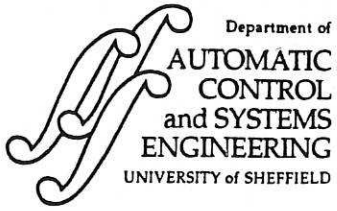

HOMOGENEOUS AND HETEROGENEOUS PARALLEL ARCHITECTURES IN REAL-TIME SIGNAL PROCESSING AND CONTROL

M O Tokhi and M A Hossain

Department of Automatic Control and Systems Engineering, The University of Sheffield,
P O Box 600, Mappin Street, Sheffield, S1 4DU, UK.

Tel: + 44 (0)114 2825136.
Fax: + 44 (0)114 2731 729.
E-mail: O.Tokhi@sheffield.ac.uk.

Research Report No. 558

January 1995

Abstract

This paper presents an investigation into the real-time performance of homogeneous and heterogeneous parallel architectures in signal processing and control applications. Several algorithms of regular and irregular nature are considered. These are implemented on a number of uni-processor and multi-processor parallel architectures. Hardware and software resources, capabilities of the architectures and characteristics of the algorithms are considered for suitable matching between the algorithms and the architectures. The partitioning and mapping of the algorithms on the architectures and inter-processor communication techniques are investigated. Finally, a comparison of the results of implementations is made to establish merits of design and development of parallel architectures for real-time signal processing and control applications.

Key words: Homogeneous architecture; Heterogeneous architecture; Digital signal processing; Parallel processing; Real-time signal processing and control..

CONTENTS

LIST OF TABLES AND FIGURES

1 Introduction

Parallel processing (PP) is currently a subject of widespread interest among scientists. The concept of PP on different problems or different parts of the same problem is not new. Discussions of parallel computing machines are found in the literature at least as far back as the 1920s (Denning, 1986). It is noted that, throughout the years, there has been a continuing research effort to understand parallel computation (Hocney and Jesshope, 1981). Such effort has intensified dramatically in the last few years, with hundreds of projects around the world involving scores of different parallel architectures for all kinds of applications, including signal processing, control, artificial intelligence, pattern recognition, computer vision, computer aided design and discrete event simulation.

The performance demands of modern signal processing and control require the employment of complex algorithms with demanding operations. This, in turn, leads to shorter sampling times. Many demanding complex signal processing and control algorithms cannot be satisfactorily realised with conventional uni-processor and multiprocessor systems. Alternative strategies where multi-processor based systems are employed, utilising high performance reduced instruction set computer (RISC) processors, digital signal processing (DSP) devices, transputers and PP techniques, could provide suitable methodologies (Ching and Wu, 1989; Jones, 1989; Tokhi et. al, 1992).

In a conventional parallel system all the processing elements (PEs) are identical. This architecture can be described as homogeneous. However, many signal processing and control algorithms are heterogeneous, as they usually have varying computational requirements. The implementation of an algorithm on a homogeneous architecture is constraining, and can lead to inefficiencies because of the mismatch between the hardware requirements and the hardware resources. In contrast, a heterogeneous architecture having PEs of different type and features can provide a closer match with the varying hardware requirements and, thus, lead to performance enhancement. However, the relationship between algorithms and heterogeneous architectures for real-time control systems is not clearly understood (Megson, 1992). The mapping of algorithms onto heterogeneous architectures is, therefore, especially challenging. To exploit the heterogeneous nature of

the hardware it is required to identify the heterogeneity of the algorithm so that a close match be forged with the hardware resources available (Baxter et. al, 1994).

One of the challenging aspects of PP, as compared to sequential processing, is how to distribute the computational load across the PEs. This requires a consideration of several issues, including the choice of algorithm, the choice of processing topology, the relative computation and communication capabilities of the processor array and partitioning the algorithm into tasks and the scheduling of these tasks (Crummey et. al, 1994). It is, thus, essential to note that in implementing an algorithm on a parallel computing platform, a consideration of the issues related to the interconnection schemes, the scheduling and mapping of the algorithm on the architecture, and the mechanism for detecting parallelism and partitioning the algorithm into modules or sub-tasks, will lead to a computational speedup (Khokhar et. al, 1993).

Each PE in a parallel architecture possesses its own special features suitable for specific applications. This leads to the inherent difficulties in exploring comparative performance evaluation of different architectures. To explore the real-time performance in particular applications it is essential to implement the algorithm of that application into the architectures. This paper presents an investigation into the performance evaluation of parallel architectures incorporating high performance RISC, DSP and transputer processors in real-time applications. A comparative study of the hardware and software resources and their real-time computational performances in implementing several complex and demanding signal processing and control algorithms is carried out. The algorithms considered include, a fast Fourier transform (FFT) algorithm, a second order correlation algorithm, two adaptive filtering algorithms, a simulation algorithm of a flexible manipulator system and simulation, identification and active vibration control (AVC) algorithms for a flexible beam system. The algorithms considered are described and classified according to their degree of regularity.

The algorithms are implemented on a number of uni-processor and multi-processor, homogeneous and heterogeneous, parallel computing platforms incorporating the Intel 80860 (i860) RISC processor the Texas Instruments TMS320C40 (C40) DSP device and

the T805 (T8) transputer. The 3L Parallel C and ANSI C programming languages are used to develop the coding of the algorithms. The performance in each case is assessed and comparison of the results of these implementations, on the basis of real-time inter-processor communication and computation performance is made and discussed.

2 Hardware

Three PEs namely, an i860, a C40 and a T8 are utilised to develop four different heterogeneous and homogeneous parallel architectures. In general, the nature of any parallel architecture reflects the nature of its PEs and the algorithms. Therefore, to compare the performance of the parallel architectures, it is essential to explore the facility and features of the PEs and the parallel architecture itself. The parallel architectures with their PEs are described below.

2.1 Processing elements

The i860 is a high-performance 64-bit vector processor with 40 MHz clock speed, a peak integer performance of 40 million instructions per second (MIPS), 8 kBytes data cache and 4 kBytes instruction cache and is capable of 80 million floating-point operations per second (MFLOPS). It is the Intel's first superscalar RISC processor possessing separate integer, floating-point, graphics, adder, multiplier and memory management units. The i860 executes 82 instructions, including 42 RISC integer, 24 floating-point, 10 graphics, and 6 assembler pseudo operations in one clock cycle. All external or internal address buses are 32-bit wide and the external data path or internal data bus is 64-bits wide. However, the internal RISC integer ALU is only 32 bits wide. The instruction cache transfers 64 bits per clock cycle, equivalent to 320 Mbytes/sec at 40 MHz. In contrast, the data cache transfers 128 bits per clock cycle. There are two floating-point units, namely, the multiplier and the adder units, which can be used separately or simultaneously under the co-ordination of the floating point control unit. Special dual-operation floating-point instructions such as add-and-multiply and subtract-and-multiply use both the multiplier and

adder units in parallel. Furthermore, both the integer and the floating-point control units can execute concurrently (Hwang, 1993).

The C40 is a high performance Texas Instruments 32-bit DSP processor with 40 MHz clock speed, 8 kBytes on-chip RAM, 512 bytes on-chip instructions cache and is capable of 275 million operation per second (MOPS) and 40 MFLOPS. This DSP processor possesses six parallel high speed communication links for inter-processor communication with 20 Mbytes/sec asynchronous transfer rate at each port and eleven operations/cycle throughput. In contrast, it possesses two identical external data and address buses supporting shared memory systems and high data rate, single-cycle transfers. It has separate internal program, data, and DMA coprocessor buses for support of massive concurrent I/O of program and data throughput, thereby maximising sustained CPU performance (Texas Instruments, 1991, Brown, 1991).

The T8 is a general purpose medium-grained 32-bit Inmos parallel PE with 25 MHz clock speed, yielding up to 20 MIPS performance, 4 kBytes on-chip RAM and is capable of 4.3 MFLOPS. The T8 is a RISC processor possessing an on-board 64-bit floating-point unit and four serial communication links. The links operate at a speed of 20 Mbits/sec achieving data rates of up to 1.7 MBytes/sec unidirectionally or 2.3 MBytes/sec bi-directionally. Most importantly, the links allow a single transputer to be used as a node among any number of similar devices to form a powerful PP system. The transputer thus provides an important bridge between single chip, real-time control and general purpose real-time computer control systems, and, in effect, removes the current distinction between the two (Irwin and Fleming, 1992; Transtech Parallel Systems Limited, 1991).

2.2 *Homogeneous architectures*

The homogeneous architectures considered include a network of C40s and a network of T8s. The homogeneous architecture of C40s comprises of a network of C40s resident on a Transtech TDM410 motherboard and a TMB08 motherboard incorporating a T8 as a root processor. A SUN SPARC station is used as host for downloading programs into the

network. The T8 possesses 1 MByte local memory and communicates with the TDM410 (C40s network) via link adapter using serial to parallel communication links. The C40s, on the other hand, communicate with each other via parallel communication links. Each C40 processor possesses 3 MBytes DRAM and 1 MByte SRAM. Figure 1 shows the topology of the network. The T8 root processor is used to provide an interface between the host and the first C40. The topology was chosen on the basis of the algorithm structure, which is simple to realise and well reflected as a linear farm.

The homogeneous architecture of T8s comprises of a network of T8s resident on a Transtech TMB08 motherboard. A SUN SPARC station is used as a host for downloading programs into the network. The T8s are all identical 32-bit with 25 MHz clock speed as mentioned earlier. The root T8 incorporates 2 MBytes local memory with the rest of the T8s each having 1 MByte. The serial links of the processors are used for communication with one another. Figure 2 shows the topology of the transputer network. The topology shown is utilised as it is simple to realise. Moreover, it reflects the structure of the algorithms considered in this study.

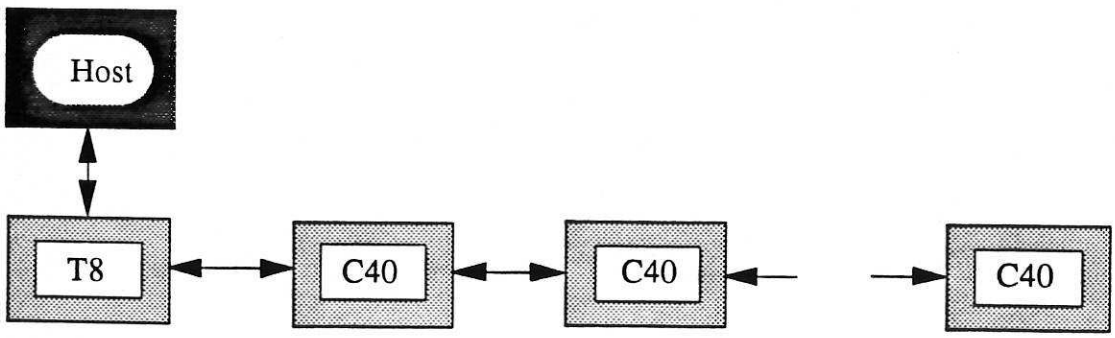

Figure 1: Operational configuration of the homogeneous architecture incorporating a network of C40s.

