Distributed VLSI Architectures for Fast Jacobian and Inverse Jacobian Formulations

A. Y. ZOMAYA and A. S. MORRIS

Departement of Control Engineering
University of Sheffield
Mappin Street
Sheffield S1 3JD

Research Report No. 346

October 1988

Abstract

The rapid development in VLSI technology makes it possible to implement highly complicated and time consuming algorithms to suit real-time applications. Parallel processing techniques can now be used to reduce the computational time for models of a highly mathematical nature such as the kinematical description of robot manipulators. The development system used to implement the algorithms consists of an INMOS TRANSPUTER (a VLSI single chip computer) running the OCCAM concurrent programming language. This system is used to construct and evaluate the performance and cost effectiveness of several proposed methods to solve for the JACOBIAN and INVERSE JACOBIAN problems with special attention to the case of the robot operating in the neighbourhood of singular points. Detailed analysis is performed and successful results are obtained for a 6 dof robot arm (PUMA 560). Execution time comparisons between Von Neumann (uniprocessing) and parallel processing architectures are also included to show the superiority of the latter approaches.
Keywords : Robot manipulators, robotics, robot arms, VLSI architectures, distributed systems, occam, transputer, jacobian, inverse jacobian, parallel processing.

1 Introduction

The last few years have seen remarkable achievements in the field of robotics and related technology. The control of most existing robot manipulators is relatively simple and well defined, based on a servo mechanism at each joint. However, sophisticated control algorithms are needed to improve the speed and precision of active interaction of the robot with its environment. Hence, several schemes had been proposed by past researchers (Fu, Gonzales, and Lee 1987).

The position of the robot arm is most naturally expressed in joint coordinates whereas the usual point of interest is the position of the end-effector expressed in cartesian coordinates. Therefore, the transformation from joint to cartesian coordinates and visa versa is very important. This transformation is accomplished by using the *Jacobian* and *Inverse Jacobian* formulations which play a major role in many problems such as the control techniques (Whitney 1969, 1972, 1987; Luh, Walker, and Paul 1980; Wu and Paul 1982), the inverse kinematics (Tsai and Orin 1987; Angeles 1985; Ang and Torrassis 1987), and assisting in the general description of the kinematic behaviour and static forces equilbration of robot manipulators (Paul 1981; Craig 1986; Wolovich 1987). Different techniques have been proposed to solve for the jacobian and its inverse (Renaud 1981; Paul 1981; Warldon 1982; Featherstone 1983a, 1983b; Elgazzar 1984, 1985a, 1985b; Lenarcic 1984; Mitra and Mohalanabis 1984; Orin and Schrader 1984; Paul and Zhang 1986; Leahy, Nugent, Saridis, and Perreira 1987). However, only a few attempts have been made to incorporate parallelism to speed up the computations and achieve satisfactory real time standards and efficient performance (Orin, Chao, Olson, and Schrader 1985), unlike the area of robot dynamics which gained a lot of attention and good algorithms were developed (Luh and Lin 1982; Lathrop 1985; Lee and Chang 1986, 1988; Vukobratovic, Kircanski, and Li 1988). The purpose of this paper is to introduce new computational techniques for real time control implementations embodying Jacobian and Inverse Jacobian calculations within acceptable sampling rates of no less than 60 Hz.

The problem is solved for a 6 dof PUMA 560 robot arm. The results and discussions are presented in the following order; Section (2) presents the computer architecture used to implement this work. Section (3) highlights the problems of the Jacobian and Inverse Jacobian calculations from published literature. Parallelism is introduced to solve the problem in sections (5) and (6). Conclusions and further comments are given in section (7).

2 The TRANSPUTER and OCCAM

Recent years have witnessed rapid development in VLSI technology which is weighting the arguments in favour of parallel processing techniques (Kung 1982; Zakharov 1984; Hwang and Briggs 1985). This is achieved by distributing the task over a number of processors, ideally in such a way that all the processors used are fully utilised. To accomplish that, general purpose systems which employ parallel architectures have evolved to meet the increasing demand for more computing power and higher processing speed.

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 **T414** transputer in (Fig.1) which is used in this work is a 32 bit microcomputer with 2 Kbytes on chip RAM (50 ns static RAM) for high speed processing, a configurable memory interface, 4 bidirectional communication links, and a timer. It provides high performance arithematic and micro code support for floating point operations and 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 construct a parallel processing system, and permits massive concurrency to be used 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 by INMOS to run on the transputer (INMOS 1984, 1985, 1986; IEE Workshop 1987, 1988; Kerridge 1987), and is as important as the assembly language is for the ordinary microprocessor, because transputer features are best exploited by using Occam. 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 the simulation, modelling and control of complicated physical systems (Jones 1985; Hamblen 1987).

3 Jacobian and Inverse Jacobian

3.1 The Jacobian

The Jacobian (J) relates changes in joint space to changes in cartesian space. Hence, (J) is necessary in any cartesian based control scheme

$$\delta \mathbf{x} = \mathbf{J}(\theta)\delta\theta \tag{1}$$

where
$\mathbf{x}$ is the cartesian coordinates vector, and
θ is the position vector of joint angles
Orin and Schrader (1984) reviewed some of the methods used to compute (J) and some other techniques were proposed by (Lenaric 1984; Mitra and Mahalanabis 1984; Leahy, Nugent, Saridis, and Valavanis 1987). In this paper the adapted method is the one first outlined by Whitney (1972) and later refined by Paul (1981) because of its simple and algorithmic nature.

3.1.1 Nomenclature

The well known conventions first proposed by Denavit and Hartenberg (1955) are used throughout this paper. The main idea is to assign a co-ordinate 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}, translation of distance d_i along the z_{i-1} (offset distance), a_i the shortest distance between z_{i-1} and z_i (link length), and rotation angle (α_i) about the x_i (twist angle). From these parameters a 4×4 homogeneous transformation matrix is produced

$$\mathbf{A}_i = \begin{pmatrix} \cos\theta_i & -\sin\theta_i\cos\alpha_i & \sin\theta_i\sin\alpha_i & a_i\cos\theta_i \\ \sin\theta_i & \cos\theta_i\cos\alpha_i & -\cos\theta_i\sin\alpha_i & a_i\sin\theta_i \\ 0 & \sin\alpha_i & \cos\alpha_i & d_i \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{2}$$

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 and $a_i = 0$. To achieve transformation between different coordinate frames a matrix $\mathbf{T_n}$ is defined such that

$$\begin{aligned} \mathbf{T}_n &= \mathbf{A}_1\mathbf{A}_2\mathbf{A}_3\mathbf{A}_4\mathbf{A}_5\mathbf{A}_6 \\ &= \begin{pmatrix} \mathbf{R}_i^{i-1}(\theta, axis) & \mathbf{P}_i^{i-1} \\ 0^T & 1 \end{pmatrix} \end{aligned} \tag{3}$$

where

$\mathbf{R}_i^{i-1}$ is a 3×3 matrix that describes the orientation and rotation between successive coordinate frames.
$\mathbf{P}_i^{i-1}$ is a 3×1 vector which denotes pure translation .
Using these matrices the derivation of (J) for a 6 dof robot arm, as given by Paul (1981) , is as follows

$$\mathbf{T}_6^0 = \mathbf{A}_1\mathbf{A}_2\mathbf{A}_3\mathbf{A}_4\mathbf{A}_5\mathbf{A}_6 \tag{4}$$
$$\mathbf{T}_6^1 = \mathbf{A}_2\mathbf{A}_3\mathbf{A}_4\mathbf{A}_5\mathbf{A}_6 \tag{5}$$
$$\mathbf{T}_6^2 = \mathbf{A}_3\mathbf{A}_4\mathbf{A}_5\mathbf{A}_6 \tag{6}$$
$$\mathbf{T}_6^3 = \mathbf{A}_4\mathbf{A}_5\mathbf{A}_6 \tag{7}$$
$$\mathbf{T}_6^4 = \mathbf{A}_5\mathbf{A}_6 \tag{8}$$
$$\mathbf{T}_6^5 = \mathbf{A}_6 = \begin{pmatrix} \mathbf{n} & \mathbf{o} & \mathbf{a} & \mathbf{p} \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{9}$$

Accordingly, for a revolute joint (i), each column of (J) is of the form

$$\mathbf{J}_i = \begin{pmatrix} n_y^i p_x^i - n_x^i p_y^i \\ o_y^i p_x^i - o_x^i p_y^i \\ a_y^i p_x^i - a_x^i p_y^i \\ n_z^i \\ o_z^i \\ a_z^i \end{pmatrix} \tag{10}$$

For a translational joint (i) ;

$$\mathbf{J}_i = \begin{pmatrix} n_z^i \\ o_z^i \\ a_z^i \\ 0 \\ 0 \\ 0 \end{pmatrix} \tag{11}$$

3.2 The Inverse Jacobian

To determine the changes in joint variables ($\delta\theta$) to achieve a specified displacement (δx), it is necessary to invert the (J), i.e.

$$\delta\theta = \mathbf{J}^{-1}\delta\mathbf{x} \tag{12}$$

Melouah and Andre (1982) reduced the inverse computation of the (6×6) (J) to two (3×3) submatrices inversions. Although this technique easily identifies the singularities, it restricts the two submatrices to have an inverse which is not always valid in real time situations. Leahy, Nugent, Saridis, and Valavanis (1987) employed symbolic inversion. This is very difficult to solve owing to the complexity of (J) elements, unless this difficulty is minimised by taking into account the architecture of the manipulator and other simplifying factors, which is not an easy task that can be relied on.

