered to be link (0). The last link (N) carries a gripper (hand) or a tool (drill, pincer) and is called the end effector of the robot. The location of an object in space is determined by six degrees of freedom (dof), three of which represent position and the other three orientation. If a task is performed in space without constraints, 6 dof are necessary. But if the task is performed in a plane, only 3 dof are necessary. Usually, a typical robot manipulator arm consists of 6 dof.

2.1. Kinematic Description

The widely used conventions of Denavit and Hartenberg (DH) [10] are adopted in this work. The main idea is to assign a coordinate frame to each link with the z-axis along the joint axis. This gives rise to four transformations; the rotation angle (Θ_i) which rotates about the z_{i-1} in the counter-clockwise direction, d_i distance between x_{i-1} and x_i axes along the z_{i-1} (link length), a_i the shortest distance between z_{i-1} and z_i (offset distance), and rotation angle (α_i) about the x_i (twist angle). The link parameters of a PUMA 560‡ robot arm are given (table 1).

LINK PARAMETERS				
Link	Θ	a	d	α
1	Θ_1	0	0	−90
2	Θ_2	a_2	d_2	0
3	Θ_3	a_3	0	90
4	Θ_4	0	d_4	−90
5	Θ_5	0	0	90
6	Θ_6	0	d_6	0

Table 1. Link Parameters between the different frames of a PUMA 560-Like Manipulator

From these parameters an orthogonal rotation matrix can be formed which transforms a vector in the $(i-1)^{th}$ coordinate frame to a coordinate frame which is parallel to the $(i)^{th}$ coordinate frame:

$$\mathbf{A}_i = \begin{bmatrix} \cos\Theta_i & -\sin\Theta_i \cos\alpha_i & \sin\Theta_i \sin\alpha_i \\ \sin\Theta_i & \cos\Theta_i \cos\alpha_i & -\cos\Theta_i \sin\alpha_i \\ 0 & \sin\alpha_i & \cos\alpha_i \end{bmatrix} \tag{1}$$

and the position of the $(i)^{th}$ coordinate frame with respect to the $(i-1)^{th}$ coordinate frame is given by:

$$\mathbf{p}_i = \begin{bmatrix} a_i \\ d_i \sin\alpha_i \\ d_i \cos\alpha_i \end{bmatrix} \tag{2}$$

For a revolute joint, Θ_i changes while d_i, a_i, and α_i remain constant. For a translational joint d_i is changing while a_i, Θ_i, and α_i are constants. To achieve transformation between different coordinate frames a matrix $\mathbf{T}_n$ is defined such that

$$\mathbf{T}_n = \mathbf{A}_1\mathbf{A}_2\mathbf{A}_3\mathbf{A}_4\mathbf{A}_5\mathbf{A}_6 \tag{3}$$

‡ PUMA 560 is a trademark of Unimation, Inc.

The previous equations describe the kinematic behaviour of the robot arm.

3. Previous Work

Several parallel architectures have been proposed by researchers to solve for the inverse dynamics problem. The innovative work of Luh and Lin [35] , 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 inbalances because some of the processors are under-utilized and the interprocessor communication and synchronization of the (NE) are ignored.

Orin et. al. [40] 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 [29] proposed two parallel algorithms using special purpose processors. First, is a linear parallel algorithm which is related to the Luh and Lin method. The second is a logarithmic-parallel-algorithm based on the partial sum technique. Kasahara and Narita [24] 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 [31] introduced a method based on the recursive doubling algorithm with a modified inverse perfect shuffle interconnection scheme between a set of parallel processors. Vukobratovic et. al. [47] 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 practical applications. Recently, Hashimoto and Kimura [13] presented a scheme based on the so-called resolved (NE) algorithm. Their approach is suitable for VLSI implementation. Finally, some more work in this area can be found in the literature [2, 6, 8, 33, 42, 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 *Multiple processors 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.

4. The Inverse Dynamics

The Inverse Dynamics problem can be stated as follows: Given the input vectors of joint positions $\Theta(t_0)$, joint velocities $\dot{\Theta}(t_0)$, and joint accelerations $\ddot{\Theta}(t_0)$ at any instant of time (t_0), calculate the applied force/torque $\tau(t_0)$. The dynamic equations of motion of a manipulator can written as:

$$\tau(t) = \mathbf{D}(\Theta)\,\ddot{\Theta}\,(t) + \mathbf{C}(\Theta,\dot{\Theta}) + \mathbf{h}(\Theta) \tag{4}$$

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; $\mathbf{D}(\Theta)$ is an $n \times n$ effective and coupling inertia matrix; $\mathbf{C}(\Theta,\dot{\Theta})$ is an $n \times 1$ Coriolis and Centripetal effects vector; and $\mathbf{h}(\Theta)$ is an $n \times 1$ gravitational force vector, where (n) is the no. of degrees of freedom (dof).

The computation of eq.(4) is computationlly expensive and has until recently posed a major bottleneck in real-time control of robot manipulators. As mentioned earlier, much effort has been allocated to producing more time-efficient techniques [3,4,9,11,32,37,51].