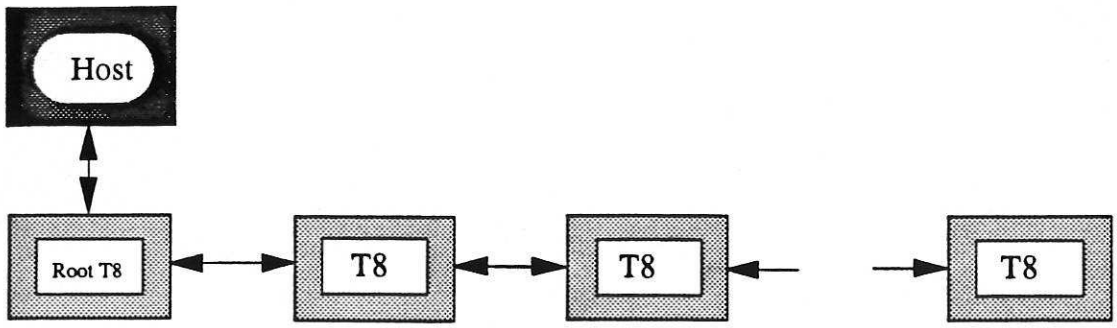

Figure 2: Operational configuration of the homogeneous architecture incorporating a network of T8s .

2.3 *Heterogeneous architectures*

Two heterogeneous parallel architectures, namely, an integrated i860 and T8 system and an integrated C40 and a T8 system, are considered in this study. The operational configuration of the i860+T8 architecture is shown in Figure 3. This comprises of an IBM compatible PC, A/D and D/A conversion facility, a TMB16 mother board and a TTM110 board incorporating a T8 and an i860. The TTM110 board also possesses 16 MBytes of shared memory accessible by both the i860 and the T8 and 4 MBytes of private memory accessible only by the T8. The i860 and the T8 processors are communicating with each other via this 16 MBytes shared memory. The interface between the i860 and the shared memory is highly optimised allowing data transmission rates of up to 160 MBytes/second. The T8 is used to boot the i860 by holding the i860 in reset whilst loading the memory with bootstrap code. Upon resetting the TRAM, the transputer is reset in the normal manner but the i860 is held in reset by the hardware, with HOLD asserted, releasing the bus to the transputer. The i860 code is then written to the boot address in the system RAM by the T8 before the i860 is released from reset by the transputer. The PC is used as a host for downloading the program into the T8 and the i860 under control of the TMB16 motherboard.

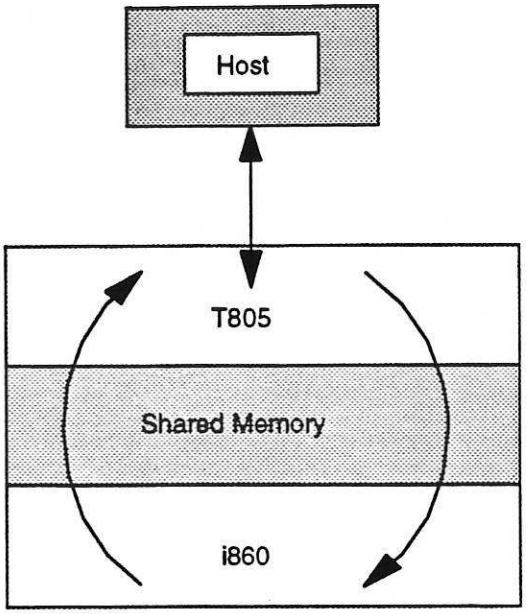

Figure 3: Operational configuration of the heterogeneous architecture incorporating an i860 and a T8.

The operational configuration of the C40+T8 architecture is shown in Figure 4. The T8 in the network is used both as the root processor providing interface with the host and as an active PE. The C40 and the T8 are communicating with each other via serial to parallel or parallel to serial links. A SUN SPARC station is used as a host for downloading programs into the system.

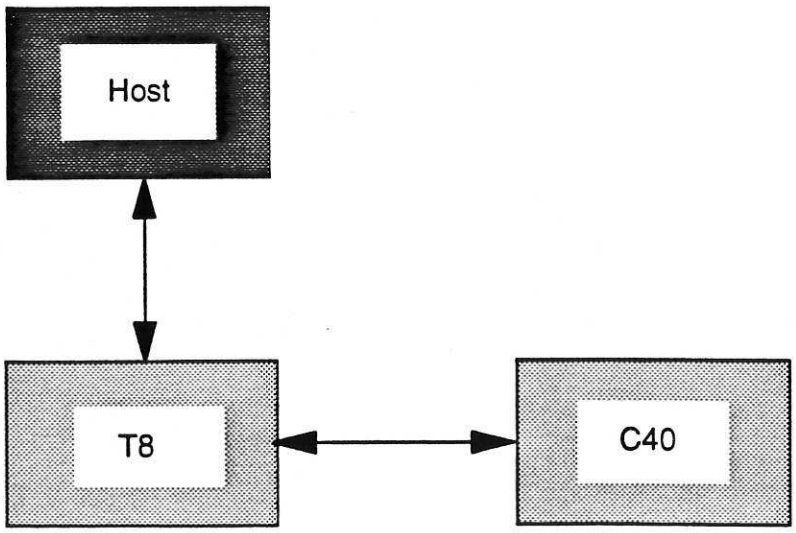

Figure 4: Operational configuration of the heterogeneous architecture incorporating a C40 and a T8.

3 Software

The development of efficient programs in high-level languages requires the necessary software support. In this context, compilers have a significant impact on the performance of the system. This means that some high-level languages have advantages in certain computational domains and some have advantages in other domains. The compiler itself is critical to the performance of the system as the mechanism and efficiency of taking a high-level description of the application and transforming it into a hardware dependent implementation differs from compiler to compiler. Identifying the foremost compiler for the application in hand is, therefore, especially challenging. The algorithms considered were coded in high-level languages consisting of ANSI C and 3L Parallel C as appropriate for the hardware used. Both programming languages support PP. Table 1 shows the compilers with the corresponding computing platforms used.

Table 1: Compilers used with the computing platforms.

i860	C40s	T8s	i860+T8	C40+T8
Portland Group ANSI C	3L Parallel C, V. 1.0.1	INMOS ANSI C	Portland Group ANSI C	3L Parallel C, V. 2.1 & V. 1.0.1

4 Algorithms

The algorithms considered in this investigation consist of FFT, second-order correlation, least mean square (LMS) and recursive least squares (RLS) adaptive filters, finite difference (FD) simulation, identification and AVC of a flexible beam structure and FD simulation of a flexible manipulator system. These are briefly described below.

4.1 Fast Fourier transform

Fast Fourier transform comprises a class of algorithms allowing efficient computation of discrete Fourier transform (DFT) of a sequence. Fast Fourier transform is commonly utilised in many applications, including spectral analysis, system identification and frequency response estimation. A real periodic discrete-time signal $x(n)$ of period N can

be expressed as a weighted sum of complex exponential sequences. Since sinusoidal sequences are unique only for discrete frequencies from 0 to 2π, the expansion contains only a finite number of complex exponentials. The complex DFT series $X(k)$ of a periodic discrete-time signal can be written as

$$X(k) = \sum_{n=0}^{N-1} x(n) W_N^{kn} \tag{1}$$

where, W_N is defined as

$$W_N = e^{-j2\pi/N} \tag{2}$$

Using the divide-and-conquer approach, equation (3) can be simplified as

$$X(p,q) = \sum_{l=0}^{L-1} \left\{ W_N^{lq} \left[\sum_{m=0}^{M-1} x(l,m) W_M^{mq} \right] \right\} W_L^{lp} \tag{3}$$

Equation (3) involves the computation of DFT of sequences of lengths M and L respectively. In this manner, the total computation will be half of that of a direct DFT computation (Ifeachor and Jervis, 1993; Proakis and Manolakis, 1988).

4.2 Cross-correlation

Correlation is commonly utilised in many signal processing applications, including spectral analysis, model structure validation and adaptive filtering. Cross-correlation is a measure of the similarity between two waveforms. Consider two signal sequences $x(n)$ and $y(n)$ each having finite energy. The cross-correlation of $x(n)$ and $y(n)$ is a sequence $r_{xy}(l)$, defined as

$$r_{xy}(l) = \sum_{n=-\alpha}^{\alpha} x(n) y(n-l); \quad l = 0, \pm 1, \ldots \tag{4}$$

or, equivalently, as

$$r_{xy}(l) = \sum_{n=-\alpha}^{\alpha} x(n+l) y(n); \quad l = 0, \pm 1, \ldots \tag{5}$$

The index l is the (time) shift (or lag) parameter and the subscripts xy on the cross-correlation sequence $r_{xy}(l)$ indicate that the sequences are being correlated. As shifting $x(n)$ to the left by l units relative to $y(n)$ is equivalent to shifting $y(n)$ to the right by l units relative to $x(n)$, the computations (4) and (5) yield identical cross-correlation sequences (Proakis and Manolakis, 1988).

4.3 Simulation and active vibration control of a flexible beam structure

Structural vibration in flexible systems is a common problem in many applications. These lead to performance deterioration and eventual physical damage of the structure. Active vibration control utilises the superposition of waves by generating cancelling signals to destructively interfere with undesirable vibrations and, thus, reduce the level of these vibrations. In contrast to passive methods which are bulky and un-economical at low frequencies, AVC is found to be efficient and effective for low-frequency vibration suppression. In this paper the implementation of an AVC mechanism within a simulated cantilever beam system is considered.

Consider a cantilever beam of length L, with a force $U(x,t)$ applied at a distance x from the fixed (clamped) end of the beam at time t resulting a deflection $y(x,t)$ of the beam from its stationary (unmoved) position at the point where the force has been applied. The motion of the beam in transverse vibration is governed by the well known fourth-order partial differential equation (PDE) (Tokhi and Hossain, 1994)

$$\mu^2 \frac{\partial^4 y(x,t)}{\partial x^4} + \frac{\partial^2 y(x,t)}{\partial t^2} = \frac{1}{m} U(x,t) \qquad (6)$$

where μ is a beam constant given by $\mu^2 = \dfrac{EI}{\rho A}$, with ρ, A, I and E representing the mass density, cross-sectional area, moment of inertia of the beam and the Young modulus respectively, and m is the mass of the beam. The corresponding boundary conditions at the fixed and free ends of the beam are given by

$$y(0,t) = 0 \quad \text{and} \quad \frac{\partial y(0,t)}{\partial x} = 0$$
$$\frac{\partial^2 y(L,t)}{\partial x^2} = 0 \quad \text{and} \quad \frac{\partial^3 y(L,t)}{\partial x^3} = 0 \tag{7}$$

Note that the model, thus, utilised incorporates no damping. To obtain a solution to the PDE, describing the beam motion, the partial derivative terms $\frac{\partial^4 y(x,t)}{\partial x^4}$ and $\frac{\partial^2 y(x,t)}{\partial t^2}$ in equation (6) and the boundary conditions in equation (7) are approximated using first order central FD approximations. This involves a discretisation of the beam into a finite number of equal-length sections (segments), each of length Δx, and considering the beam motion (deflection) for the end of each section at equally-spaced time steps of duration Δt. In this manner, let $y(x,t)$ be denoted by $y_{i,j}$ representing the beam deflection at point i at time step j. Let $y(x+v\Delta x, t+w\Delta t)$ be denoted by $y_{i+v,j+w}$, where v and w are non-negative integer numbers.