For different degrees of freedom (N), eq.(12) is altered and the evaluation of $\mathbf{J}^{-1}$ requires the use of pseudo and generalised inverses (Ben-Israel and Greville 1974; Lawson and Hanson 1974), where

$$\delta\theta = (\mathbf{J}^T\mathbf{J})^{-1}\mathbf{J}^T\delta\mathbf{x}, N < 6 \tag{13}$$

$$\delta\theta = \mathbf{J}^T(\mathbf{J}^T\mathbf{J})^{-1}\delta\mathbf{x}, N > 6 \tag{14}$$

Tucker and Perreira (1987) reviewed this problem and proposed a technique to solve the problem in case of singularities but without considering real-time situations. In this work, the case of (N=6) is studied but no restriction is imposed on other cases.

4 Parallelism in the Jacobian Formulation

Parallel processing can be divided into four levels (Hwang and Briggs 1985)

- Job or program level.
- Task or procedure level.
- Interinstruction level.
- Intrainstruction level.

In this section the first and the second levels are used and addressed. The first level depends upon developing parallel processable algorithms where multiple programs are used to solve a large problem. The second level is achieved among procedures or tasks within the same program which involves the decomposition of a program (algorithm) into multiple tasks.

To show how parallel processing can be used to compute (J), three methods for dividing the task are described to achieve an optimal configuration. The main difference between the three methods is how the algorithm is divided, that is, the amount of work carried out by each processor and the overhead caused by the communication between the processors in the network. Real-time results are included to compare uniprocessing and multiprocessing architectures.

4.1 Method 1

For this method a tree structured network is used (Fig.2) were (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 (θ_i) and receives the columns of (J) 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 calculating the different columns of (J) is divided as shown

- P_1 : compute the fourth, fifth, and sixth columns of (J) using eq.(7-9) respectively.
- P_2 : compute the second and third columns of (J) using eq.(5,6) respectively.
- P_3 : compute the first column of (J) using eq.(4) .

The whole procedure will work as follows

1. P_0 sends θ_1, θ_2 to P_3, θ_3, θ_4 to P_2, and θ_5, θ_6 to P_1, and this is performed in parallel. Then, P_0 will start receiving (J) columns from the different processors, i.e.

```
SEQ
  PAR
      ...Send $\theta_1$ and $\theta_2$ to $P_3$
      ...Send $\theta_3$ and $\theta_4$ to $P_2$
      ...Send $\theta_5$ and $\theta_6$ to $P_1$
  PAR
      ...Receive $J_1$ from $P_3$
      ...Receive $J_2$ and $J_3$ from $P_2$
      ...Receive $J_4$, $J_5$, and $J_6$ from $P_1$
```

2. This stage is divided into 3 substages working in parallel together, but each substage is running sequentially (i.e instruction execution is sequential)

   (a) Processor P_1

```
SEQ
  ...Form $A_5$ and $A_6$
  ...Multiply $A_5$ by $A_6$ to form $T_6^4$
  ...Send $T_6^4$ to $P_2$ and $P_3$ and receive $A_4$ from $P_2$
  ...Multiply $A_4$ by $T_6^4$
  ...Form $J_4$ from $(A_4 * T_6^4)$
  ...Form $J_5$ from $T_6^4$
  ...Form $J_6$ from $A_6$
  ...Send $J_4$, $J_5$, and $J_6$ to $P_0$
```

   (b) Processor P_2

```
SEQ
  ...Form $A_3$ and $A_4$
  ...Multiply $A_3$ by $A_4$ and store in $T_1$
  ...Send $T_1$ and $A_4$ to $P_3$ and $P_1$ respectively,
     and receive $(A_5 * A_6)$ from $P_1$ and $A_2$ from $P_3$
  ...Multiply the matrices to form $T_6^2$ and $T_6^1$
  ...Form $J_2$ from $T_6^1$
  ...Form $J_3$ from $T_6^2$
  ...Send $J_2$, $J_3$ to $P_0$
```

   (c) Processor P_3

```
SEQ
  ...Form $A_1$ and $A_2$
  ...Multiply $A_1$ by $A_2$ and store in $T_1$
  ...Send $A_2$ to $P_2$ and receive $(A_3 * A_4)$
     and $(A_5 * A_6)$ from $P_2$ and $P_1$ respectively
  ...Multiply the matrices to form $T_6^0$
  ...Form $J_1$ from $T_6^0$
  ...Send $J_1$ to $P_0$
```

In the previous method four processors were used to solve the problem, one as controller and the other three carrying out operations. Each processor is considered as a seperate unit executing its operation sequentially but at the same time the other processors are doing the same thing. Hence, the whole network is running in parallel. Any number of methods could be used to divide the problem but the best allocation would be, of course, to solve the problem as fast as possible. To do so, each processor should be kept busy performing useful computations. If a large amount of data must be transferred between the processors, a potential communication bottleneck may occur which would slow down the network. To avoid that, only the first 3 rows of the (4×4) transformation matrices are transferred between different processors. Another restriction is the matrix by matrix multiplication. In this case the operation is performed in a way which avoids useless multiplications and additions, i.e.

$$\begin{aligned} \mathbf{T}_i^{i-1}\,\mathbf{T}_{i+1}^i &= \begin{pmatrix} \mathbf{R}_i^{i-1} & \mathbf{P}_i^{i-1} \\ 0^T & 1 \end{pmatrix} \begin{pmatrix} \mathbf{R}_{i+1}^i & \mathbf{P}_{i+1}^i \\ 0^T & 1 \end{pmatrix} \\ &= \begin{pmatrix} \mathbf{R}_i^{i-1}\mathbf{R}_{i+1}^i & \mathbf{R}_i^{i-1}\mathbf{P}_{i+1}^i + \mathbf{P}_i^{i-1} \\ 0^T & 1 \end{pmatrix} \end{aligned} \tag{15}$$

4.2 Method 2

This method is used with the same network configuration (Fig.2) as that used in method 1. The main difference is a reduction in the communication between the slave processors by redistributing the job to increase the independence of each processor. Therefore, each processor is spending its time doing useful computations instead of waitting to receive data from other processors. The following job schedule is used

- P_1 : compute the fifth and the sixth columns of (J) using eq.(8,9) respectively.
- P_2 : compute the third and the fourth columns of (J) using eq.(6,7) respectively.
- P_3 : compute the first and the second columns of (J) using eq.(4,5) respectively.

The algorithm will proceed as follows

1. P_0 sends θ_1, θ_2 to P_3, θ_3, θ_4 to P_2, and θ_5, θ_6 to P_1, and this is performed in parallel. Then, P_0 will start receiving (J) columns from the different processors, i.e.

```
SEQ
  PAR
     ...Send $\theta_1$ and $\theta_2$ to $P_3$
     ...Send $\theta_3$ and $\theta_4$ to $P_2$
     ...Send $\theta_5$ and $\theta_6$ to $P_1$
  PAR
     ...Receive $J_1$ and $J_2$ from $P_3$
     ...Receive $J_3$ and $J_4$ from $P_2$
     ...Receive $J_5$ and $J_6$ from $P_1$
```

2. This stage is divided into 3 substages working in parallel together, but each substage is running sequentially

   (a) Processor P_1

```
SEQ
   ...Form $A_5$ and $A_6$
   ...Multiply $A_5$ by $A_6$ to form $T_6^4$
   ...Send $T_6^4$ to $P_2$ and $P_3$
   ...Form $J_5$ by using $T_6^4$
   ...Form $J_6$ by using $A_6$
   ...Send $J_5$ and $J_6$ to $P_0$
```

   (b) Processor P_2

```
SEQ
   ...Form $A_3$ and $A_4$
   ...Multiply $A_3$ by $A_4$ and store in $T_1$
   ...Send $T_1$ to $P_3$ and receive $T_6^4$ from $P_1$
   ...Multiply the $A_4$ by $T_6^4$
   ...Multiply the $T_1$ by $T_6^4$
   ...Form $J_3$ from $(T_1 * T_6^4)$
   ...Form $J_4$ from $(A_4 * T_6^4)$
   ...Send $J_3$ and $J_4$ to $P_0$
```

(c) Processor P_3

```
SEQ
    ...Form $A_1$ and $A_2$
    ...Multiply $A_1$ by $A_2$ and store in $T_1$
    ...Receive $(A_3*A_4)$ and $(A_5*A_6)$ from $P_2$ and $P_1$ respectively
    ...Multiply $(A_3*A_4)$ by $(A_5*A_6)$ and store in $T_2$
    ...Multiply $T_2$ by $A_2$
    ...Multiply $T_2$ by $T_1$ to form $T_6^0$
    ...Form $J_1$ by using $T_6^0$
    ...Form $J_2$ from $(A_2*T_2)$
    ...Send $J_1$ and $J_2$ to $P_0$
```

It is important to note that, the sending and receiving of matrices and data is performed in parallel. For example, if matrices T_1 and T_2 are sent from P_1 to P_2 and P_3 respectively and T_3 is received by P_1 from P_3, this operation is coded as follows :

```
SEQ i = 1 FOR 3
  SEQ j = 1 FOR 3
    PAR
      C1.2 ! T1[i][j]
      C1.3 ! T2[i][j]
      C3.1 ? T3[i][j]
```

where C1.2, C1.3, and C3.1 are communication channels implemented by OCCAM and correspond to actual hardware communication links connecting the different transputers.