Based on the (NE) [34] the dynamic algorithm for a revolute arm (PUMA 560) divides into two recursive phases:

Initialisation:

$$\mathbf{z}_0 = \begin{bmatrix} 0 & 0 & 1 \end{bmatrix}^T$$

$$\omega_0 = \dot{\omega}_0 = 0$$

$$\dot{\mathbf{v}}_0 = g\mathbf{z}_0 \qquad g = 9.8062\ m/s^2$$

$$\mathbf{f}_{n+1} = \textit{force exerted at the end-effector}$$

$$\mathbf{n}_{n+1} = \textit{moment exerted at the end-effector}$$

Phase 1 (Forward iteration):

```
For i = 0,...,n-1 do;
        $\boldsymbol{\omega}_{i+1} = \mathbf{A}_{i+1}^{T} [\, \boldsymbol{\omega}_i + \mathbf{z}_0 \dot{\boldsymbol{\Theta}}_{i+1} \,]$    (5)
        $\dot{\boldsymbol{\omega}}_{i+1} = \mathbf{A}_{i+1}^{T} [\, \dot{\boldsymbol{\omega}}_i + \mathbf{z}_0 \ddot{\boldsymbol{\Theta}}_{i+1} + \boldsymbol{\omega}_i \times (\mathbf{z}_0 \dot{\boldsymbol{\Theta}}_{i+1}) \,]$    (6)
        $\dot{\mathbf{v}}_{i+1} = \mathbf{A}_{i+1}^{T} [\, \dot{\mathbf{v}}_i \,] + \dot{\boldsymbol{\omega}}_{i+1} \times \mathbf{p}_{i+1} + \boldsymbol{\omega}_{i+1} \times (\boldsymbol{\omega}_{i+1} \times \mathbf{p}_{i+1})$    (7)
        $\boldsymbol{\psi}_{i+1} = \dot{\boldsymbol{\omega}}_{i+1} \times \mathbf{s}_{i+1} + \boldsymbol{\omega}_{i+1} \times (\boldsymbol{\omega}_{i+1} \times \mathbf{s}_{i+1}) + \dot{\mathbf{v}}_{i+1}$    (8)
        $\mathbf{F}_{i+1} = m_{i+1} \boldsymbol{\psi}_{i+1}$    (9)
        $\mathbf{N}_{i+1} = \mathbf{J}_{i+1} \dot{\boldsymbol{\omega}}_{i+1} + \boldsymbol{\omega}_{i+1} \times (\mathbf{J}_{i+1} \boldsymbol{\omega}_{i+1})$    (10)
End (Phase_1);
```

<u>Phase 2 (Backward iteration):</u>

```
For i = n,...,1 do;
        $\mathbf{f}_i = \mathbf{A}_{i+1} [\, \mathbf{f}_{i+1} \,] + \mathbf{F}_i$    (11)
        $\mathbf{n}_i = \mathbf{A}_{i+1} [\, \mathbf{n}_{i+1} \,] + \mathbf{p}_i \times \mathbf{f}_i + \mathbf{N}_i + \mathbf{s}_i \times \mathbf{F}_i$    (12)
        $\tau_i(t) = \mathbf{n}_i^{T} (\mathbf{A}_i^{T} \mathbf{z}_0)$    (13)
End (Phase_2);
```

where

$\boldsymbol{\omega}_i$ and $\dot{\boldsymbol{\omega}}_i$	*Angular velocity and acceleration of the i^{th} coordinate frame*
$\dot{\mathbf{v}}_i$	*Linear acceleration of the i^{th} coordinate frame.*
$\boldsymbol{\psi}_i$	*Linear acceleration of the centre of mass of link (i).*
$\mathbf{F}_i$ and $\mathbf{N}_i$	*Net force and moment exerted on link (i).*
$\mathbf{f}_i$ and $\mathbf{n}_i$	*Force and moment exerted on link (i) by link (i-1).*
$\mathbf{s}_i$	*Position of the centre of mass of link (i).*
$\mathbf{J}_i$	*Inertia tensor matrix of link (i) about its centre of mass.*
m_i	*Mass of link (i).*

In the previous equations, all the matrices and vectors are of size (3×3) and (3×1) respectively. The computational requirements for the previous general purpose (NE) algorithm are listed in (table 2). Note that the sparsity of ω_0, $\dot{\omega}_0$, and $\dot{v}_0$ is not incorporated in simplifying the computations for the sake of generality. Also, to find the computational cost of the worst case situation.

TOTAL COMPUTATIONS (NE)		
Equation	*Multiplications*	*Additions Subtractions*
$\mathbf{A}_i$ *Matrices*	$4N$	0
$\mathbf{p}_i$ *Vectors*	$2N$	0
$\boldsymbol{\omega}_{i+1}$	$12N$	$9N$
$\dot{\boldsymbol{\omega}}_{i+1}$	$18N$	$15N$
$\dot{\mathbf{v}}_{i+1}$	$27N$	$21N$
$\boldsymbol{\psi}_{i+1}$	$18N$	$15N$
$\mathbf{F}_{i+1}$	$3N$	0
$\mathbf{N}_{i+1}$	$24N$	$18N$
$\mathbf{f}_i$	$9N$	$9N$
$\mathbf{n}_i$	$21N$	$21N$
τ_i	$12N$	$8N$
TOTAL	$150N$	$116N$
$N = 6$ *DOF*	900	696

Table 2. N-E Computational Requirements for a Revolute Manipulator

The previous table shows the computational requirements for a general revolute robot. It takes *900 multiplications* and *696 additions* to compute the inverse dynamics algorithm based on the (NE) and without any simplifying assumptions.

Less computationally-expensive algorithms have been produced by employing *Customisation and Symbolic Modelling Techniques* [26, 36, 37]. In these techniques more computational savings were realised by exploiting the sparsity of some of the matrices and vectors. Also, some of the approaches included the assumption of a certain robot type. As a consequence, further simplifications were made such as removing the multiplication by zero/one operations and some other redundant computations.

5. General Simplifications

In this section the (NE) algorithm will be simplified further. The analysis will concentrate on applying general simplifying assumptions. This will lead to less complications and reduce the amount of tedious work needed to modify the algorithm if another robot is assumed.

In this case the simplifed (NE) algorithm utilises the sparse nature of the ($\mathbf{z}_0$) vector and the diagonality of ($\mathbf{J}_i$) matrix (if applicable). The procedure will be divided into two parts.