Using a first-order central FD method the partial derivatives $\frac{\partial^2 y}{\partial t^2}$ and $\frac{\partial^4 y}{\partial x^4}$ can be approximated as

$$\frac{\partial^2 y(x,t)}{\partial t^2} = \frac{y_{i,j+1} - 2y_{i,j} + y_{i,j-1}}{(\Delta t)^2}$$
$$\frac{\partial^4 y(x,t)}{\partial x^4} = \frac{y_{i+2,j} - 4y_{i+1,j} + 6y_{i,j} - 4y_{i-1,j} + y_{i-2,j}}{(\Delta x)^4} \tag{8}$$

Substituting for $\frac{\partial^2 y}{\partial t^2}$ and $\frac{\partial^4 y}{\partial x^4}$ from equation (8) into equation (6) and simplifying yields

$$y_{i,j+1} = 2y_{i,j} - y_{i,j-1} - \lambda^2\{y_{i+2,j} - 4y_{i+1,j} + 6y_{i,j} - 4y_{i-1,j} + y_{i-2,j}\} + \frac{(\Delta t)^2}{m}U(x,t) \tag{9}$$

where, $\lambda^2 = \frac{(\Delta t)^2}{(\Delta x)^4}\mu^2$. Similarly, descretising the boundary conditions in equation (7) and manipulating yields

$$y_{0,j} = 0 \quad \text{and} \quad y_{-1,j} = y_{1,j}$$
$$y_{n+1,j} = 2y_{n,j} - y_{n-1,j} \quad \text{and} \quad y_{n+2,j} = 2y_{n+1,j} - 2y_{n-1,j} + y_{n-2,j} \tag{11}$$

Equations (9) and (11) give the complete set of relations necessary for construction of the simulation algorithm characterising the behaviour of the beam. Substituting the discretised boundary conditions for the fixed and free ends from equations (11) into equation (9) yields the beam deflection at the grid points in a matrix form as

$$Y_{j+1} = -Y_{j-1} - \lambda^2 S Y_j + (\Delta t)^2 U(x,t)\frac{1}{m} \tag{12}$$

where,

$$Y_{j+1} = \begin{bmatrix} y_{1,j+1} \\ y_{2,j+1} \\ \vdots \\ y_{n,j+1} \end{bmatrix}, \quad Y_j = \begin{bmatrix} y_{1,j} \\ y_{2,j} \\ \vdots \\ y_{n,j} \end{bmatrix}, \quad Y_{j-1} = \begin{bmatrix} y_{1,j-1} \\ y_{2,j-1} \\ \vdots \\ y_{n,j-1} \end{bmatrix},$$

and S is a matrix given (for $n = 19$, say) as

$$S = \begin{bmatrix} a & -4 & 1 & 0 & 0 & 0 & \cdots & \cdots & 0 \\ -4 & b & -4 & 1 & 0 & 0 & \cdots & \cdots & 0 \\ 1 & -4 & b & -4 & 1 & 0 & \cdots & \cdots & 0 \\ 0 & 1 & -4 & b & -4 & 1 & \cdots & \cdots & 0 \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ \cdots & \cdots & \cdots & \cdots & 1 & -4 & b & -4 & 1 \\ \cdots & \cdots & \cdots & \cdots & 0 & 1 & -4 & c & -2 \\ \cdots & \cdots & \cdots & \cdots & 0 & 0 & 2 & -4 & d \end{bmatrix}$$

where, $a = 7 - \frac{7}{\lambda^2}$, $b = 6 - \frac{2}{\lambda^2}$, $c = 5 - \frac{2}{\lambda^2}$ and $d = 2 - \frac{2}{\lambda^2}$. Equation (12) is the required relation for the simulation algorithm, characterising the behaviour of the cantilever beam system, which can be implemented on a digital computer easily. For the algorithm to be stable it is required that the iterative scheme described in equation (12), for each grid point, converges to a solution. It has been shown that a necessary and sufficient condition for stability satisfying this convergence requirement is given by $0 < \lambda^2 \leq 0.25$ (Kourmoulis, 1990).

A schematic diagram of an AVC structure is shown in Figure 5. The unwanted (primary) disturbance is detected by a detection sensor, processed by a controller to

generate a cancelling (secondary/control) signal so that to achieve cancellation at the observation point. The objective in Figure 5 is to achieve total (optimum) vibration suppression at the observation point. Synthesising the controller on the basis of this objective yields (Tokhi and Hossain, 1994)

$$C = \left[1 - \frac{Q_1}{Q_0}\right]^{-1} \tag{13}$$

where, Q_0 and Q_1 represent the equivalent transfer functions of the system (with input at the detector and output at the observer) when the secondary source is off and on respectively. Equation (13) is the required controller design rule which can easily be implemented on-line on a digital processor. This leads to a self-tuning AVC algorithm comprising of the processes of identification and control. The process of identification involves obtaining Q_0 and Q_1 using a suitable system identification algorithm. The process of control, on the other hand, involves designing the controller according to equation (13) and implementing this in real-time.

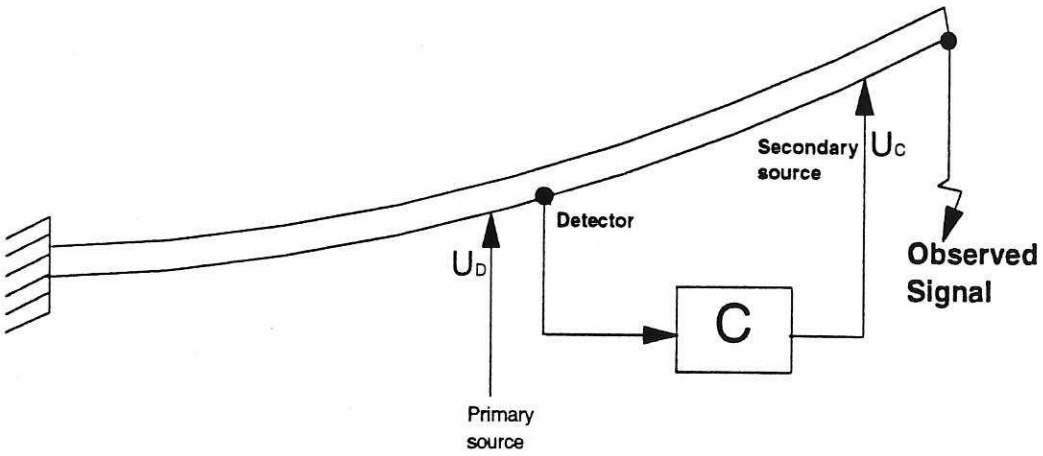

Figure 5: Schematic diagram of the AVC structure.

The identification algorithm is described here as the process of estimating parameters of the required controller characteristics. In this manner, it consists of the processes of estimating the system models Q_0 and Q_1 and the controller design calculation. The RLS algorithm is used here to estimate the system models Q_0 and Q_1 in the discrete-time

domain in parametric form. This is based on the well known least squares method briefly described below.

Let an unknown plant with input $u(n)$ and output $y(n)$ be described by a discrete linear model of order m as

$$y(n) = b_0u(n) + b_1u(n) + \ldots + b_mu(n-m) - a_1y(n-1) - \ldots - a_my(n-m)$$

or

$$y(n) = \Psi(n)\Theta(n) \tag{14}$$

where, Θ is the model parameter vector and Ψ, known as the observation matrix, is a row vector of the measured input/output signals. In this manner, the RLS estimation process at a time step k is described by

$$\varepsilon(k) = \Psi(k)\Theta(k-1) - y(k)$$

$$\Theta(k) = \Theta(k-1) - P(k-1)\Psi^T(k)\left[1 + \Psi(k)P(k-1)\Psi^T(k)\right]^{-1}\varepsilon(k) \tag{15}$$

$$P(k) = P(k-1) - P(k-1)\Psi^T(k)\left[1 + \Psi(k)P(k-1)\Psi^T(k)\right]^{-1}\Psi(k)P(k-1)$$

where, $P(k)$ is the covariance matrix. Thus, the RLS estimation process is to implement and execute the relations in equation (15) in the order given. The performance of the estimator can be monitored by observing the parameter set at each iteration. Once convergence has been achieved the routine can be stopped. The convergence is determined by the magnitude of the modelling error $\varepsilon(k)$ or by the estimated set of parameters reaching a steady level (Tokhi and Leitch, 1992).

The process of calculation of parameters of the controller uses a set of design rules based on equation (13). Let the system models Q_0 and Q_1 be described as

$$Q_0 = \frac{b_{00} + b_{01}z^{-1} + b_{02}z^{-2}}{1 + a_{01}z^{-1} + a_{02}z^{-2}} \quad , \quad Q_1 = \frac{b_{10} + b_{11}z^{-1} + b_{12}z^{-2}}{1 + a_{11}z^{-1} + a_{12}z^{-2}} \tag{16}$$

Substituting for Q_0 and Q_1 from equation (16) into equation (13) and simplifying yields the required controller transfer function as

$$C = \frac{b_{C0} + b_{C1}z^{-1} + b_{C2}z^{-2} + b_{C3}z^{-3} + b_{C4}z^{-4}}{1 + a_{C1}z^{-1} + a_{C2}z^{-2} + a_{C3}z^{-3} + a_{C4}z^{-4}} \tag{17}$$

where,

$$\begin{aligned}
&b_{C0}(b_{00} - b_{10}) = b_{00}, \\
&b_{C1}(b_{00} - b_{10}) = b_{01} + b_{00}a_{11}, && a_{C1}(b_{00} - b_{10}) = b_{01} + b_{00}a_{11} - b_{10}a_{01} - b_{11}, \\
&b_{C2}(b_{00} - b_{10}) = b_{02} + b_{01}a_{11} + b_{00}a_{12}, && a_{C2}(b_{00} - b_{10}) = b_{02} + b_{01}a_{11} + b_{00}a_{12} - b_{10}a_{02} - b_{11}a_{01} - b_{12}, \\
&b_{C3}(b_{00} - b_{10}) = b_{02}a_{11} + b_{01}a_{12}, && a_{C3}(b_{00} - b_{10}) = b_{02}a_{11} + b_{01}a_{12} - b_{11}a_{02} - b_{12}a_{01}, \\
&b_{C4}(b_{00} - b_{10}) = b_{02}a_{12}, && a_{C4}(b_{00} - b_{10}) = b_{02}a_{12} - b_{12}a_{02}.
\end{aligned} \tag{18}$$

This gives the set of design rules for calculation of the required controller parameters.