4.3 Method 3

A different network (Fig.3) is used for this method to give more independence to each processor and eliminate communication between slave processors, so that the communication bottleneck is minimised.

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. Equations (4) to (9) are distributed on the six processors such that each processor works on computing one column of J, i.e. there is one processor per link.

In this case the structure of the OCCAM program is the same for the six

processors except for a slight difference in the data flow path, that is, 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 (i.e. P_6 talks to P_0 through P_3). This difference appears where position variables (θ_i) are sent from P_0 to the network and Jacobian columns (J_i) are received from different processors.
Now three types of procedures are given to illustrate the kind of code written to run on each processor

1. Processor P_0

```
SEQ
  PAR
       ...Send $\theta_1$, $\theta_2$,..., $\theta_6$to $P_3$
       ...Send $\theta_3$,..., $\theta_6$ to $P_2$
       ...Send $\theta_5$,..., $\theta_6$ to $P_1$
  PAR
       ...Receive $J_1$ and $J_2$ from $P_3$
       ...Receive $J_3$ and $J_4$ from $P_2$
       ...Receive $J_5$ and $J_6$ from $P_1$
```

2. Processor P_2 (example of a processor in level 1)

```
SEQ
  ...Receive $\theta_3$,...,$\theta_6$ from $P_0$
  ...Send $\theta_4$,..., $\theta_6$ to $P_5$
  ...Form $A_3$,...,$A_6$
  ...Multiply the chain of matrices ($A_3 * A_4 * \ldots A_6$)
  ...Form $J_3$
  ...Receive $J_4$ from $P_5$
  ...Send $J_3$ and $J_4$ to $P_0$
```

3. Processor P_5 (example of a processor in level 2)

```
SEQ
    ...Receive $\theta_4,\ldots,\theta_6$ from $P_2$
    ...Form $A_4,\ldots,A_6$
    ...Multiply the chain of matrices $(A_4*A_5*\ldots A_6)$
    ...Form $J_4$
    ...Send $J_4$ to $P_2$
```

The whole procedure is shown in a block diagram (Fig.4.)
Its important to note that in writing OCCAM code, efficiency is the main aim if real-time implementations are sought. Hence, no redundant calculations are performed and the (DH) parameters reside on each transputer (processor) in the network to minimise the communication overhead. Also, the calculation of the direct kinematics problem is solved implicitly without the need for extra processing time.

4.4 Results

The previous three algorithms are implemented using OCCAM to show their suitability for real time applications. The results are compared with a sequential version written using FORTRAN and executed on a SUN workstation (SUN Inc. 1984) with additional floating point hardware. The results are given in Table 1.

Table 1.

Algorithm	Exectution time (msec)
Sequential	15.0
Method 1	3.84
Method 2	4.67
Method 3	0.256

It can be noticed from Table 1. that the methods 1, 2, and 3 are adequate for real time applications, and this shows the superiority of parallel processing techniques. Method 3 is 58 times faster than the sequential approach. The choice between the three methods will be made according to the speed requirements needed.

5 Parallelism in the Inverse Jacobian

In this section the problem of the Inverse Jacobian is solved using two classical techniques for solving linear systems of equations; the Gaussian Elimination (GE) and the Gauss Jordan (GJ). Parallelism is introduced to both techniques to reduce the complexity of the computations. Quantification of speed up and utilisation, with real-time implementation results are included.

5.1 The Gaussian Elimination and Gauss Jordan

The GE and GJ were chosen in this study for the following reasons

1. There is straight forward, understandable, and well established literature covering the different aspects of both techniques (Press, Flannery, Teukolsky, and Vetterling 1986).
2. Parallelism can be easily introduced because it flows naturally from the structure of the sequential algorithms.
3. The use of direct methods reduces the effects of rounding errors, especially if pivoting and equilibration stratgies are used (Burden, Faires, and Reynolds 1981).
4. Divergence problems are not encountered in direct techniques, unlike iterative ones.
5. The inverse of the system matrix is solved implicitly which is the case in the Inverse Jacobian formulation.

The (GE) algorithm is distributed on the network shown in (Fig.5). The configuration used evolves from the basic structure of the algorithm of (GE) with simple row interchange. The network is divided into 4 main levels which work in the following manner

LEVEL 1
- The processor (T) prepares the (J) by augmenting it with the world coordinates vector (WCV).
- A check is performed to avoid a zero pivoting element, and a row interchange might be performed to satisfy this requirement.
- Normalisation of the row is performed by dividing the whole row by the first entry in that row (i.e. j_{11} for first row).

- The remaining five 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
$$J_k = J_k - M_{ki} * J_i \tag{16}$$
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.

- (T) will also check whether the operation is completed successfully. If not, the fault is located and corrected as fast as possible.

LEVEL 4 • Back substitution is performed on the resulting matrix. Then the value of the joint variables is sent to the output unit.

The whole procedure is repeated if a new (J) is received.

The same analysis is performed again for the case of (GJ). The network used is shown in (Fig.6.), and again the configuration evolved from the basic structure of the algorithm of (GJ) with simple row interchange.
This network is divided only into 3 levels, as follows :

LEVEL 1 • This level plays the same role as level 1 in the previous network of (GE).

LEVEL 2 • These M processors work in parallel and each processor is loaded with a row of the system matrix (J|WCV) from processor (T) in level 1.

- The processors will perform an elimination process which sets each kth element in each loaded row equal to zero. This is achieved by using the kth row in common with all the parallel processors
- The processed rows are sent to the processor (T) in level 3.
- The array of processors is ready to receive other rows (if any).

LEVEL 3 • This processor will receive the processed rows sent from level 2 and will form a new system matrix, which is checked if further processing is needed. If so, it will be sent back to level 1 again and the previous operation repeated. If not, the values of the unknowns are calculated and sent to the output unit.

The whole procedure is repeated if a new (J) is received. An analysis was performed to check the cost effectiveness and efficiency of (Level 2) in both networks. Level 2 can be implemented by multiple processors, but in this work only the options of one, two and three processors have been studied to minimise the cost and complexity of the network. The criteria used to quantify the suitability of each option were processor utilisation and system throughput . Utilisation is defined to be the ratio of the processing time of each processor to the total processing time of the array. System throughput is the number of basic computations (multiplications/additions) processed per unit time. The analysis starts first by assuming that there is only one processor in Level 2, then two processors, finishing with three processors. The results are shown in table 2, 3, and 4.
Real time implementation for a 6 dof arm is also included and the results agree with the analysis. The results show that when three processors are used in level 2, the system throughput and total time in case of (GE) are better than (GJ).
In table 3, a better utilisation is achieved using (GJ), but it can be seen from table 4 that a better processing time is approached by using (GE). To reach a compromise, the (GE) is selected because the main interest is to get good real time results. In table 5, real time results are given for the two techniques and comparisons are made with sequential implementations.

Graphical illustrations are also given to show the results more clearly.(See Fig. 7, 8, 9, 10)

Table 2.

	GE			GJ		
	Number of Processors					
	1	2	3	1	2	3
Total Time (time units)	312	184	146	375	225	150
Throughput (operation\time)	1.63	2.77	3.64	1.44	2.4	3.6

Table 3.

Number of Processors	GE			GJ		
	Utilisation (%)					
	1	2	3	1	2	3
1	100			100		
2	100	69.9		100	66.7	
3	100	76.1	47.1	100	100	50

Table 4.

Number of Processors	GE			GJ		
	Processing time (time units)					
	1	2	3	1	2	3
1	312.5			375		
2	184	128.5		225	150	
3	140	106.5	66	150	150	75

Table 5.

Algorithm	Exectution time (msec)
Sequential	*22 **30
Parallel	*4.48 **5.38

$*$ GE
$**$ GJ

5.2 Inverse Jacobian near Singularity Points

At singular configurations, the manipulator loses one or more degrees of freedom and the determinant of (J) approaches zero. Even more important sometimes is the fact that in the vicinity of singular points, unpractically high joint velocities are required to move the arm with a reasonable speed.