5.1. Off-Line Analysis

For a PUMA-560 arm the rotation matrices can be shown to be:

$$\mathbf{A}_1 = \begin{bmatrix} \cos\Theta_i & 0 & -\sin\Theta_i \\ \sin\Theta_i & 0 & \cos\Theta_i \\ 0 & -1 & 0 \end{bmatrix}, \quad \mathbf{A}_2 = \begin{bmatrix} \cos\Theta_i & -\sin\Theta_i & 0 \\ \sin\Theta_i & \cos\Theta_i & 0 \\ 0 & 0 & 1 \end{bmatrix}, \quad \mathbf{A}_3 = \begin{bmatrix} \cos\Theta_i & 0 & \sin\Theta_i \\ \sin\Theta_i & 0 & -\cos\Theta_i \\ 0 & 1 & 0 \end{bmatrix}$$

$$\mathbf{A}_4 = \begin{bmatrix} \cos\Theta_i & 0 & -\sin\Theta_i \\ \sin\Theta_i & 0 & \cos\Theta_i \\ 0 & -1 & 0 \end{bmatrix}, \quad \mathbf{A}_5 = \begin{bmatrix} \cos\Theta_i & 0 & \sin\Theta_i \\ \sin\Theta_i & 0 & -\cos\Theta_i \\ 0 & 1 & 0 \end{bmatrix}, \quad \mathbf{A}_6 = \begin{bmatrix} \cos\Theta_i & -\sin\Theta_i & 0 \\ \sin\Theta_i & \cos\Theta_i & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

and the position vectors are:

$$\mathbf{p}_1 = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}, \mathbf{p}_2 = \begin{bmatrix} a_2 \\ 0 \\ d_2 \end{bmatrix}, \mathbf{p}_3 = \begin{bmatrix} a_3 \\ 0 \\ 0 \end{bmatrix}, \mathbf{p}_4 = \begin{bmatrix} 0 \\ -d_4 \\ 0 \end{bmatrix}, \mathbf{p}_5 = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}, \mathbf{p}_6 = \begin{bmatrix} 0 \\ 0 \\ d_6 \end{bmatrix}$$

However, the only general simplifying element which will be used during the on-line stage is the zero value of the element at position (3,1) of the $\mathbf{A}_i$*'s* matrices. In addition, the sparsity of ($\mathbf{z}_0$) will provide further simplifications:

(a) In eq.(5,6)

$$\mathbf{z}_0\, \dot{\Theta}_{i+1} = \begin{bmatrix} 0 & 0 & \dot{\Theta}_{i+1} \end{bmatrix}^T$$

$$\mathbf{z}_0\, \ddot{\Theta}_{i+1} = \begin{bmatrix} 0 & 0 & \ddot{\Theta}_{i+1} \end{bmatrix}^T$$

$$\boldsymbol{\omega}_i \times (\, \mathbf{z}_0\, \dot{\Theta}_{i+1}\,) = \begin{bmatrix} \omega_2\, \dot{\Theta}_{i+1} & -\omega_1\, \dot{\Theta}_{i+1} & 0 \end{bmatrix}^T$$

(b) In eq.(10) if $\mathbf{J}_i$ is diagonal then:

$$\mathbf{J}_i\, \dot{\boldsymbol{\omega}}_{i+1} = \begin{bmatrix} J_{11} & 0 & 0 \\ 0 & J_{22} & 0 \\ 0 & 0 & J_{33} \end{bmatrix} \bullet \begin{bmatrix} \omega_1 \\ \omega_2 \\ \omega_3 \end{bmatrix}$$

could be substituted by

$$\mathbf{J}_i\, \dot{\boldsymbol{\omega}}_{i+1} = \begin{bmatrix} J_{11} \\ J_{22} \\ J_{33} \end{bmatrix} \bullet \begin{bmatrix} \omega_1 \\ \omega_2 \\ \omega_3 \end{bmatrix} \quad \textit{(simple vector dot product operation)}$$

the same applies to the ($\mathbf{J}_i\ \omega_{i+1}$) case.

(c) In eq.(13):

$$\mathbf{A}_i^T\ \mathbf{z}_0 = \begin{bmatrix}0 & \sin(\alpha_i) & \cos(\alpha_i)\end{bmatrix}^T$$

which will lead to:

$$\mathbf{A}_1^T\ \mathbf{z}_0 = \begin{bmatrix}0\\-1\\0\end{bmatrix}\ ,\ \mathbf{A}_2^T\ \mathbf{z}_0 = \begin{bmatrix}0\\0\\1\end{bmatrix}\ ,\ \mathbf{A}_3^T\ \mathbf{z}_0 = \begin{bmatrix}0\\1\\0\end{bmatrix}\ ,\ \mathbf{A}_4^T\ \mathbf{z}_0 = \begin{bmatrix}0\\-1\\0\end{bmatrix}\ ,\ \mathbf{A}_5^T\ \mathbf{z}_0 = \begin{bmatrix}0\\1\\0\end{bmatrix}\ ,\ \mathbf{A}_6^T\ \mathbf{z}_0 = \begin{bmatrix}0\\0\\1\end{bmatrix}$$

Accordinglly,

$$\tau_i(t) = \mathbf{n}_i^T\ (\ \mathbf{A}_i^T\ \mathbf{z}_0\ ) = (\ n_2^i \sin(\alpha_i) + n_3^i \cos(\alpha_i)\ )$$

this will yield:

$$\tau_1(t) = (\ -n_2^1\ )\ ,\ \tau_2(t) = (\ n_3^2\ )\ ,\ \tau_3(t) = (\ n_2^3\ )$$

$$\tau_4(t) = (\ -n_2^4\ )\ ,\ \tau_5(t) = (\ n_2^5\ )\ ,\ \tau_6(t) = (\ n_3^6\ )$$

The computational load of the previous operations will be avoided in the on-line implementation. It is important to note that if the algorithm is used for another robot arm, the change to be made is minor and can be done manually.

5.2. <u>On-Line Procedure</u>

The cost of the dynamics computation is reduced as shown in table (3).