The control algorithm, as considered here, consists of the process of on-line implementation of the controller to generate the control signal. This involves the implementation of the controller, as designed through the identification algorithm above, in discrete form using the equivalent difference equation formulation as

$$y(n) = \sum_{i=0}^{4} b_{Ci}u(n-i) - \sum_{j=1}^{4} a_{Cj}y(n-j) \tag{19}$$

where, $u(n)$ and $y(n)$ in equation (19) correspond to the discrete input and output signals of the controller. Note that in implementing equation (19) within the simulation environment, the simulation algorithm becomes an integral part of the process. Thus, the control algorithm consists of the combined implementation of equation (19) and the beam simulation algorithm.

4.4 Simulation of a flexible manipulator system

A schematic representation of the single-link flexible manipulator under consideration is shown in Figure 6. A control torque τ is applied at the pinned end (hub) of the arm by an actuator motor. θ represents the hub angle, POQ is the original co-ordinate system while P'OQ' is the co-ordinate system after an angular rotation θ. I_h is the hub inertia, I_p is the inertia associated with a payload of mass M_p and u is the elastic deflection of the arm at a

distance x from the hub. The dynamic equation of the flexible manipulator, considered as an Euler-Bernoulli beam equation, can be expressed as

$$\rho\frac{\partial^2 y(x,t)}{\partial t^2} + EI\frac{\partial^4 y(x,t)}{\partial x^4} = \tau(x,t) \tag{20}$$

where, $y(x,t)$ is the manipulator displacement (deflection) at a distance x from the hub of the manipulator at time t , ρ is the density per unit length of the manipulator material, E is the Young modulus, I is the second moment of inertia, $\tau(x,t)$ is the applied torque and EI represents the flexural rigidity of the manipulator.

The boundary conditions at the hub end are given by

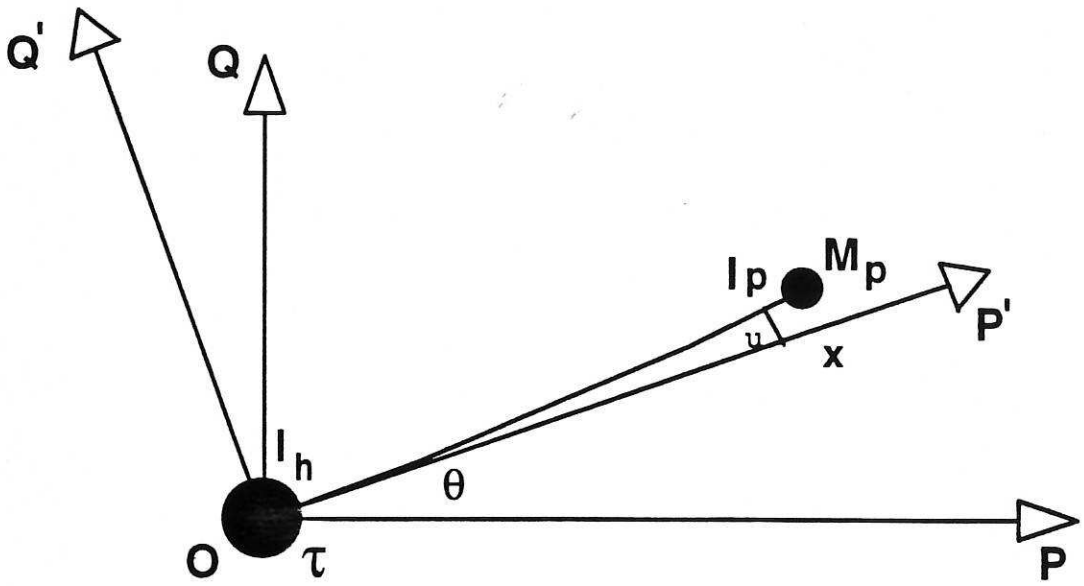

Figure 6: Schematic representation of the flexible manipulator system.

$$y(0,t) = 0 \quad , \quad I_h\frac{\partial^3 y(0,t)}{\partial t^2 \partial x} - EI\frac{\partial^2 y(0,t)}{\partial x^2} = \tau(t) \tag{21}$$

where, $\tau(t)$ is the torque applied at the manipulator hub. Similarly, the boundary conditions at the tip (end-point) of the manipulator are given by

$$M_p \frac{\partial^2 y(L,t)}{\partial t^2} - EI \frac{\partial^3 y(L,t)}{\partial x^3} = 0$$
$$I_p \frac{\partial^3 y(L,t)}{\partial t^2 \partial x} + EI \frac{\partial^2 y(L,t)}{\partial x^2} = 0 \qquad (22)$$

where, L is the length of the manipulator. The initial conditions along the t co-ordinate are given as

$$y(0,t) = 0 \quad \text{and} \quad \frac{\partial y(x,0)}{\partial x} = 0 \qquad (23)$$

The above relations describe the state of behaviour of the flexible manipulator system which can be used to construct a simulation environment of the system.

To solve the PDE in equation (20), it is replaced by a set of difference equations defined by the central difference quotients of the FD method (Azad, 1994). The manipulator length and movement time are each divided into suitable number of sections of equal length represented by Δx $(x = i\Delta x)$ and Δt $(t = j\Delta t)$ respectively. A difference equation, corresponding to each point of the grid is, thus, developed. The displacement, $y_{i,j+1}$, of section i of the manipulator at time step $j+1$ can, thus, be obtained as

$$y_{i,j+1} = -c\left[y_{i-2,j} + y_{i+2,j}\right] + b\left[y_{i-1,j} + y_{i+1,j}\right] + a y_{i,j} - y_{i,j-1} + \frac{\Delta t^2}{\rho}\tau(i,j) \qquad (24)$$

where, $a = 2 - \dfrac{6\Delta t^2 EI}{\rho \Delta x^4}$, $b = \dfrac{4\Delta t^2 EI}{\rho \Delta x^4}$ and $c = \dfrac{\Delta t^2 EI}{\rho \Delta x^4}$. Using matrix notation, equation (24) can be written as

$$y_{i,j+1} = A y_{i,j} - y_{i,j-1} + BF \qquad (25)$$

where,

$$y_{i,j+1} = \begin{bmatrix} y_{1,j+1} \\ y_{2,j+1} \\ \vdots \\ y_{n,j+1} \end{bmatrix}, \quad y_{i,j} = \begin{bmatrix} y_{1,j} \\ y_{2,j} \\ \vdots \\ y_{n,j} \end{bmatrix}, \quad y_{i,j-1} = \begin{bmatrix} y_{1,j-1} \\ y_{2,j-1} \\ \vdots \\ y_{n,j-1} \end{bmatrix},$$

$$A = \begin{bmatrix} m_1 & m_2 & m_3 & 0 & 0 & \dots & 0 & 0 & 0 & 0 & 0 \\ b & a & -b & -c & 0 & \dots & 0 & 0 & 0 & 0 & 0 \\ -c & b & a & b & -c & \dots & 0 & 0 & 0 & 0 & 0 \\ \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots \\ 0 & 0 & 0 & 0 & 0 & \dots & -c & b & a & b & -c \\ 0 & 0 & 0 & 0 & 0 & \dots & 0 & m_{11} & m_{12} & m_{13} & m_{14} \\ 0 & 0 & 0 & 0 & 0 & \dots & 0 & m_{21} & m_{22} & m_{23} & m_{24} \end{bmatrix},$$

$$F = \begin{bmatrix} \tau(i,j) \\ 0 \\ \vdots \\ 0 \end{bmatrix}, \quad B = \frac{\Delta t^2}{\rho}$$

Equation (25) is the general solution of the PDE giving the displacement of section i of the manipulator at time step $j+1$.

It follows from equation (24) that, to obtain the displacements $y_{1,j+1}$, $y_{n-1,j+1}$ and $y_{n.j+1}$ the displacements of the fictitious pints $y_{-1,j}$, $y_{n+1,j}$ and $y_{n+2,j}$ are required. The estimation of these displacements is based on the boundary and initial conditions related to the dynamic equation of the flexible manipulator which in turn determine the values of m_1 to m_3 and m_{11} to m_{24} in matrix A of equation (25).

4.5 The LMS filter

The LMS algorithm is one of the most successful adaptive algorithms developed by Windrow and his co-workers (Widrow et. al, 1975). It is based on the steepest descent method where the weight vector is updated according to

$$\mathbf{W}_{k+1} = \mathbf{W}_k - 2e_k\mu\mathbf{X}_k \tag{26}$$

where $\mathbf{W}_k$ and $\mathbf{X}_k$ are the weight and the input signal vectors at time step k respectively, μ is a constant controlling the stability and rate of convergence and e_k is the error given by

$$e_k = y_k - \mathbf{W}_k^T\mathbf{X}_k \tag{27}$$

where, y_k is the current contaminated signal sample. It is clear from the above that the LMS algorithm does not require prior knowledge of the signal statistics. The weights obtained by the LMS algorithm are not only estimates, but these are adjusted so that the filter learns the characteristics of the signals leading to a convergence of the weights. The condition for convergence is given by

$$0 < \mu > 1/\lambda_{max} \tag{28}$$

where, λ_{max} is the maximum eigenvalue of the input data covariance matrix.

4.6 The RLS filter

The RLS algorithm is based on the well known least squares method. An output signal $y(k)$ of the filter is measured at the discrete time k, in response to a set of input signals $x(k)$ (Tokhi and Leitch, 1992). The error variable is given by

$$\varepsilon(k) = \Psi(k)\Theta(k-1) - y(k) \tag{29}$$

where Θ and Ψ represent the parameter vector and the observation matrix of the filter respectively. These are given by

$$\Theta^T = [\theta(1), \quad \theta(2), \quad ..., \quad \theta(m)]$$

$$\Psi = [\psi(1), \quad \psi(2), \quad ..., \quad \psi(m)]$$

where m represents the order and ψ the input sample of the filter. The new parameter vector is given by

$$\Theta(k) = \Theta(k-1) - \mathbf{P}(k-1)\Psi^T(k)\left[1 + \Psi(k)\mathbf{P}(k-1)\Psi^T(k)\right]^{-1}\varepsilon(k) \tag{30}$$

with $\mathbf{P}(k)$, representing the covariance matrix at time step k, given by

$$\mathbf{P}(k) = \mathbf{P}(k-1) - \mathbf{P}(k-1)\Psi^T(k)\left[1 + \Psi(k)\mathbf{P}(k-1)\Psi^T(k)\right]^{-1}\Psi(k)\mathbf{P}(k-1) \tag{31}$$

The performance of the filter can be monitored by observing the error variable $\varepsilon(k)$ at each iteration.

5 Practical issues in parallel real-time processing

Application goals of PP for implementing real-time signal processing and control algorithms must satisfy critical time constraints associated with sampling intervals. When contemplating the implementation of algorithms and associated software on PP systems, it is essential to organise the algorithm to realise the maximum benefits of parallelism. This has several associated problems as discussed below.