The problem of degeneracy and robustness of the (J^{-1}) is a very interesting and important issue which must be treated carefully (Uchiyama 1979; Lai and Yang 1986; Aboaf and Paul 1987; Rivin 1988). Several solutions have been proposed to solve for the problem, such as redesigning the work space in a way to avoid degenerative situations, and the manufacturing of singularity-proof robot wrists.

In mathematical terms, singularity affects the rank of (J). If this value is less than 6 (in case of 6 dof), then no unique solution exists; one or more of the rows is redundant, depending on the right hand side vector (WCV). To determine the rank, triangularisation by (GE) can be applied and , if no zeros show up on the diagonal of the final triangularised (J), the rank is equal to 6 (full rank) and a unique solution exists. If, in spite of pivoting, one or more zeros occur on the the final diagonal, there is no unique solution.

Tucker and Perreria (1987) used generalised inverses techniques (Moore-Penrose inverse) to obtain the solution for robots with less than, equal to, or greater than 6 dof. Singularities were also investigated using the same technique which employs the Singular Value Decomposition (SVD) at some stage.
The (SVD) is used to solve for very ill conditioned matrices and it requires solving for the eigenvalues and eigenvectors. This involves heavy computations that make the real-time implementation a difficult task to accomplish (Press, Flannery, Teukolsky, and Vetterling 1986). In the previous discussion, the suitability of the (GE) was proved, and in this section the robustness of (GE) is improved by using pivoting strategies and distributing

the job on the network given in (Fig.5). The analysis is supported by two examples and a (PUMA 560) arm is used as a case study.

Example 1 *For $\theta_i = (90, 0, 90, 0, 90, -90)$, the Jacobian is;*

$$\mathbf{J} = \begin{pmatrix} -0.865 & 0.0 & 0.0 & -0.056 & 0.0 & 0.0 \\ 0.149 & 0.036 & 0.036 & 0.0 & 0.056 & 0.0 \\ 0.0 & 0.865 & 0.433 & 0.0 & 0.0 & 0.0 \\ -1.0 & 0.0 & 0.0 & 0.0 & 0.0 & 1.0 \\ 0.0 & 0.0 & 0.0 & -1.0 & 0.0 & 0.0 \\ 0.0 & -1.0 & -1.0 & 0.0 & -1.0 & 0.0 \end{pmatrix}$$

the determinant of $\mathbf{J} = 0.0075$, *and with such a small value the use of (GE) with simple row interchange might not be the right choice in terms of efficiency and accuracy. So, a row equilibration technique (Forsysth and Moler 1967; Burden, Faires, and Reynolds 1981) is used. The solution of eq.(12) is changed to the solution of the following equation,*

$$\mathbf{D}^{-1}\mathbf{J}\delta\mathbf{x} = \mathbf{D}^{-1}\delta\theta \tag{17}$$

where
$\mathbf{D}$ *is a diagonal matrix whose i^{th} entry is (e_i),*

$$e_i = \max_{1\leq k\leq 6} \mid \mathbf{J}_{i,k} \mid$$

where $i = 1, \ldots, 6$ and the row interchange is performed in such a way that the pivoting element is selected to be the largest absolute value in the same column that is below the diagonal, i.e.

$$\mid j^k_{p,k} \mid = \max_{k\leq i\leq 6} \mid \mathbf{J}^k_{i,k} \mid$$

which is purely a row interchange.
This technique is known as GE with Scaled Column Pivoting (GESCP) and it managed to find the solution for $\mathbf{J}^{-1}$ within real-time constraints. (See table 6)

Example 2 *In this case a less robust situation is addressed.*
For $\theta_i = (90, 90, 0, 0, 0.001, 90)$, *the Jacobian is;*

$$\mathbf{J} = \begin{pmatrix} 0.489 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & -0.49 & -0.489 & 0.0 & -0.056 & 0.0 \\ -0.149 & -0.411 & 0.021 & 0.0 & 0.0 & 0.0 \\ -0.001 & 0.0 & 0.0 & 1.0 & 0.0 & 1.0 \\ 1.0 & 0.0 & 0.0 & 0.001 & 0.0 & 0.0 \\ 0.0 & 1.0 & 1.0 & 0.0 & 1.0 & 0.0 \end{pmatrix}$$

the determinant of $\mathbf{J} = 9 \times 10^{-5}$, *and with a value that is nearly approaching zero, a very well conditioned technique should be used. So a total pivoting approach, the GE with Maximal Pivoting (GEMP) which incorporate row and column interchanges, is used to guarantee an accurate solution.*
The pivoting element is selected to be

$$| j^k_{p,k} | = \max_{k \le i,l \le 6} | \mathbf{J}^k_{i,l} |$$

which is accomplished by both row and column interchanges. A row and a column equilibration is used to assure the convergance of the solution. A diagonal matrix $\mathbf{B}$ *is assumed to have an* i^{th} *entry* f_i, *where*

$$f_i = \max_{1 \le k \le 6} | \frac{\mathbf{J}_{k,i}}{e_k} |$$

where $i = 1, \ldots, 6$. *The system of equations which is solved is*

$$\mathbf{D}^{-1}\mathbf{J}\mathbf{B}\delta\mathbf{x} = \mathbf{D}^{-1}\delta\theta \qquad (18)$$

The results obtained show the possibility of applying (GEMP) and still satisfying real time-conditions. (See table 6)

Table 6.

Algorithm	GESCP	GEMP
Sequential (msec)	60.0	120.0
Parallel (msec)	5.7	7.74

6 Conclusions

The Kinematical description of a typical robot manipulator such as the PUMA 560 is systematic and simple in concept but complicated in respect of the computational burden inherent in real-time control applications. A major cause of this complication is the computation of the Jacobian and Inverse Jacobian, and it is this difficulty which the work described in this paper has addressed.

A suitable algorithm for computing the Jacobian has previously been presented by Paul (1981). Three alternative ways of distributing the computation of this algorithm on a general purpose distributed computing system have been suggested in this paper. The computational efficiency of these methods has been compared with that for an equivalent sequential implementation. This comparison has demonstrated the efficiency of all three methods and shown the feasibility of using them in real-time applications.

Two classical techniques for computing the Inverse Jacobian; Gaussian Elimination and a Gauss Jordan method have been investigated . Parallelism at both job and task levels was introduced into these to reduce the computational burden. Simulation of their real-time performance has shown that the Gaussian Elimination technique is superior. The case of the robot arm operating near singular points has also been considered. Two different pivoting techniques were used which allow the real-time constraints to be met whilst still guaranteeing the robustness and stability of the Gaussian Elimination.

The work described has demonstrated how the recent advances in computer technology, particularly in new VLSI architectures, can be utilised beneficially in the implementaion of Jacobian based control schemes. Suitable foundations have been set for the development of a wide range of control algorithms, unhindered by computational restrictions.

7 References

- Aboaf, E. W., and Paul, R. P. 1987. Living with the singularity of robot wrists. Proc. 1987 IEEE Int. Conf. on Robotics and Automation, pp. 1713-1717.

- Ang, M. H., and Tourassis, V. D. 1987. General purpose inverse kinematics transformations for robotic manipulators. J. Robotic Systems, 4(4), pp. 527-549.

- Angeles, J. 1985. On the numerical solution of the inverse kinematic problem. Int. J. Robotics Research, 4(2), pp. 21-37.
- Ben-Israel, A., and Greville, T. N. E. 1974. Generalized Inverses : Theory and Applications. Willey, NEW YORK.
- Burden, R. L., Faires, J. D., and Reynolds, A. C. 1981. Numerical Analysis, PWS Publishers.
- Craig, J. J. 1986. Introduction to Robotics : mechanics and control. Addison- Wesley, Reading, Mass.
- Denavit, J., and Hartenberg, R. S. 1955. A kinematic notation for lower-pair mechanisms based on matrices. Trans. ASME J. Applied Mechanics, vol. 22, pp. 215-221.
- Elgazzar, S. 1984. Efficient solution for the kinematic positions for the PUMA 560 robot. rep. NRC/ERB-973, National research council of Canada.
- Elgazzar, S. 1985a. Efficient solution for the kinematic velocities and accelerations for the PUMA 560 robot. rep. NRC/ERB-974, National research council of Canada.
- Elgazzar, S. 1985b. Efficient kinematic transformation for the PUMA 560 robot. IEEE J. Robotics and Automation, vol. 1, no. 3, pp. 142-151.
- Featherstone, R. 1983a. Position and velocity transformation between robot end-effector coordinates and joint angles. Int J. Robotics Research, 2(2), pp. 35-45.
- Featherstone, R. 1983b. Calculation of robot joint rates and actuator torques from end-effector velocities and applied forces. Mechanism and Machine Theory. 18(3), pp. 193-198.
- Forsyth, G. , and Moler, C. B. 1967. Computer Solution of Linear Algebraic Systems. Prentice Hall, Englewood Cliffs, N.J.
- Fu, K. S., Gonzales, R. C., Lee, C. S. G. 1987. Robotics : Control, Sensing, Vision, and Intelligence. McGraw-Hill, New York.
- Hamblen, J. O. 1987. Parallel continuous system simulation using the transputer. Simulation, 49(6), pp. 249-253.