TOTAL COMPUTATIONS Semi-Customised (NE)		
Equation	*Multiplications*	*Additions Subtractions*
$\boldsymbol{\omega}_{i+1}$	$8N$	$6N$
$\dot{\boldsymbol{\omega}}_{i+1}$	$10N$	$8N$
$\dot{\mathbf{v}}_{i+1}$	$26N$	$20N$
$\boldsymbol{\psi}_{i+1}$	$18N$	$15N$
$\mathbf{F}_{i+1}$	$3N$	0
$\mathbf{N}_{i+1}$	$24N$	$18N$
$\mathbf{N}^{*}_{i+1}$	$12N$	$6N$
$\mathbf{f}_i$	$8N$	$8N$
$\mathbf{n}_i$	$20N$	$20N$
TOTAL	$117N$	$95N$
*TOTAL**	$105N$	$83N$
$N = 6$ *DOF*	702	570
$N = 6$ *DOF**	630	498

* *In case* $\mathbf{J}_i$ *is diagonal*

Table 3. Semi-Customised (N-E) Computational Requirements for a Revolute Manipulator

It can be noticed from the previous table that the amount of reduction in the total floating-point operations for the (NE) is 20% or 29% if ($\mathbf{J}_i$) is dense or diagonal respectively. In this case a trade off was made between the very specific fully customised and the general numerical solutions. This is because part of the (NE) algorithm was customised and expressed symbolically where the sparsity of some of the vectors and matrices was exploited, while the other part remained numerical and generality was maintained. So it can be called the *Semi-Customised Symbolic (NE) Algorithm (SCSNEA)*.

6. 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 practical 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, 16, 28, 49].

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.*

6.1. 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 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 [18, 19, 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 [12, 20, 21].

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

6.2. Processor Farms

Processor farms are based on a simple concept which might be useful in a wide range of applications [17]. 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.

7. Parallel Implementation of the Dynamics

Hollerbach [7] states that "*Some improvements could arise by taking advantage of particular kinematic structures of manipulators or by parallel computation*". In section (5) a simplified version of the (NE) produced some computational savings. In this section the algorithm will be distributed over a parallel processor system in order to speed up the computation. Therefore, it will be called the *Parallel Semi-Customised Symbolic (NE) Algorithm (PSCSNEA)*. Real-time results are included to show the superiority of multiprocessing architectures.

7.1. The Algorithm

For this method a tree structured network is used (Fig.3) where (P_0) is the master processor (controller) and the other three processors P_1, P_2, and P_3 are slave processors (the names processor and transputer are used interchangably).

The master processor is connected to a personal computer (PC) which works as a link between the user and the network. P_0 sends the position variables, velocities and accelerations $(\Theta_i, \dot{\Theta}_i, \ddot{\Theta}_i)$ and receives the columns of $\tau_i(t)$ from the slave processors in the network. The main role of P_0 is to supervise the network and to check for any faulty event. The job of computing the dynamics is divided as shown:

- P_1 : compute eq.(5,10) and eq.(12) in the first and second phases respectively.
- P_2 : compute eq.(6,8) and eq.(13) in the first and second phases respectively.
- P_3 : compute eq.(7,9) and eq.(11) in the first and second phases respectively.

The whole procedure will be divided into four modules each running on a processor.

7.1.1. Module (0):

P_0 sends Θ_i,$\dot{\Theta}_i$ and $\ddot{\Theta}_i$ to P_1, P_2, and P_3. This is performed in parallel. Then, P_0 will start receiving $\tau_i(t)$ values from processor P_2. As shown in OCCAM code:

```
SEQ
  PAR
    ... Send $\Theta_i$, $\dot{\Theta}_i$, and $\ddot{\Theta}_i$ where $i = 1,...,n$ to $P_1$
    ... Send $\Theta_i$, $\dot{\Theta}_i$, and $\ddot{\Theta}_i$ where $i = 1,...,n$ to $P_2$
    ... Send $\Theta_i$, $\dot{\Theta}_i$, and $\ddot{\Theta}_i$ where $i = 1,...,n$ to $P_3$
  PAR
    ... Receive $\tau_i(t)$ where $i = 1,...,n$ from $P_2$
```

The following three modules are working in parallel together, but each one of them is running sequentially. During the operation the three modules communicate with one another.

7.1.2. Module (1):

This module computes eq.(5,10) and eq.(12). It is divided into two phases.

Initial Stage:

```
SEQ
    ... Receive $\Theta_i$, $\dot{\Theta}_i$, and $\ddot{\Theta}_i$ from $P_0$
    ... Form $A_1, A_2, \cdots, A_6$
    ... Initialise $\omega_0$
```

Phase 1:

```
SEQ
    SEQ i = 0 FOR 6
        SEQ
            PAR
                ... Send $\omega_i$ to $P_2$
                ... Send $\omega_i$ to $P_3$
                ... Receive $\dot{\omega}_i$ from $P_2$
            ... Compute $Temp\_Store = [\ \omega_i + z_0\ \dot{\Theta}_{i+1}\ ]$
            ... Compute $(A_{i+1}^T * Temp\_Store) \Rightarrow$ Eq.(5)
            ... Compute $(J_{i+1}\ \omega_{i+1})$ and $(J_{i+1}\ \dot{\omega}_{i+1})$
            ... Compute $(N_{i+1}) \Rightarrow$ Eq.(10)
```

Phase 2:

```
SEQ
    ... Initialise $(n_{i+1})$
    WHILE ( i >= 1 )
        SEQ
            ... Receive $F_i$ from $P_3$
            ... Compute $Temp\_Store = (A_{i+1}\ n_{i+1})$
            ... Compute $n_i = Temp\_Store + N_i$
            ... Compute $s_i \times F_i$
            ... Receive $f_i$ from $P_3$
            ... Compute $p_i \times f_i$
            ... Compute $n_i \Rightarrow$ Eq.(12)
            ... Send $n_i$ to $P_2$
```