5.1 Performance evaluation

For PP with widely different architectures and different PEs, performance measurements such as MIPS and MFLOPS of the PEs are meaningless. Of more importance is to rate the performance of each architecture with its PEs on the type of program likely to be encountered in a typical application. The different architectures and their different clock rates, memory cycle times of the PEs, inter-processor communication speed, optimisation facility and compiler performance etc. all confuse the issue of attempting to rate the architecture. This is an inherent difficulty in selecting a parallel architecture, for better performance, for algorithms in signal processing and control system development applications. The ideal performance of a parallel architecture demands a perfect match between the capability of the architecture and the program behaviour. Capability of the architecture can be enhanced with better hardware technology, innovative architectural features and efficient resources management. In contrast, program behaviour is difficult to predict due to its heavy dependence on application and run-time conditions. Moreover, there are many other factors that influence program behaviour. These include algorithm design, partitioning and mapping of an algorithm, inter-processor communication, data structures, language efficiency, programmer skill, and compiler technology (Hwang, 1993; Ching and Wu, 1989, Cvetanvoic, 1987).

Parallelism is beneficial when it successfully yields higher performance with reasonable hardware and software cost. Whether parallelism is worthwhile depends on the application. Two extreme situations, however, may be distinguished;

(i) For relatively simple, high volume applications, minimising manufacturing cost is critical. In this case it may be best to invest development effort in optimising algorithm and code efficiency to achieve a single-processor solution.

(ii) For relatively complex, low volume applications, minimising development cost is important. In this case it may be best to preserve the structure of the high level code and introduce additional PEs.

There will always be many applications which are satisfied by a uni-processor implementation. Before adopting a PP solution, it must be clear that it possesses some feature that cannot be provided by a single processor. Through the investigations carried out in this study and from practical experiences, a chart of performance criteria made is shown in Figure 7. This indicates the main components that play important role in the performance of an architecture.

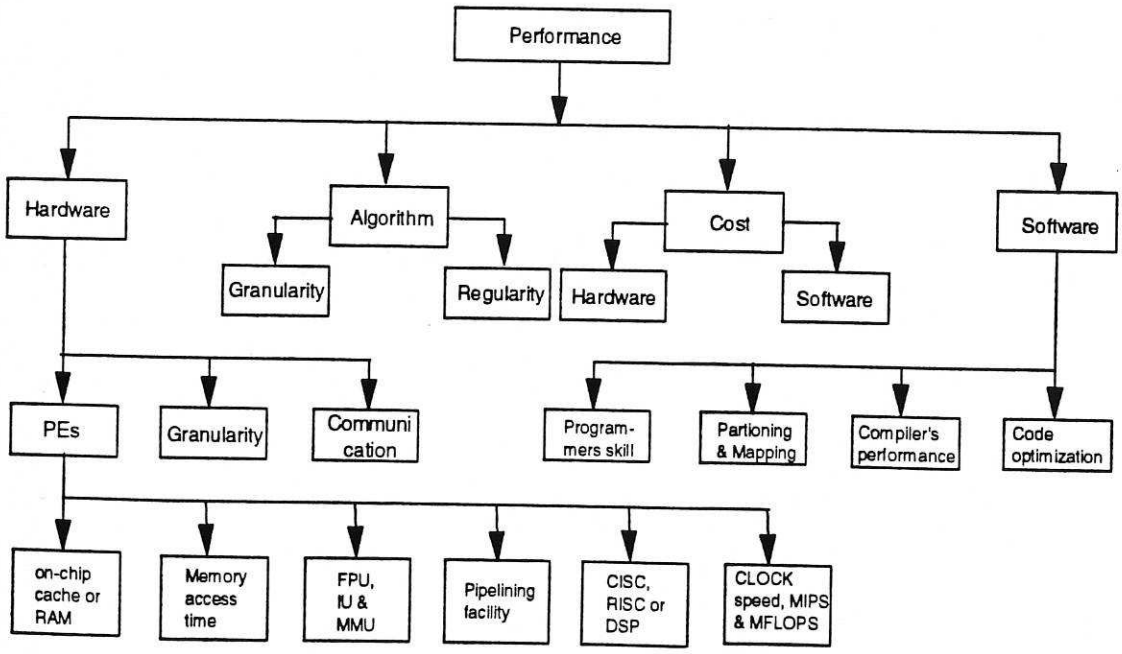

Figure 7: Factors influencing performance evaluation of architectures.

The most widely accepted measure used to evaluate the performance of a parallel system is speedup. Speedup (S_N) is defined as the ratio of the execution time (T_1) on a single processor to the execution time (T_N) on N processors;

$$S_N = \frac{T_1}{T_N} \tag{32}$$

The theoretical maximum speed that can be achieved with a parallel architecture of N identical processors working concurrently on a problem is N. This is known as the ideal speedup. In practice, the speedup is much less, since some processors are ideal at a given time due to conflicts over memory access, communication delays, algorithm inefficiency and mapping for exploiting the natural concurrency in a computing problem (Hwang and Briggs, 1985). But, in some cases, the speedup can be obtained above the ideal speedup, due to anomalies in programming, compilation, architecture usage, etc. For example, a single processor system may store all its data off-chip, whereas the multiprocessor system may store all its data on-chip leading to an unpredicted increase in performance.

Another useful measure in evaluating the performance of a parallel system is efficiency (E_N). This can be defined as

$$E_N = \frac{S_N}{N} \times 100\% = \frac{T_1}{NT_N} \times 100\% \tag{33}$$

Efficiency can be interpreted as providing an indication of the average utilisation of the 'N' processors, expressed as a percentage. Furthermore, this measure allows a uniform comparison of the various speedups obtained from systems containing different number of processors. It has also been illustrated that the value of efficiency is directly related to the granularity of the system (Stone, 1987). For example, consider an ideal problem which can be partitioned into 'N' equal subtasks requiring 'R' units of time, with associated communication overhead of 'C' units. Thus, the ideal system for efficiency can be defined as

$$E_N = \frac{R/C}{1+(R/C)} \times 100\% = \frac{Granularity}{1+Granularity} \times 100\% \tag{34}$$

Although, this analysis uses an ideal model, this value of granularity is used as a guideline during the partitioning process as described later.

5.2 *Inter-processor communication*

Inter-processor communication between PEs is one of the important issues on which the real-time performance of a number of parallel architectures can be compared and suitability of an algorithm evaluated. When several processors are required to work co-operatively on a single task, one expects frequent exchange of data among the several subtasks that comprise the main task. The amount of data, the frequency with which they are transmitted, the speed of their transmission, and the route that they take are all significant in affecting the inter-communication within the architecture. The first two factors depend on the algorithm itself and how well it has been partitioned. The remaining two factors depend on the hardware and are the points of discussion here for investigation. These, further, depend on the inter-connection strategy, whether tightly coupled or loosely coupled. Any evaluation of the performance of the inter-connection must be, to a certain extent, quantitative. However, once a few candidate networks have been tentatively selected, detailed (and expensive) evaluation including simulation can be carried out and the best one selected for the proposed application (Agrawal et. al, 1986). To explore the real-time performance of the architectures, investigations into inter-processor communication were carried out with all the four different parallel architectures utilised in this investigation. These inter-processor communication techniques for the different architectures are

(i) C40 to C40: parallel communication link,

(ii) T8 to T8: serial communication link,

(iii) T8 to i860: shared memory communication, and

(iv) T8 to C40: serial to parallel communication link.

The performance of these inter-processor communication links are evaluated by utilising a similar strategy for exactly the same data block without any computation during the communication time.

5.3 Partitioning and mapping

There are three different problems to be considered in implementing signal processing and control algorithms on PP systems;

(a) identifying parallelism in the algorithm,

(b) partitioning the algorithm into subtasks, and

(c) allocating the tasks to processors.

These include inter-processor communication, issues of granularity of the algorithm and the hardware and regularity of the algorithm. Hardware granularity is a ratio of computational performance over communication performance of each processor within the architecture. Similarly, task granularity is the ratio of computational demand over the communication demand of the task. Performance benefits of parallel architectures strongly depend on these ratios (Maguire, 1991; Stone, 1987). These ratios reveal the amount of communication overhead associated with each unit of computation. When the ratio is very low, it becomes ineffective to use parallelism. When the ratios are very high, parallelism is potentially profitable. Typically a high compute/communication ratio is desirable. The concept of task granularity can also be viewed in terms of compute time per task. When this is large, it is a coarse-grain task implementation. When it is small, it is a fine-grain task implementation. Although, large grains may ignore potential parallelism, partitioning a problem into the finest possible granularity does not necessarily lead to the fastest solution, as maximum parallelism also has the maximum overhead, particularly due to increased communication requirements. Therefore, when partitioning the application across PEs, it is essential to choose an algorithm granularity that balances useful parallel computation against communication and other overheads (Nocetti and Fleming, 1991).

Regularity is a term used to describe the degree of uniformity in the execution thread of the computation. Many algorithms can be expressed by matrix computations. This leads to the so called regular iterative (RI) type of algorithms due to their very regular structure. In implementing these type of algorithms, a vector processor will, principally, be expected

to perform better. Moreover, if a large amount of data is to be handled for computation in these type algorithms, the performance will further be enhanced if the processor has more internal data cache, instruction cache and/or a built-in math coprocessor. In implementing these algorithms on a PP platform, the tasks could be distributed uniformly among the PEs. However, this may require a large amount of communication between the processors and, therefore, can be a detriment to the performance of the computing platform in both homogeneous and heterogeneous architectures.

There are two main approaches to allocating tasks to processors: statically and dynamically. In static allocation, the association of a group of tasks with a processor is resolved before running time and remains fixed throughout the execution, whereas in dynamic allocation, tasks are allocated to processors at running time according to certain criteria, such as processor availability, inter-task dependencies and task priorities. Whatever method is used, a clear appreciation is required of the overheads and parallelism /communication trade-off as mentioned earlier. Dynamic allocation offers greater potential for optimum processor utilisation, but it also incurs a performance penalty associated with scheduling software overheads and increased communication requirements which may prove unacceptable in some real-time applications.

To investigate the performance of the computing platforms in the real-time implementation of the algorithms considered in this study, the features discussed above are considered in the process of partitioning the algorithms so that the capabilities of the parallel hardware platforms are efficiently exploited and the load distribution is balanced so that to minimise inter-processor communications. The PEs computing performance and inter-processor communication, compilers performance and optimisation facility are verified to implement static task allocation for best performance.

6 Implementations and results

To investigate the real-time performance of the computing platforms in implementing the algorithms, performance of inter-processor communication links were also looked at.

Moreover, to allow suitable load distribution across PEs, compare performance and measure the speedups achieved with the parallel architectures, the performance of the individual PEs in implementing the algorithms were also investigated. Results of these investigations are presented and discussed in this section.