- Hwang, K., and Briggs, F. A. 1985. Computer Architecture and Parallel Processing. McGraw-Hill, New York.
- IEE Workshop. 1987. Parallel Processing in Control- the Transputer and other architectures, UCNW, Bangor, Wales, U.K.
- IEE Workshop. 1988. Parallel Processing in Control- the Transputer and other architectures, UCNW, Bangor, Wales, U.K.
- INMOS Ltd. 1984. OCCAM programming manual. Prentice-Hall, Englewood Cliffs, N.J.
- INMOS Ltd. 1985. IMS T414 reference manual.
- INMOS Ltd. 1986. IMS T414 transputer, Preliminary Data.
- Jones, D. I. 1985. OCCAM structures in control applications. Trans. of Inst. of Measurement and Control, vol. 7, no. 5, pp. 222-227.
- Kerridge, J. 1987. OCCAM Programming : A Practical Approach. Blackwell.
- Kung, H. T. 1982. Why systolic architectures. IEEE Computer, pp. 37-46.
- Lai, Z. C., and Yang, D. C. H. 1986. A new method for the singularity analysis of simple six-link manipulators. Int. J. Robotics Research, 5(2), pp. 66-74.
- Lathrop, R.H. 1985. Parallelism in manipulator dynamics. Int. J. Robotics Research, 4(2), pp. 80-102.
- Lawson, C. L., and Hanson, R. J. 1974. Solving least squares problems. Prentice-Hall, Englewood Cliffs, N.J.
- Leahy, M. B., Nugent, L. M., Saridis, G. N., and Valavanis, K. P. 1987. Efficient PUMA manipulator jacobian calculation and inversion. J. Robotic Systems, 4(2), pp. 185-197.
- Lee, C. S. G., and Chang, P. R. 1986. Efficient parallel algorithm for robot inverse dynamics computation. IEEE Trans. System, Man, and Cybernetics, vol. 16, no. 4, pp. 532-542.

- Lee, C. S. G., and Chang, P. R. 1988. Efficient parallel algorithm for robot forward dynamics computation. IEEE Trans. System, Man, and Cybernetics, vol. 18, no. 2, pp. 238-251.
- Lenarcic, J. 1984. A new method for calculating the jacobian for a robot manipulator. Robotica, vol. 1, part 4, pp. 205-209.
- Luh, J. Y. S., Walker, M. W., and Paul, R. P. C. 1980. Resolved acceleration control of mechanical manipulators. IEEE Trans. Automatic Control, 25(3), pp. 468-474.
- Luh, J. Y. S., and Lin, C. S. 1982. Scheduling of parallel computation for a computer controlled mechanical manipulator. IEEE Trans. System, Man, and Cybernetics, vol. 12, no. 2, pp. 214-234.
- Melouah, H., and Andre, P. 1982 (Florida). High speed computation of the inverse jacobian matrix and of servo inputs for robot arm control. Proc. of 21st IEEE Conf. Decision and Control, pp. 89-94.
- Mitra, S. K., and Mahalanabis, A. 1984. Fast computation of jacobian and inverse jacobian of robot manipulators. Proc. of SPIE Int. Soc. of Optical Eng., Intelligent Robots and Computer Vision, vol. 521, pp. 235-242.
- Orin, D. E., and Schrader, W. W. 1984. Efficient computation of the jacobian for robot manipualtors. Int. J. Robotics Research, 3(4), pp. 66-75.
- Orin, D. E., Chao, H. H., Olson, K. W., and Schrader, W. W. 1985. Pipeline/Parallel algorithms for the jacobian and inverse dynamics computations. Proc. IEEE Int. Conf. Robotics, pp. 785-789.
- Paul, R. P. 1981. Robot manipulators : Mathematics, Programming, and Control, MIT Press, Cambridge, Mass.
- Paul, R. P., and Zhang, H. 1986. Computationally efficient kinematics for manipulators with spherical wrists based on the homogeneous transformation representation. Int J. Robotics Research, 5(2), pp. 32-44.
- Press, W. H., Flannery, B. P., Teukolsky, S. A., and Vetterling, W. T. 1986. Numerical Recipes : The Art of Scientific Computing. Cambridge University Press.

- Renaud, M. 1981 (Oct, Tokyo). Geometric and kinematic models of a robot manipulator : calculation of the jacobian and its inverse. Proc. 11th Int. Symp. Industrial Robots.
- Rivin, E. I. 1988. Mechanical Design of Robots. McGraw-Hill, New York.
- SUN Microsystems Inc. 1984. FORTRAN programmer's Guide for the sun workstations, 2550 garcia avenue, mountain view, CA 94043.
- Tsai, Y. T., and Orin, D. E. 1987. A strictly convergent real time solution for inverse kinematics of robot manipulators. J. Robotic Systems, 4(4), pp. 477-501.
- Tucker, M., and Perreira, N. D. 1987. Generalized inverses for robotic manipulators. Mechanism and Machine Theory, vol. 22, no. 6, pp. 507-514.
- Uchiyama, M. 1979. A study of computer control of motion of a mechanical arm. Bull. Jpn. Soc. Mech. Eng., 22(173), pp. 1640-1647.
- Vukobratovic, M., Kircanski, N., and Li, S. G. 1988. An approach to parallel processing of dynamic robot models. Int J. Robotics Research, 7(2), pp. 64-71.
- Warldon, K. J. 1982. Geometrically based manipulator rate control algorithms. Mechanism and Machine Theory, vol. 17, no. 6, pp. 379-385.
- Whitney, D. E. 1969. Resolved motion rate control of manipulators and human prostheses. IEEE Trans. Man Machine Systems, vol. 10, no. 2, pp. 47-53.
- Whitney, D. E. 1972. The mathematics of coordinated control of prosthetic arms and manipulators. Trans. ASME., J. Dynamic Systems, Measurement, and Control, vol. 94, no. 4, pp. 303-309.
- Whitney, D. E. 1987. Historical perspective and state of the art in robot force control. Int. J. Robotics Research, 6(1), pp. 3-14.
- Wolovich, W. A. 1987. Robotics : Basic Analysis and Design. Holt, Rinehart and Winston, The Dryden Press, Saunders College.

- Wu, C. H., and Paul, R. P. 1982. Resolved motion force control of robot manipulators. IEEE Trans. System, Man, and Cybernetics, vol. 12, no. 3, pp. 266-275.
- Zakharov, V. 1984. Parallelism and array processing. IEEE Trans. Computers, vol. 33, no. 1, pp. 45-78.