7.1.3. Module (2):

This module computes eq.(6,8) and eq.(13). It is divided into two phases.

```
Initial Stage:
SEQ
    ... Receive $\Theta_i$, $\dot{\Theta}_i$, and $\ddot{\Theta}_i$ from $P_0$
    ... Form $A_1, A_2, \cdots, A_6$
    ... Initialise $\dot{\omega}_0$
Phase 1:
SEQ
    SEQ i = 0 FOR 6
        SEQ
            PAR
                ... Send $\dot{\omega}_i$ to $P_1$
                ... Send $\dot{\omega}_i$ to $P_3$
                ... Receive $\omega_i$ from $P_1$
            ... Compute $Temp\_Store = [\ \dot{\omega}_i + z_0\ \ddot{\Theta}_{i+1} + \omega_i \times (\ z_0\ \dot{\Theta}_{i+1}\ )\ ]$
            ... Compute $(A^T_{i+1} * Temp\_Store) \Rightarrow$ Eq.(6)
            ... Compute $(\dot{\omega}_{i+1} \times s_{i+1})$ and $(\omega_{i+1} \times (\ \omega_{i+1} \times s_{i+1}\ ))$
            ... Receive $(\dot{v}_{i+1})$ from $P_3$
            ... Compute $(\psi_{i+1}) \Rightarrow$ Eq.(8)
            ... Send $(\psi_{i+1})$ to $P_3$
Phase 2:
SEQ
    WHILE ( i >= 1 )
        SEQ
            ... Receive $n_i$ from $P_1$
            ... Compute $\tau_i(t) \Rightarrow$ Eq.(13)
            ... Send $\tau_i$ to the Master Processor $P_0$
```

7.1.4. Module (3):

This module computes eq.(7,9) and eq.(11). It is divided into two phases.

```
Initial Stage:
 SEQ
     ... Receive $\Theta_i$, $\dot{\Theta}_i$, and $\ddot{\Theta}_i$ from $P_0$
     ... Form $A_1, A_2, \cdots, A_6$
     ... Initialise $\dot{v}_0$
Phase 1:
 SEQ
     SEQ i = 0 FOR 6
        SEQ
            PAR
                ... Receive $\omega_i$ from $P_1$
                ... Receive $\dot{\omega}_i$ from $P_2$
            ... Compute $A^T_{i+1}\ \dot{v}_i$
            ... Compute $(\dot{\omega}_{i+1} \times p_{i+1})$ and $(\omega_{i+1} \times (\ \omega_{i+1} \times p_{i+1}\ ))$
            ... Compute $(\dot{v}_{i+1}) \Rightarrow$ Eq.(7)
            ... Send $(\dot{v}_{i+1})$ to $P_2$
            ... Receive $(\psi_{i+1})$ from $P_2$
            ... Form $(F_{i+1}) \Rightarrow$ Eq.(9)
Phase 2:
 SEQ
     ... Initialise $(f_{i+1})$
     WHILE ( i >= 1 )
        SEQ
            ... Send $F_i$ to $P_1$
            ... Form $Temp\_Store = A_{i+1} f_{i+1}$
            ... Compute $f_i = Temp\_Store + F_i \Rightarrow$ Eq.(11)
            ... Send $f_i$ to $P_1$
```

This parallel approach requires $40N$ *multiplications* and $32N$ *additions*, that is a total of 432 floating point operations, which seems sufficient for real-time applications. However, any number of methods could be used to divide the problem. In our case the choice was made to solve the problem as fast as possible and to keep the flow of information (communication) between the processors smooth.

8. Results

For the case of a PUMA 560 robot arm, the general (NE) and the (SCSNEA) were executed using one Transputer only. Afterwards, the (PSCSNEA) was distributed over a network of 4-Transputers (Fig.3). The results of the total processing time required for each of them are shown in (Fig.4). A total processing time of (2 *msec*) was achieved for the (PSCSNEA). The following can be noticed from these results:

[a]. The amount of reduction in the total floating point operations for the (PSCSNEA) is 73% and 62% compared to (NE) and (SCSNEA) respectively.

[b]. The amount of reduction in terms of total processing time for the (PSCSNEA) is 58% and 41% compared to (NE) and (SCSNEA) respectively.

This difference can be referred to the overheads and communication time between the processors because of the recursive nature of the algorithm. Zomaya and Morris [53, 54] used the (LE) to solve for the dynamics, and the communication between slave processors was minimised by enabling each processor to execute its job without the need for data from other processors. However, this algorithm has the following two advantages:

(1). The physical implementation is simple. Being based on the (NE) which requires less hardware to compute the dynamics [27] , the algorithm needs only four Transputers.

(2). The modularity of the algorithm, that is, it needs little effort and time to be modified to suit another robot arm (approximately 10-15 minutes).

9. Conclusion

The Dynamic modelling and simulation of typical robot manipulators such as the PUMA 560 arm is systematic and simple in concept but complicated in respect of the computational burden inherent in real-time control applications. A simplified form of the dynamics based on the Newton-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.

A few notes have to be kept in mind while distributing a whole of a task (algorithm) to work on several processors:

* The data flow paths and the communication between the different processors constitute a major bottleneck in many situations.

* The division of the workload between the different co-processors.
* The idle-time that each processor spends waiting for input from other processors.
* The required computing power which decides the size and complexity of the network.
* The amount of parallelism and sequentialism inherent in the algorithm.
* The efficiency in coding the algorithm (software development).

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* and *Forward Dynamics* solutions [52, 54]. Real-time results have been produced to demonstrate how the recent breakthroughs in VLSI technology can be used together with parallel processing techniques to facilitate the dynamic 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]. BRADY, M., HOLLERBACH, J. M., JOHNSON, T. L., LOZANO-PEREZ, T., AND MASON, M. T., (1982). *Robot Motion: Planning and Control,* MIT Press,

Cambridge, Mass.