6.1 Inter-processor communication

To investigate performance of the inter-processor communication links, a 4000 points floating type data was used. The communication time in sending the data from one processor to another and receiving it back was measured with the various communication links involved in the parallel architectures. In case of the C40 to T8 and the C40 to C40 communication, the speed of single lines of communication were also measured by using bi-directional data transmission in each of the 4000 iterations. This was realised by altering the direction of data transmission for sending and receiving data at each iteration. Figure 8 shows the performance of inter-processor communication links. It is noted that among these the C40-C40 double-line parallel communication is the fastest, whereas the T8-C40 single-line serial to parallel communication is the slowest. Among the double-line communications, the C40-C40 link is found to be 10 times faster than the T8-T8 serial communication, 14.89 times faster than the T8-i860 shared memory communication and 17.56 times faster than the T8-C40 serial to parallel communication. Among the single-line communications, on the other hand, the C40-C40 parallel communication is found to be 1.23 times faster than the T8-C40 serial to parallel communication. It is evidenced that as compared to serial communication, parallel communication offers substantial advantage. In shared memory communication additional time is required in accessing and/or writing into the shared memory. In serial to parallel communication, on the other hand, additional penalty is paid during transformation of data from serial to parallel and vice versa. Comparing the C40-C40 double-line and single-line parallel communications, reveals that the double-line communication is 93.94 times faster than the single-line communication. Similarly, the T8-C40 double-line is performing 6.58 times faster than the corresponding

single link communication. It follows from this that a substantial proportion of the communication time is spent in altering the direction of data transmission in a single-line bi-directional communication as compared to an equivalent double-line communication.

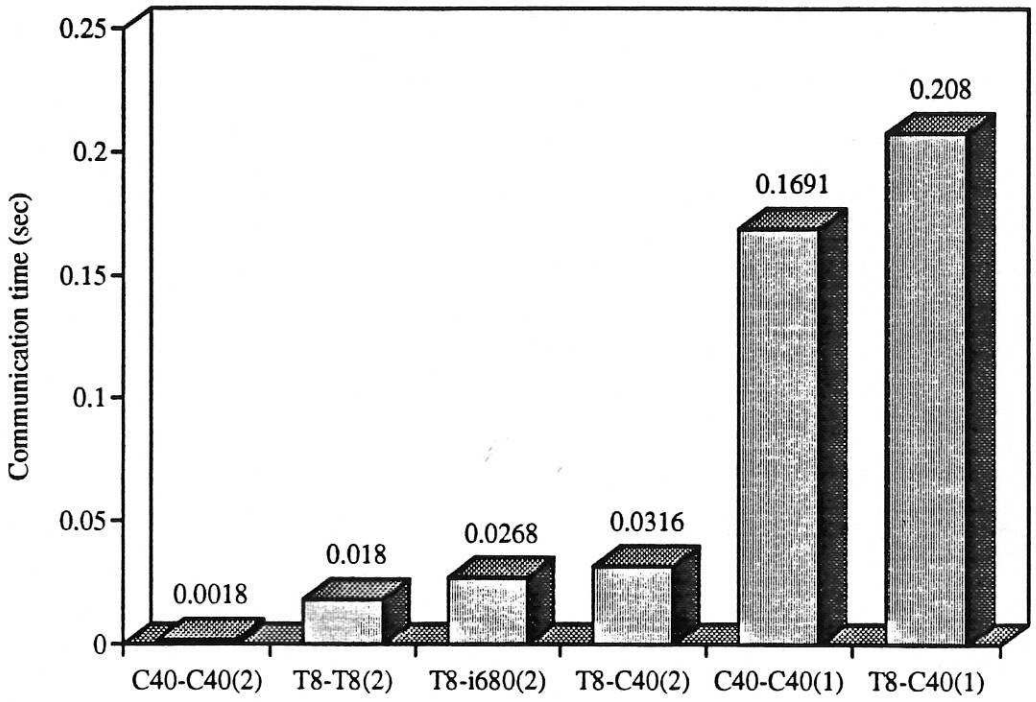

Figure 8: Speed of inter-processor communication links; (1) single-line, (2) double-line.

6.2 *The FFT algorithm*

Figure 9 shows the real-time performance of the computing platforms in implementing a 512-point FFT algorithm. As noted earlier, the FFT algorithm is a regular DSP process and highly matrix based. This is evidenced in the performance of the i860, with its vector processing resources, achieving the shortest execution time among the uni-processor architectures and the i860+T8 achieving the shortest execution time among the parallel architectures. The C40 and T8, not having such resources, are performing 32.34 and 78 times slower than the i860 respectively. Note that, although, the i860+T8 has performed 18.64, 30.54 and 53.54 times faster than the C40+C40, C40+T8 and T8+T8 respectively, its performance is 1.4 times slower than the i860. This is mainly attributed to the

performance of the T8. The serial to parallel communication link is, additionally, an influential factor in the performance of the C40+T8 in comparison to that of the C40+C40. Note further that, although, the C40 and the C40+C40 are performing faster than the T8 and T8+T8 architectures respectively, the speedup achieved with C40+C40 as compared to a single C40 is 1.24 whereas the speedup with T8+T8 as compared to a single T8 is 1.04. These speedups are roughly of a similar scale. The disparity between the two is mainly due to the inter-processor communication links in the two cases. However, in either case the performances of the C40 and T8 based architectures are not highly impressive due to a mismatch between the requirements of the algorithm and the computing resources of the architectures.

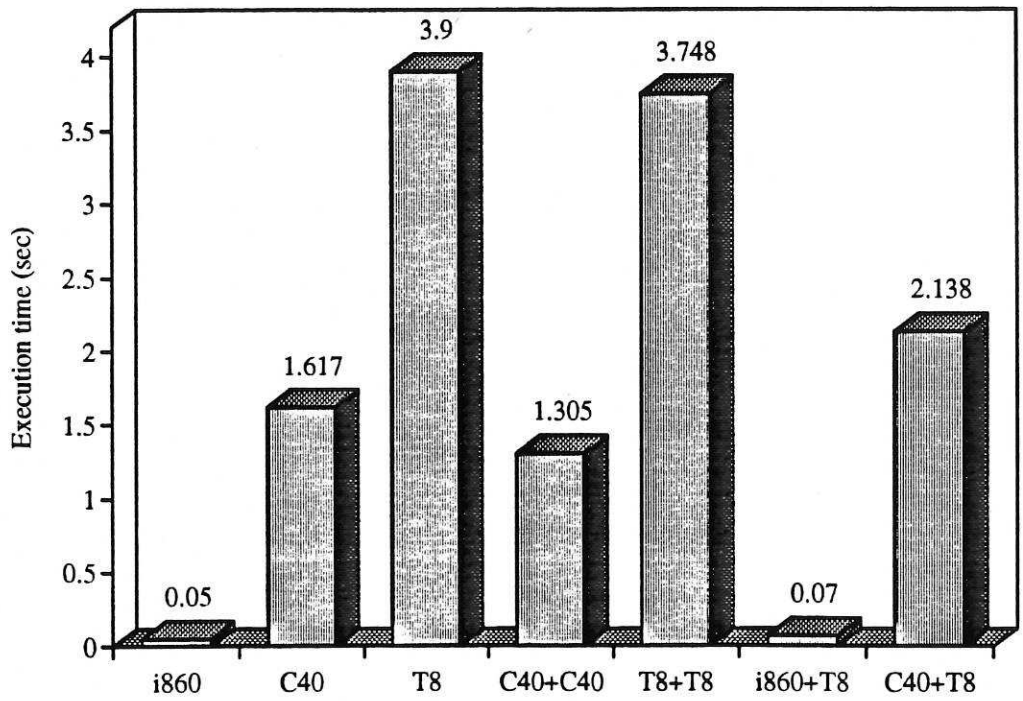

Figure 9: Execution times of the computing platforms in implementing the FFT algorithm.

6.3 The cross-correlation algorithm

To investigate the real-time implementation of the correlation algorithm, two waveforms, each of 1000 samples, were used. Figure 10 shows the real-time performance of the

computing platforms in implementing the correlation algorithm. Note that this algorithm is of the RI type for which the vector processing resources of the i860 are exploited achieving the shortest execution time among the uni-processor architectures. This is further evidenced in the performance of the i860+T8 among the parallel architectures. It is noted that the C40 and the T8 have performed 2.786 and 7.794 times slower than the i860. The speedup with two C40s as compared to a single C40 is 2.39. This super-linear speedup is due to reduction in the program, leading to less run-time memory management, suitable for the limited internal cache and memory of the C40. The speedup with two T8s as compared to a single T8 is only 1.25. This is due to the regular DSP nature of the algorithm which is not as suitable for the T8 as the C40. It follows from this that the speedup achieved with the i860+T8 as compared to either the i860 or the T8 is attributed, mainly, to the performance of the i860. Similarly, the speedup achieved with the C40+T8 as compared to either the C40 or the T8 is attributed, mainly, to the performance of the C40.

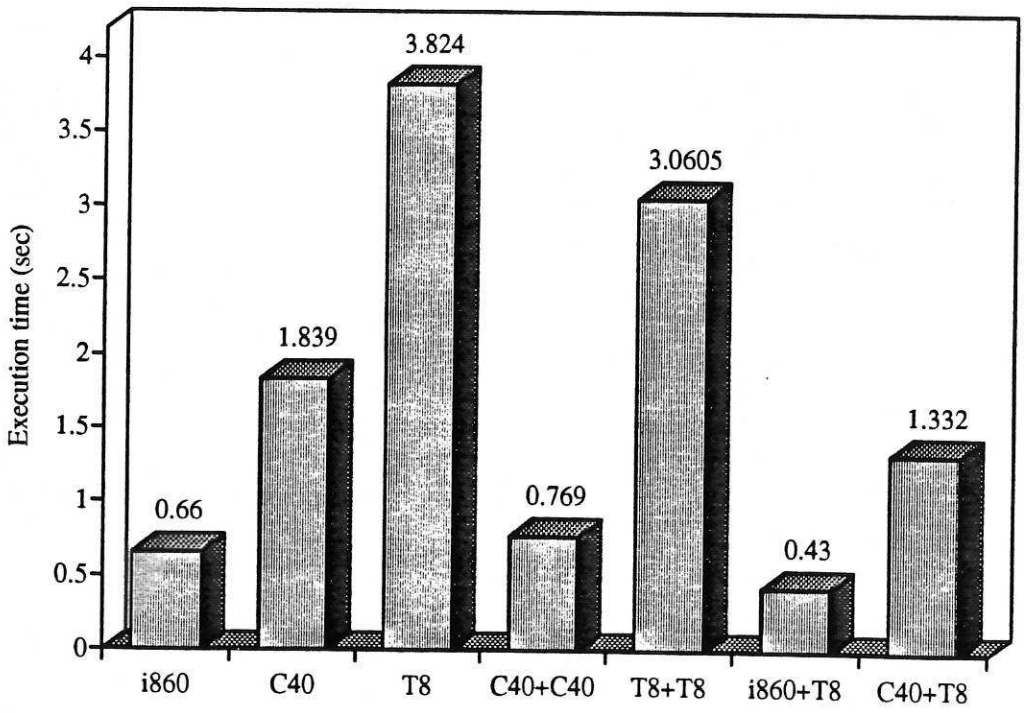

Figure 10: Execution times of the computing platforms in implementing the correlation algorithm.