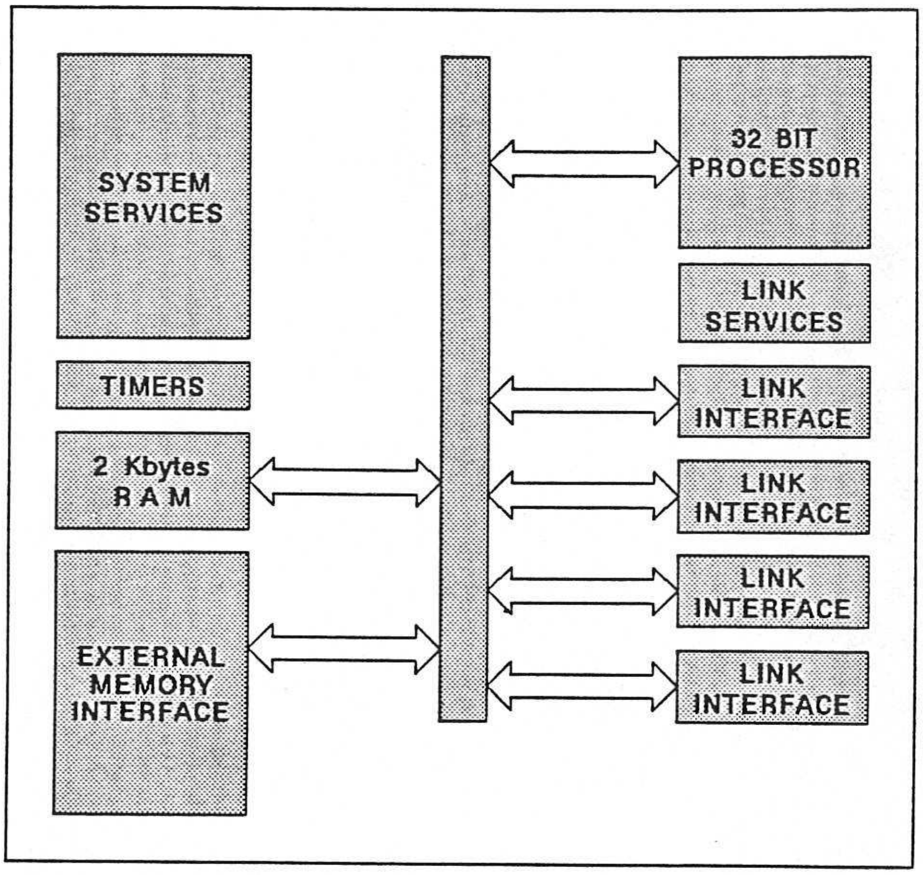
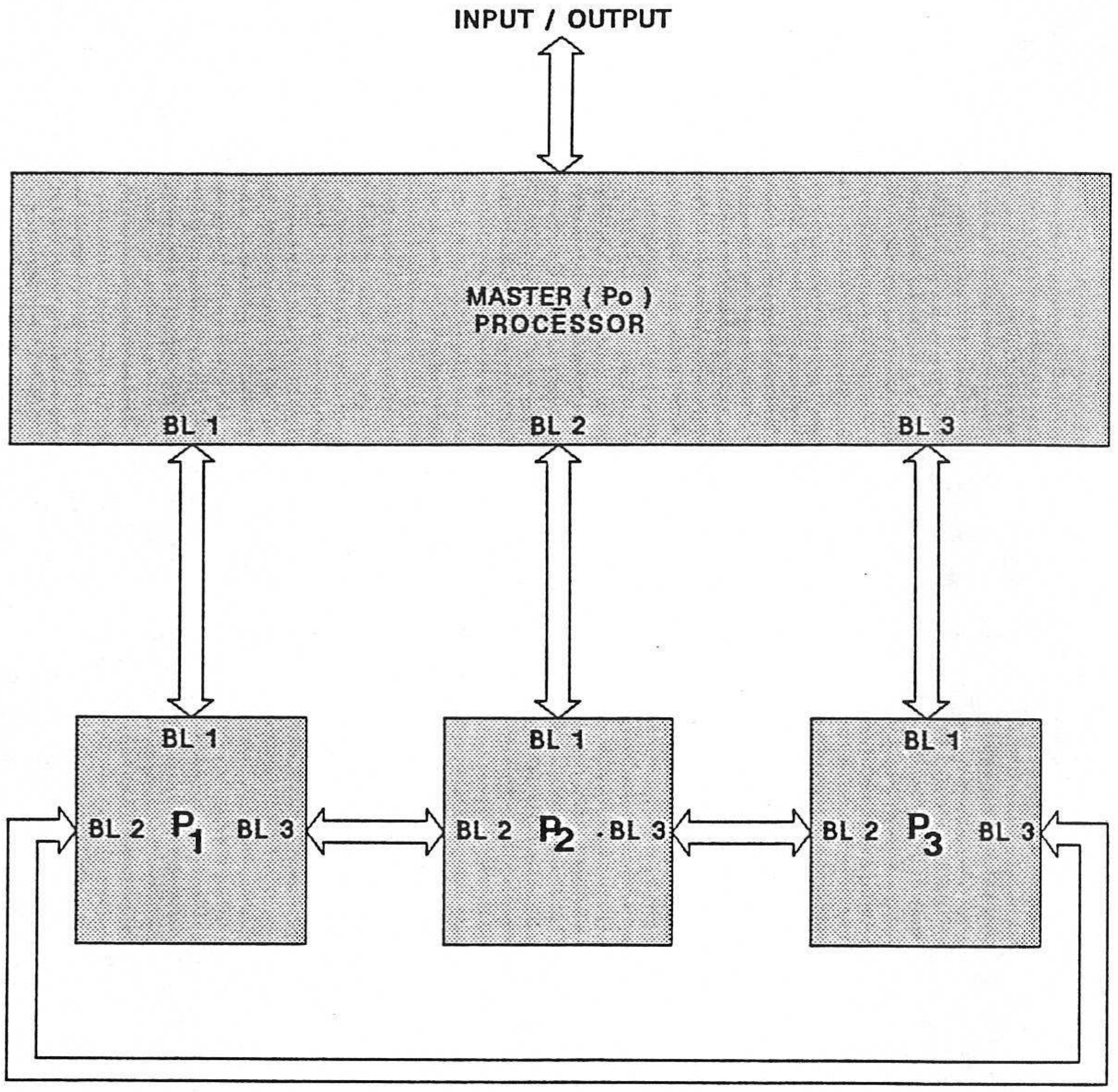


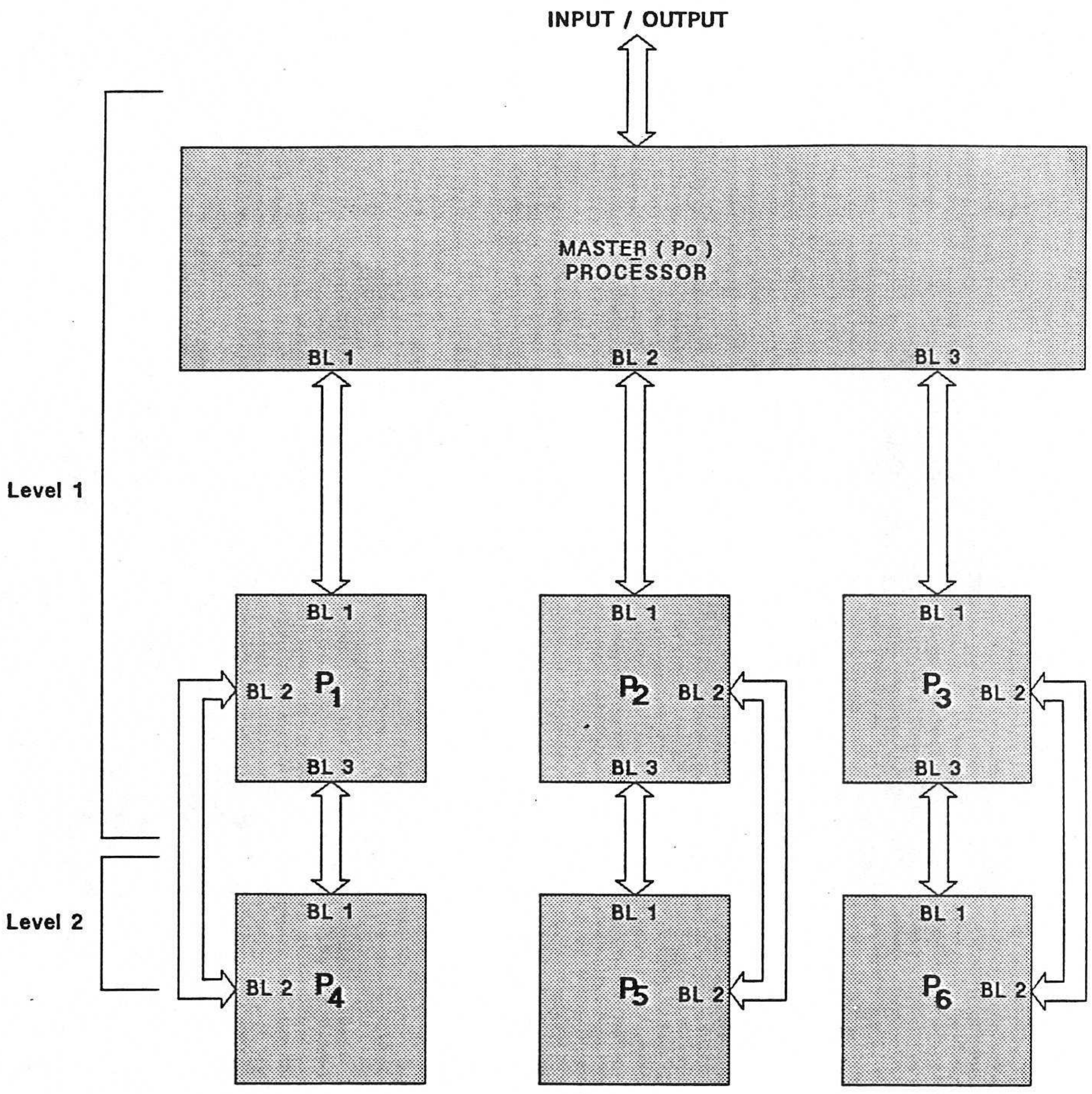
Figure 1. The INMOS T414 TRANSPUTER.



BL : Bidirectional Link

P : Processor or Transputer.

Figure 2. Simple Tree Structured Network used for Method 1 and Method 2.



BL : Bidirectional Communication Link

P : Processor or Transputer.

Figure 3. A Two Levels Network used for Method 3.