[8]. 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.

[9]. 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.

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

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

[13]. HASHIMOTO, K. AND KIMURA, H., (1989). "A New Parallel Algorithm for Inverse Dynamics," *International J. of Robotics Research*, vol. 8, no. 1.

[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]. 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.

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

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

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

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

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

[21]. 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.

[22]. KABUKA, M. AND ESCOTO, R., (Feb. 1989). "Real-Time Implementation of Motion on the NEC PD77239 DSP," *IEEE Micro*, vol. 9, no. 1.

[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]. KHOSLA, P. K. AND NEUMAN, C. P., (1985). "Computational Requirements of Customized Newton-Euler Algorithms," *J. Robotic Systems*, vol. 2, no. 3, pp. 309-327.

[27]. KHOSLA, P. K. AND RAMOS, S., (1988). "A Comparative Analysis of the Hardware Requirements for the Lagrange-Euler and the Newton-Euler Dynamics Formulations," in *IEEE International Conf. on Robotics and Automation, Philadelphia, PA*, vol. 1, pp. 291-296.

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

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

[30]. 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.

[31]. 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.

[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]. 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.

[34]. 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.

[35]. 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.

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

[37]. NEUMAN, C. P. AND MURRAY, J. J., (1987). ''Customized Computational Robot Dynamics,'' *J. Robotic Systems*, vol. 4, no. 4, pp. 503-526.

[38]. 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.

[39]. 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.

[40]. 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.

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

[42]. 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.

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

[44]. 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.

[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. 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.*

[53]. 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.*

[54]. ZOMAYA, A. Y. AND MORRIS, A. S., (1989). "Fast Forward Dynamics Algorithm for Robot Arms Using Multi-Processing," *Research Report No. 367, Dept. of Control Engineering, University of Sheffield, Sheffield S1 3JD, U.K.*

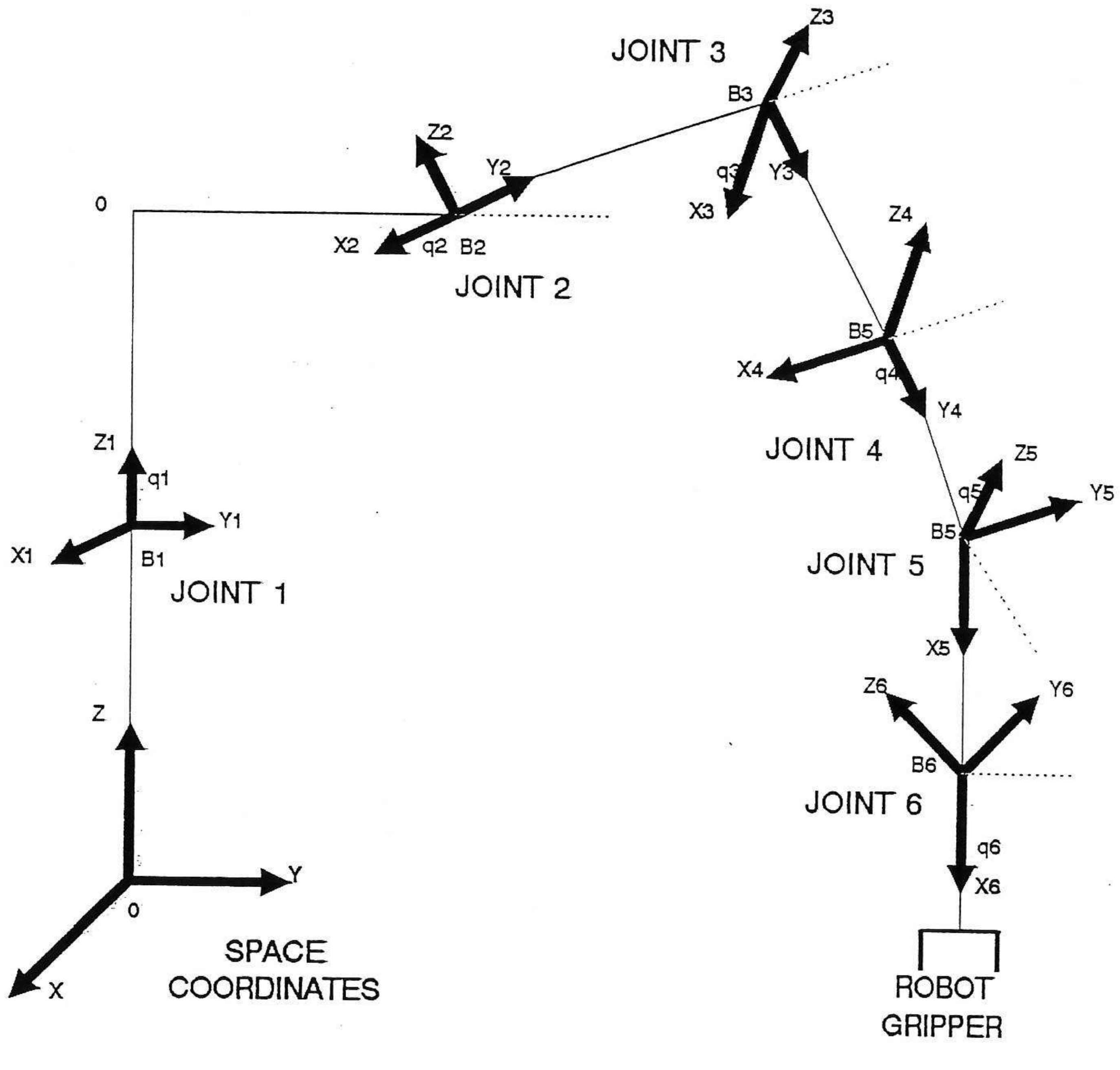

Figure 1. The Relation Between the Different Links of a Robot Arm.