6.4 Flexible beam simulation, identification and control

6.4.1 Simulation

Figure 11 shows the execution times achieved by the architectures, in implementing the simulation algorithm over 20 000 iterations. The simulation algorithm, as discussed earlier, is mainly of a matrix based computational type for which the powerful vector processing resources of the i860 are exploited and utilised to achieve the shortest execution time among the uni-processor architectures and with the i860+T8 among the parallel architectures. The C40 and the T8 do not have such vector processing resources making them 6.053 and 9.864 times slower than the i860 respectively. This implies that the C40 and the T8 are not performing well in situations where the algorithm is matrix type and extensive run time memory management is involved. The speedup achieved with two C40s as compared to a single C40 is only 1.42. Although, the program has reduced to a half for a single C40, due to the nature of the algorithm the C40+C40 has not achieved a performance better than a single i860 vector processor. The speedup achieved with two T8s as compared to a single T8 is 1.23. This speedup is similar to that achieved with two C40s in comparison to a single C40, although a serial communication link is utilised in case of the T8s. Thus, the main factor influencing the C40+T8 to perform slightly slower than the C40+C40 is the serial to parallel communication link utilised and the slower T8 processor incorporated in this architecture. Due to the shared memory communication overhead and incorporation of the slower T8 processor, the i860+T8 has achieved longer execution time as compared to a single i860.

6.4.2 Identification

As discussed earlier, the identification algorithm is composed of two components of similar nature and length, while estimating parameters of Q_0 and Q_1, and a process of controller design calculation. In case of the uni-processor architectures the algorithm was implemented sequentially. In case of the parallel architectures, on the other hand, the algorithm was partitioned so that the load at estimating parameters of Q_0 and Q_1 is equally

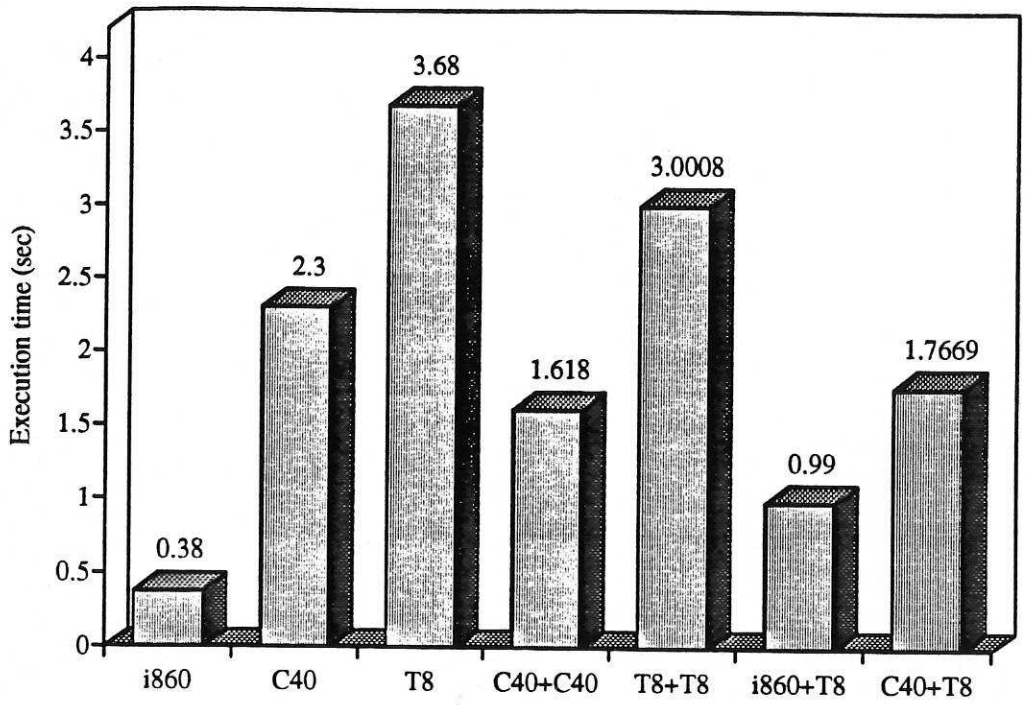

Figure 11: Execution times of the computing platforms in implementing the beam simulation algorithm.

distributed among the two PEs with one of the PEs further carrying out calculation of parameters of the controller. Moreover, in this process, limited communication, due to parameters of Q_0 and Q_1 for calculation of controller parameters, is required between the two PEs implementing the algorithm. Figure 12 shows the execution times of the computing platforms in implementing the identification algorithm over 1000 iterations using second order models for Q_0 and Q_1 . It is noted that the C40 and the C40+C40 architectures have performed as the fastest among the uni-processor and parallel architectures respectively. However, the speedup achieved with two C40s as compared to a single C40 is only 1.35. This could be due to the nature of the identification algorithm for which the pipelining nature of the C40 DSP device is not exploited much, even after a reduction of the program into two segments. The algorithm does incorporate some matrix manipulation. However, as a result of the irregular nature of the algorithm, the i860 is found to have performed slower than the C40. This is further noted in the performance of the i860+T8 as compared to that of the C40+T8. In contrast, the T8 is performing

significantly well; the speedup achieved with two T8s as compared to a single T8 is 2.22. This super-linear speedup results due to significant reduction in data handling for which the available internal memory of the T8 is sufficient, thus, reducing run-time memory management load. It follows from this that the speedup achieved with the i860+T8 in comparison to either a single i860 or a single T8 can mainly be attributed to the performance of the T8.

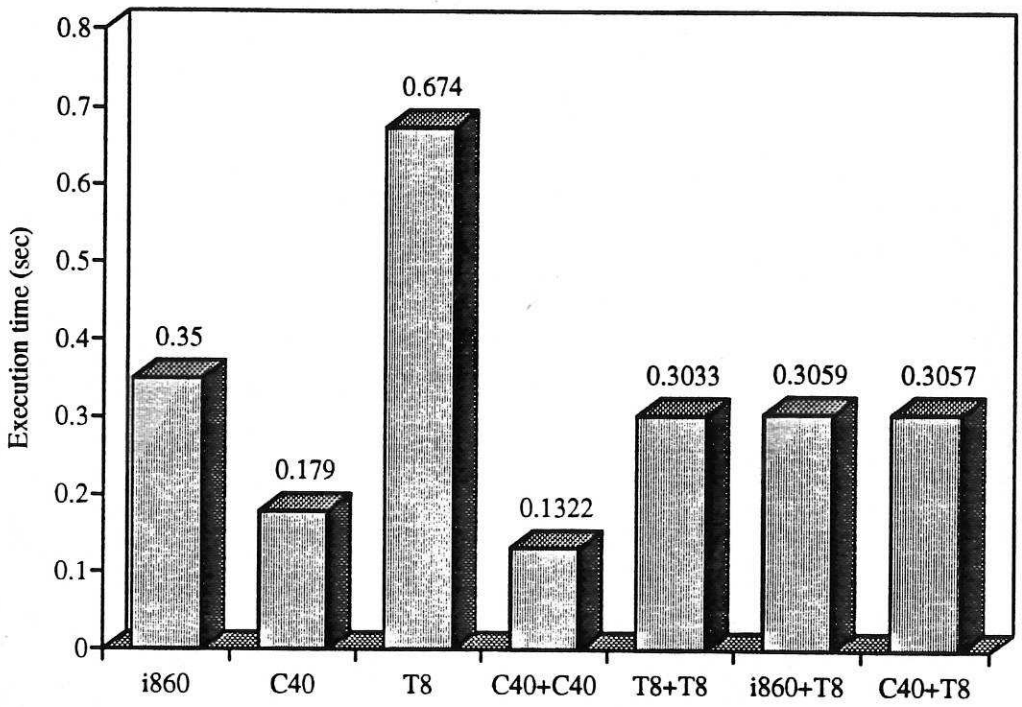

Figure 12: Execution times of the computing platforms in implementing the beam identification algorithm.

6.4.3 Control

Figure 13 shows the execution times achieved by the computing platforms in implementing the control algorithm over 20 000 iterations. Note that the beam simulation forms a large proportion of the control algorithm. This makes the algorithm mainly an RI type. Thus, as in the case of the simulation algorithm, the powerful vector processing resources of the i860 are utilised to achieve the shortest execution time among the uni-processor architectures and with the i860+T8 among the parallel architectures. The execution times

achieved with the architectures are similar to those in case of the beam simulation algorithm. The speedup achieved with two C40s as compared to a single C40 is 1.4. Similarly, the speedup achieved with two T8s as compared to a single T8 is 1.23. These are consistent with the performance of these architectures in case of the beam simulation algorithm. Therefore, similar interpretation and explanation can be made with regard to the speedups and performances of the architectures in case of the beam control algorithm.

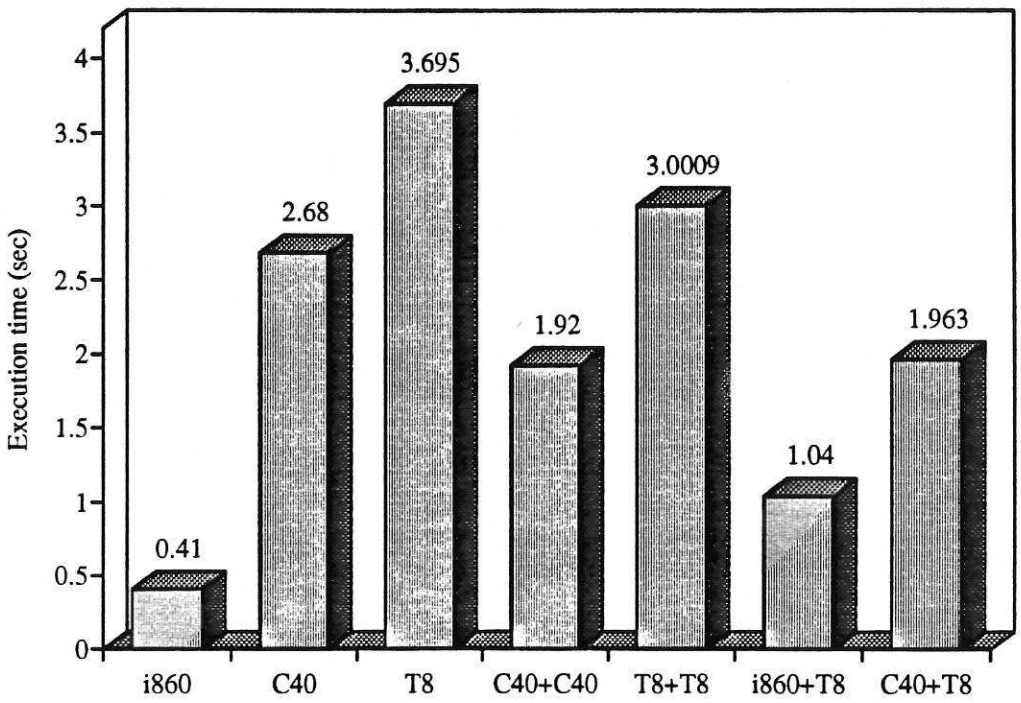

Figure 13: Execution times of the computing platforms in implementing the beam control algorithm.