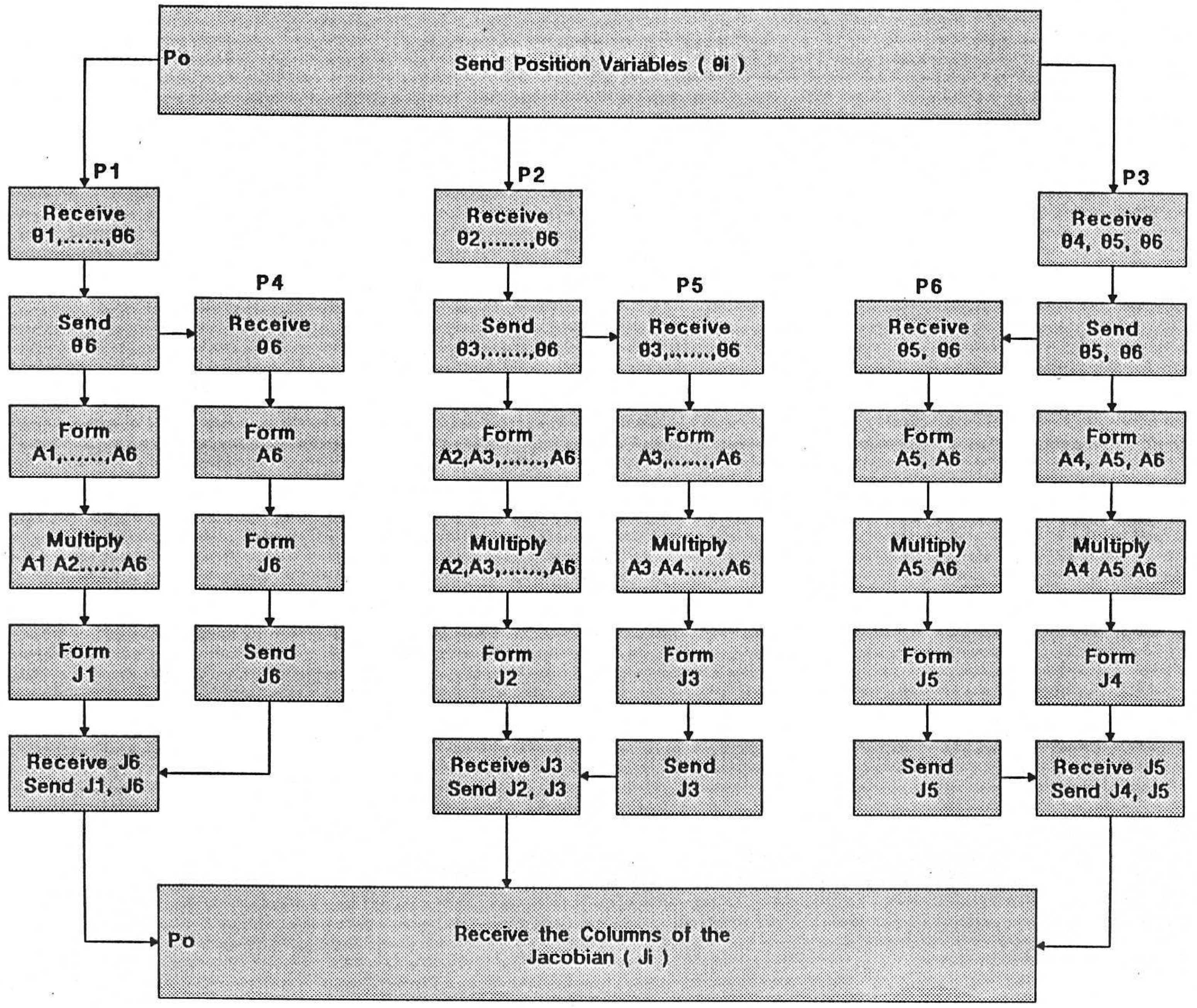


Figure 4. A Block Diagram Presenting the Complete Algorithm of Method 3.

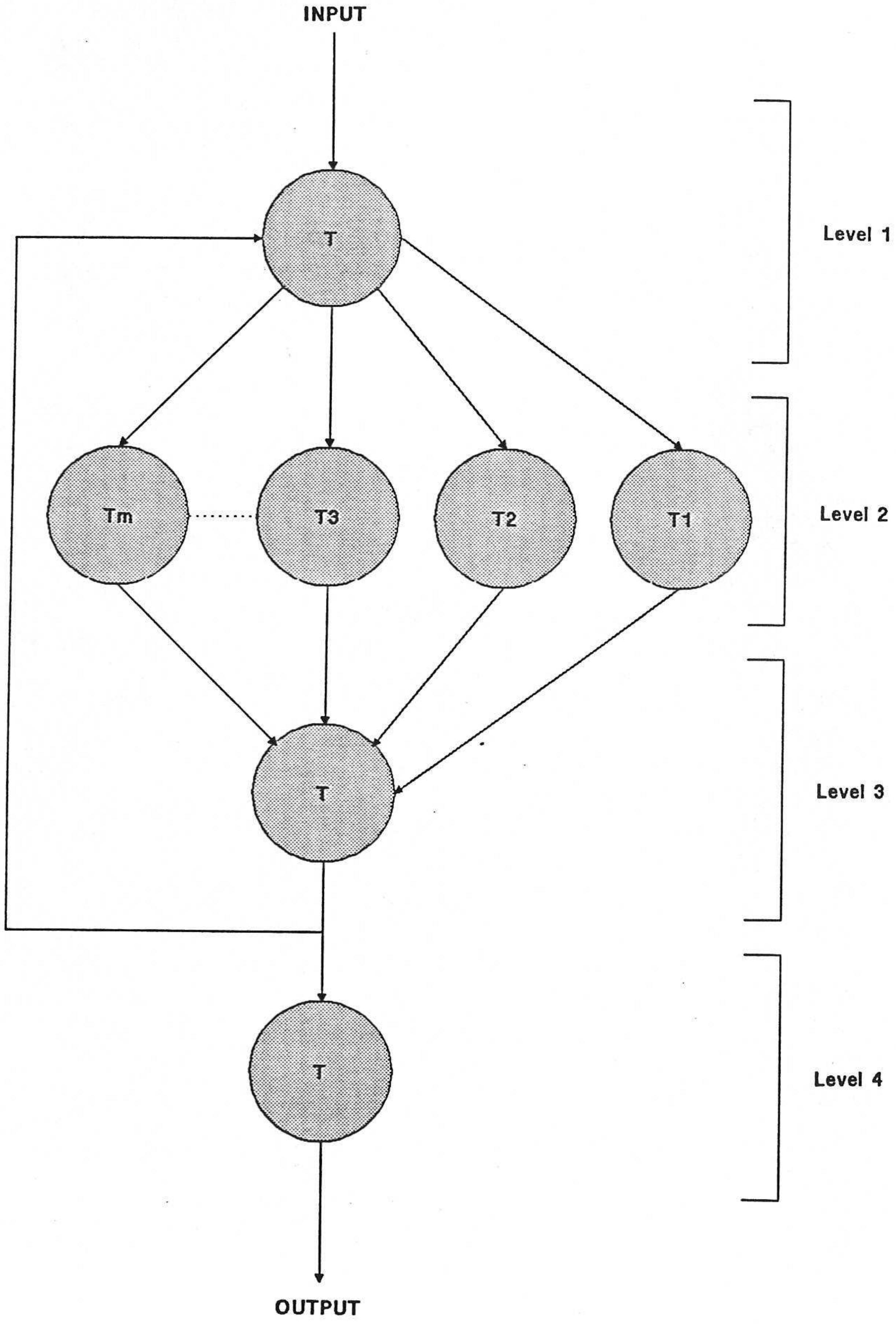


Figure 5. A Four Levels Network Scheduling the Gaussian Elimination Algorithm.