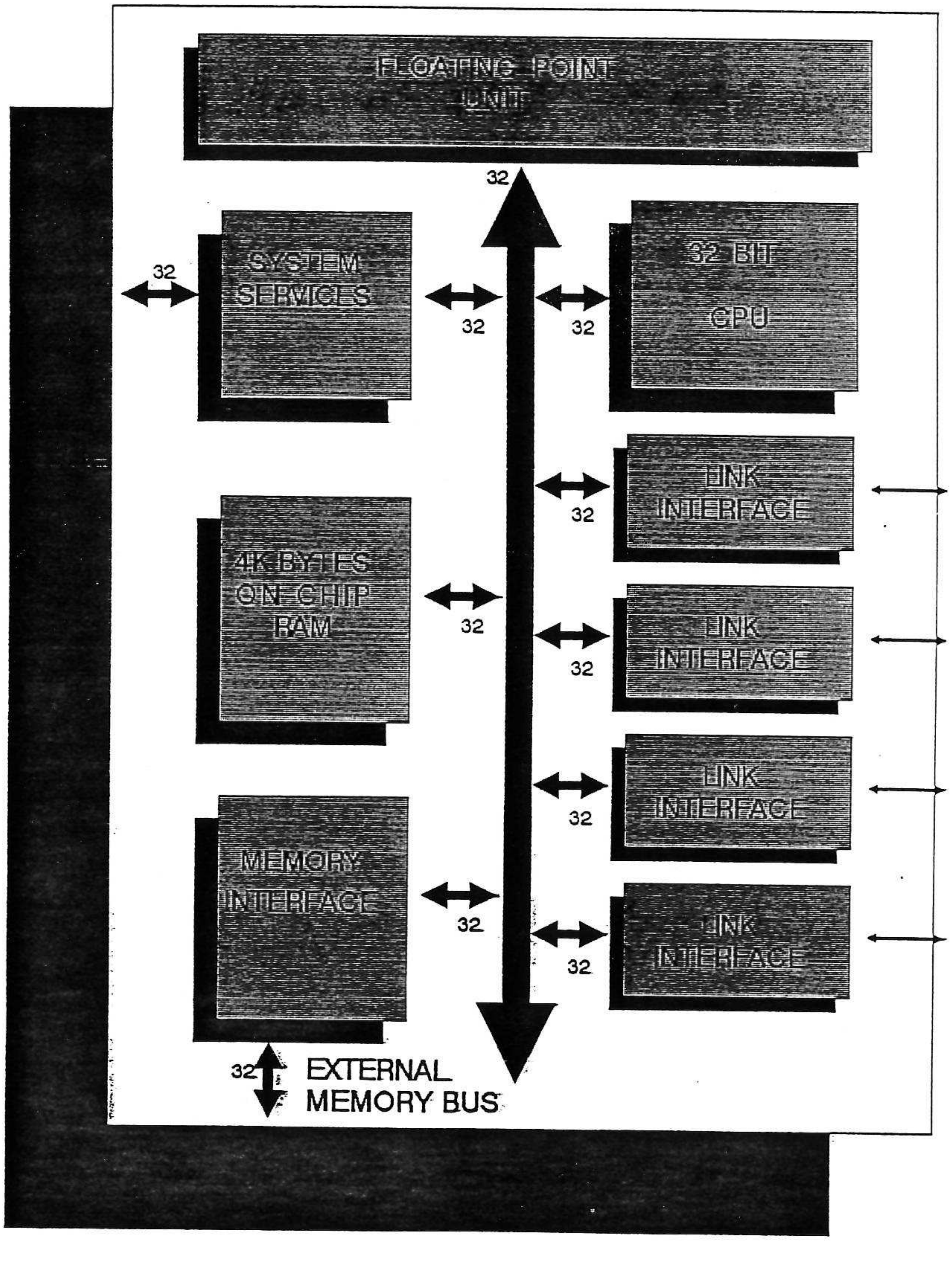

Figure 2. The well known INMOS T800 Transputer.