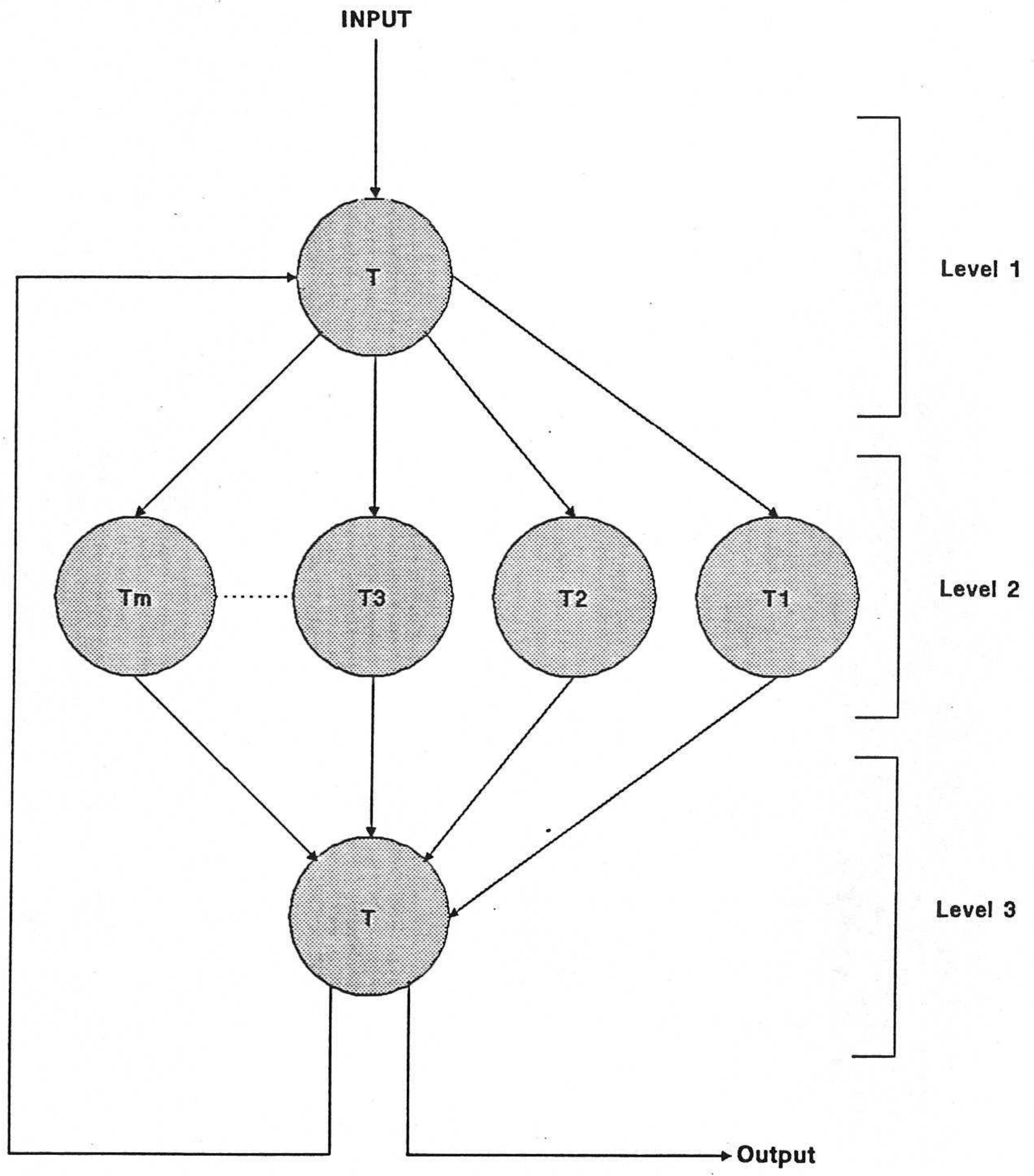


Figure 6. A Three Levels Network Scheduling the Gauss Jordan Algorithm.

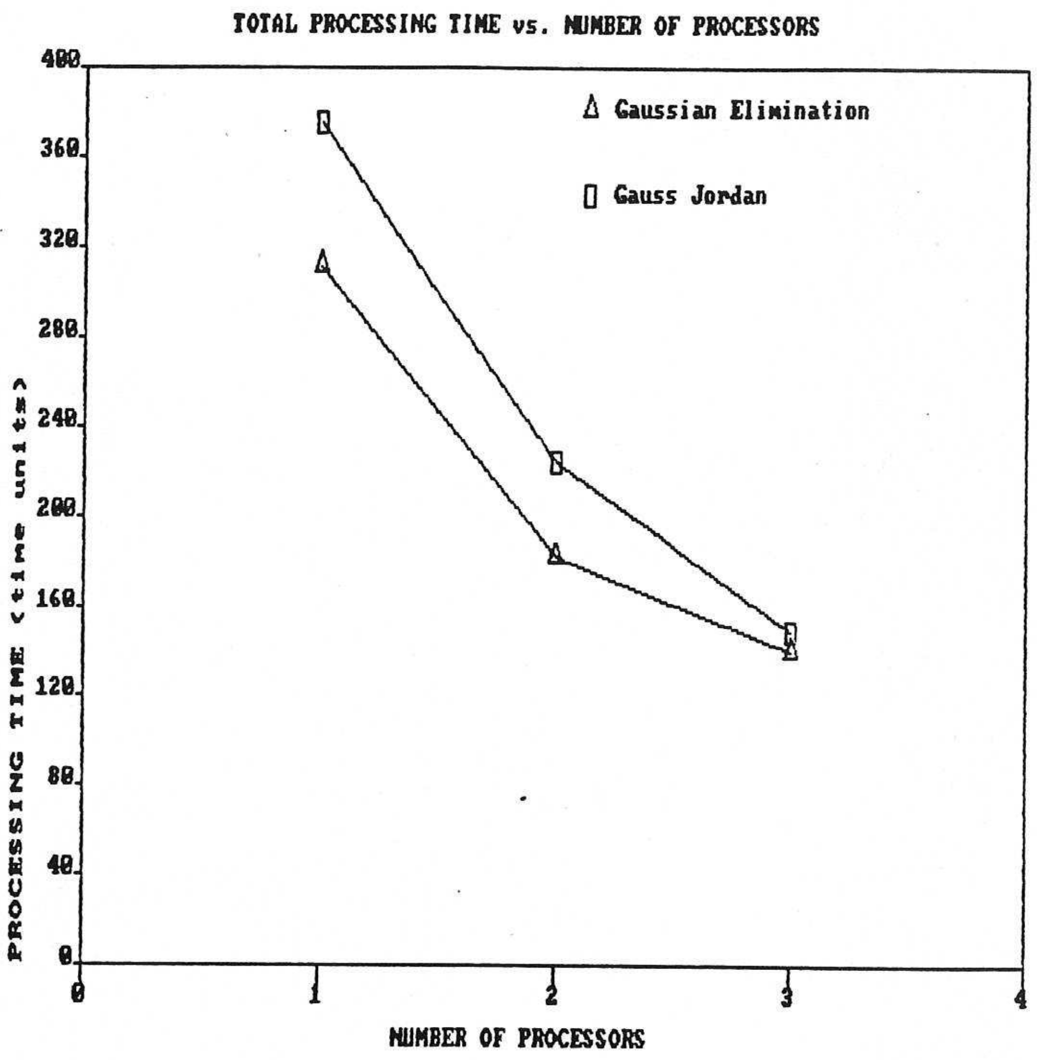


Figure 7. Total Processing Time versus Number of Processors.

Fig. 7

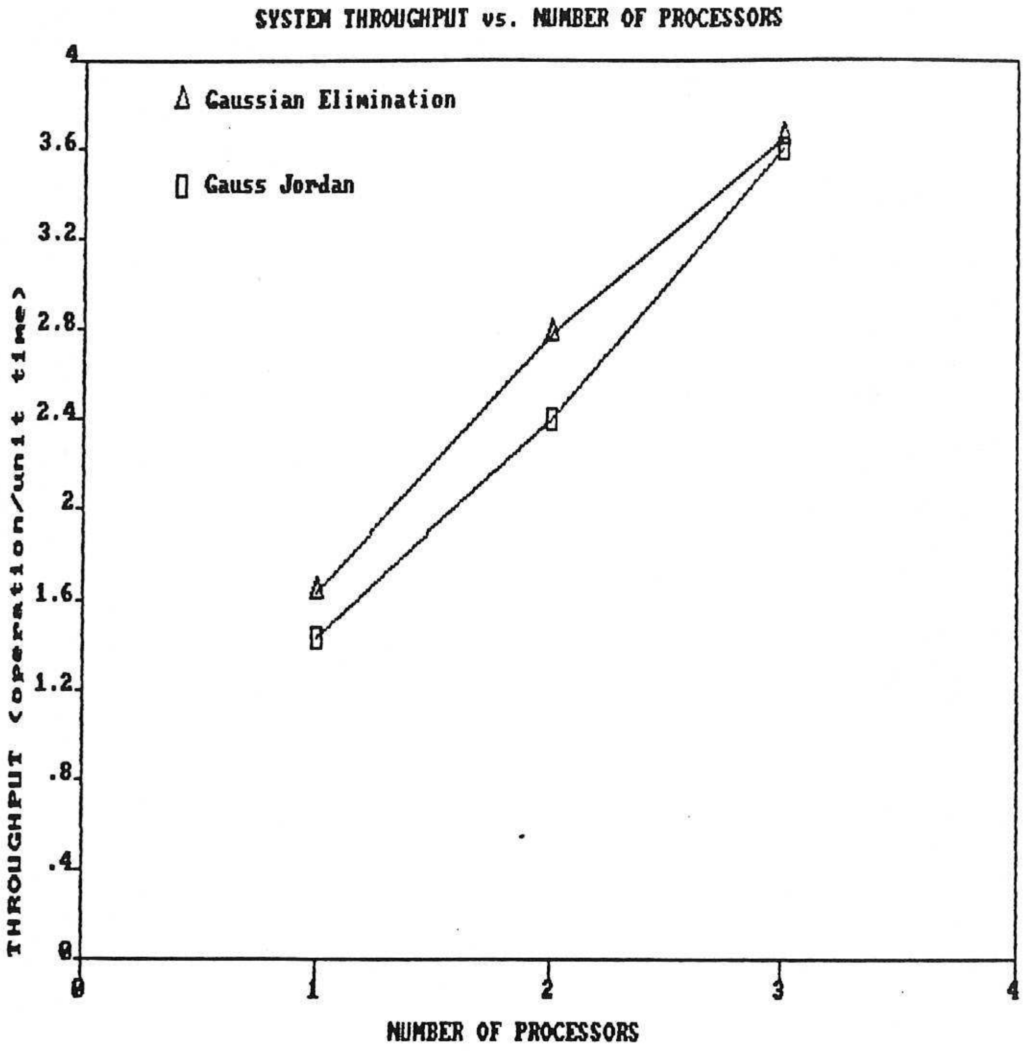


Figure 8. System Throughput versus Number of Processors.

Fig. 8