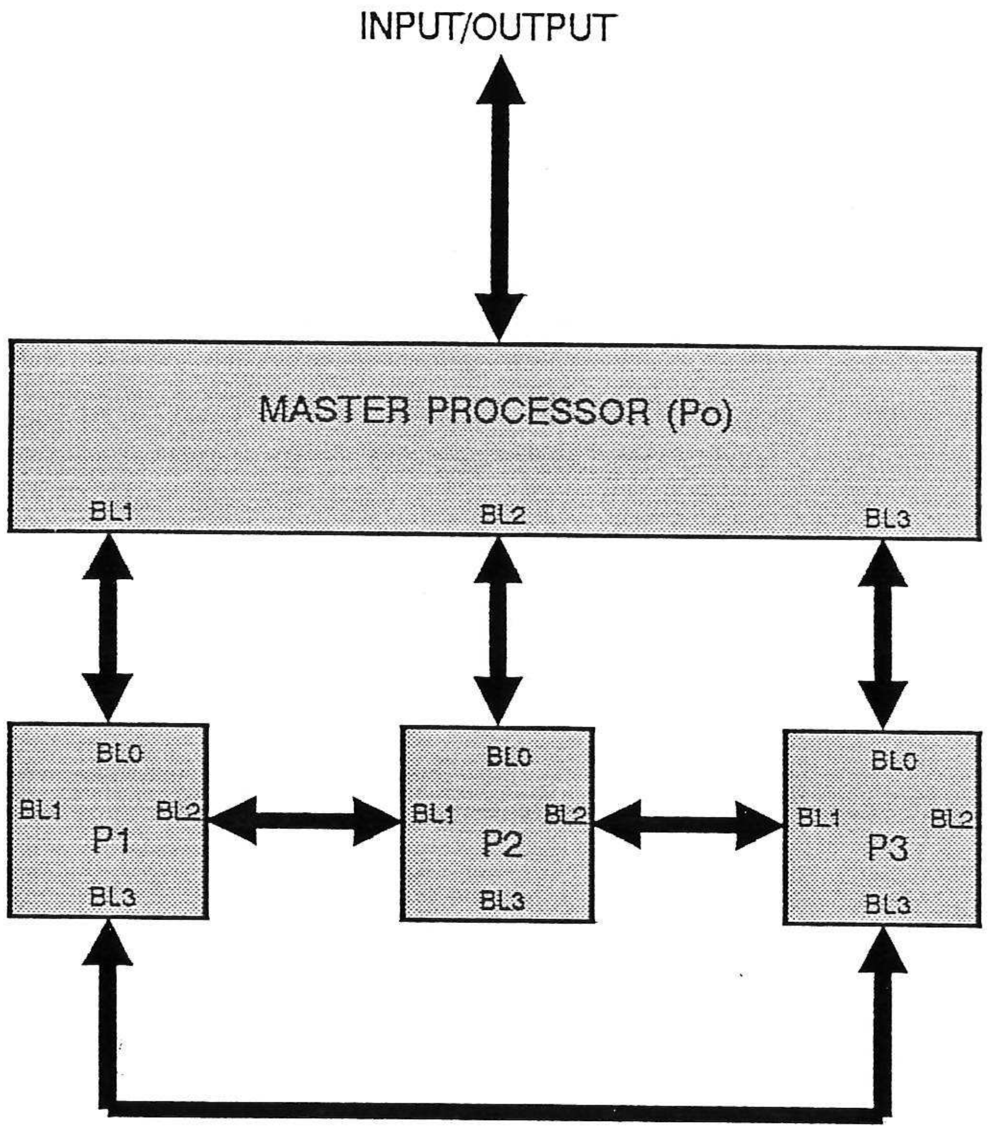

Figure 3. A Four-Processors Network.

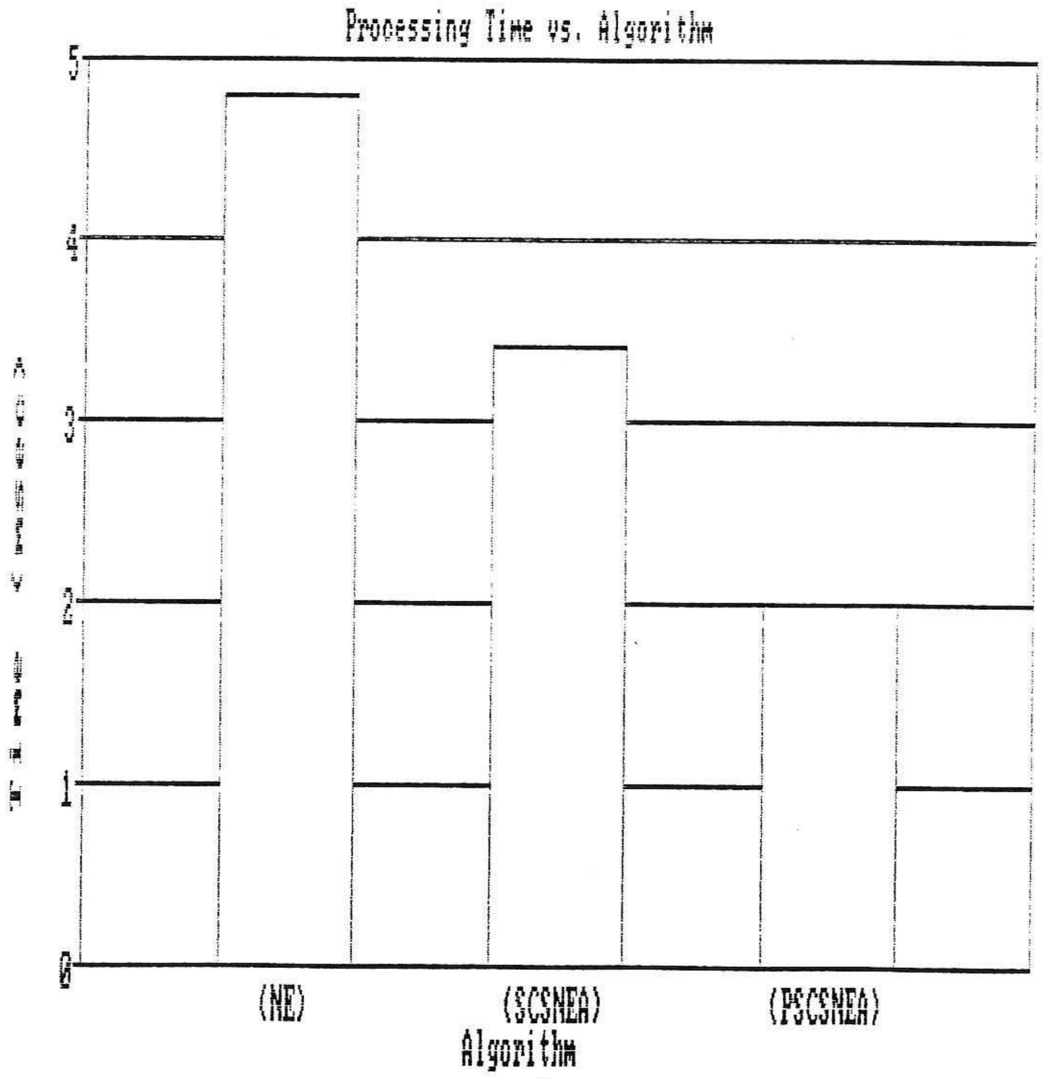

Figure 4. Processing time for the (NE), (SCSNEA), and (PSCSNEA) Implementations.