6.5 *The RLS filter*

To investigate the real-time implementation of the RLS algorithm, an infinite impulse response (IIR) filter structure of second order was used. The execution times achieved by the computing platforms, in implementing the RLS filter algorithm over 1000 iterations are shown in Figure 14. It is noted that the C40 and the C40+C40 have performed as the fastest among the uni-processor and parallel architectures respectively. However, it is noted that the speedup achieved with the C40+C40 as compared to a single C40 is 1.215.

Similarly, the speedup achieved with the T8+T8 as compared to a single T8 is 1.102, which is not much different from the speedup achieved with the C40s. This is due the matrix computation as well as the increased level of data communication involved in the RLS filter algorithm. The i860 has performed about 1.323 times slower than the C40. This is due to irregular nature of algorithm. This is further reflected in the performance of the i860+T8 achieving longer execution time than the C40+T8.

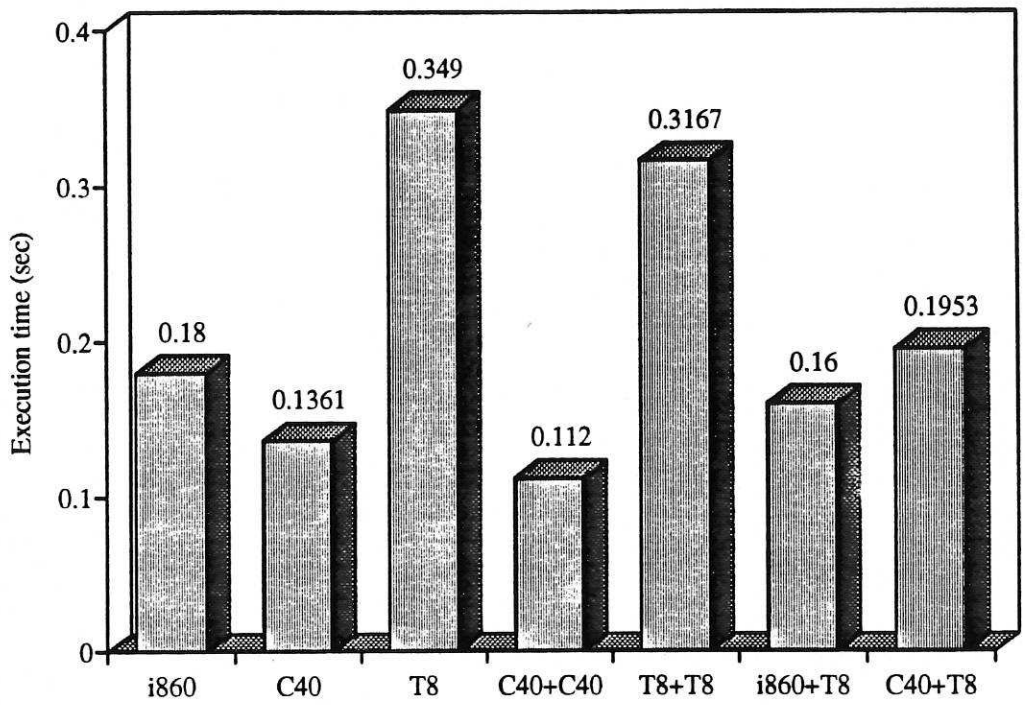

Figure 14: Execution times of the computing platforms in implementing the RLS filter.

6.6 The LMS filter

To investigate the real-time implementation of the LMS algorithm, the parameter $\mu = 0.04$ and a finite impulse response (FIR) filter structure of 40 weights was used. The execution times achieved by the computing platforms in implementing the LMS algorithm, over 1000 iterations, are shown in Figure 15. It is noted that in this case the C40 and the C40+C40 architectures have performed as the fastest among the uni-processor and parallel architectures respectively. The LMS algorithm, as noted earlier, incorporates some degree of irregularity due to the associated loops. The involvement of the regular DSP operations

in the process, on the other hand, provides some degree of regularity in the algorithm as well. The exploitation of these aspects of the process are evident in the performance of the C40 and the T8. It is noted that a super-linear speedup of 2.166 is achieved with two C40s as compared to a single C40. This is mainly due to a reduction in the amount of data that is to be handled by individual processors in the C40+C40 for which the internal cache and memory of the C40 is sufficient, thus, not requiring extra run-time memory management. The speedup achieved with two T8s in comparison to a single T8 1.669. An influential factor in achieving lower speed up in case of the T8+T8 architecture as compared the C40+C40 architecture is the serial communication link utilised with the T8s. The i860 vector processor does not have the resources necessary to be exploited in implementing the highly irregular features of the LMS algorithm and this is found to perform 2.833 times slower than the C40 and slightly faster than the T8. This is further reflected in the performance of the i860+T8 achieving the longest execution time among the parallel architectures. A further influential factor in the slow performance of the i860+T8 is the shared memory communication overhead in this architecture. Thus, the longer execution time achieved with the i860+T8 in comparison to a single i860 can mainly be due to communication overhead. In case of the C40+T8, on the other hand, both the C40 and the T8 contribute at a similar level to the performance achieved. Note that in case of the C40+T8 architecture the contribution of the serial to parallel communication link between the processors is significantly influential in slowing down the process in comparison to a single C40.

6.7 Flexible manipulator simulation

Figure 16 shows the performance of the computing platforms in implementing the manipulator simulation algorithm over 27 536 iterations. The simulation algorithm, as discussed earlier, is mainly of a regular matrix based computational type, similar to the flexible beam simulation algorithm, for which the powerful vector processing resources of the i860 are exploited and utilised to achieve the shortest execution time among the uni-

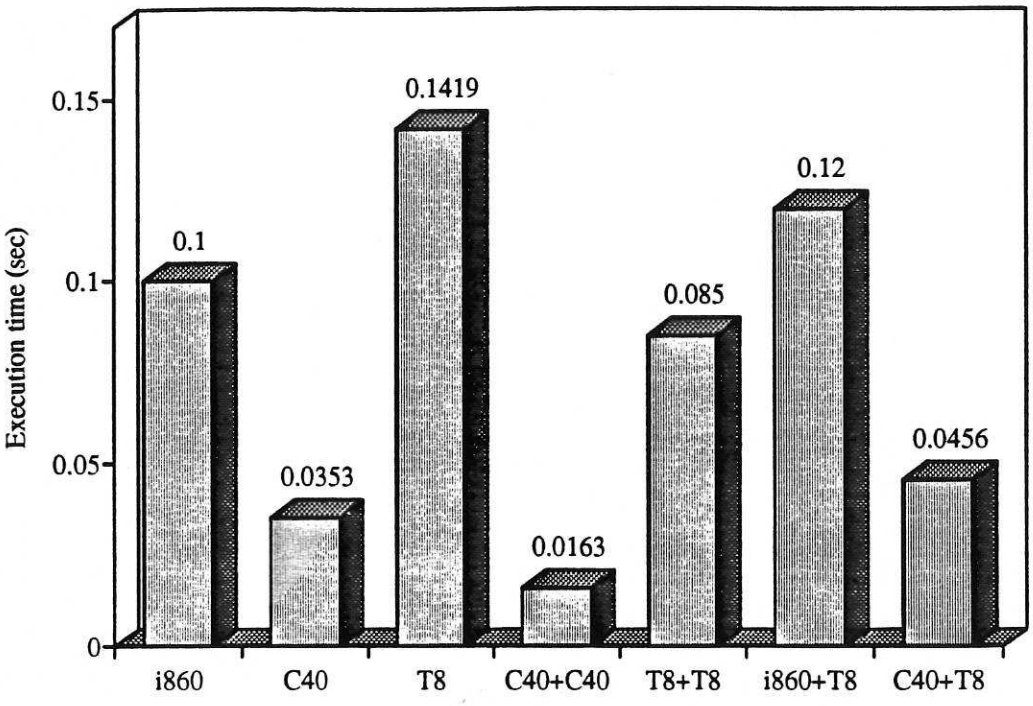

Figure 15: Execution times of the computing platforms in implementing the LMS filter.

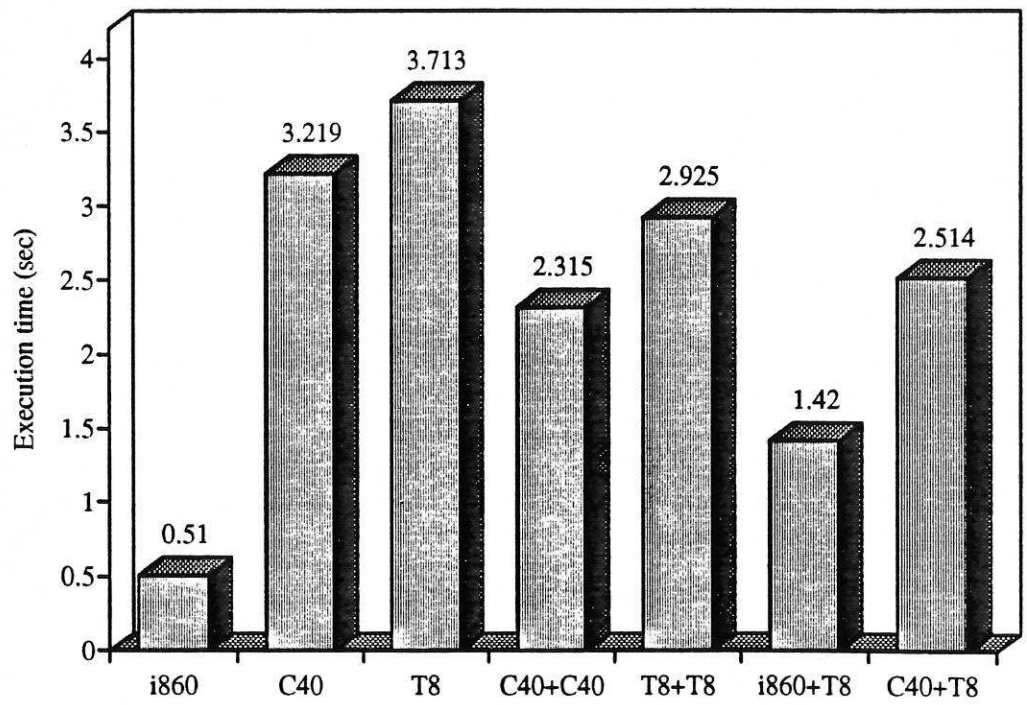

Figure 16: Execution time of the computing platforms in implementing the manipulator simulation algorithm.

processor architectures and with the i860+T8 among the parallel architectures. The C40 and the T8 have performed 6.312 and 7.280 times slower than the i860 respectively. This further implies that the C40 and the T8 do not have resources to be exploited in implementing matrix based algorithms. The speedup achieved with two C40s as compared to a single C40 is 1.390. Similarly, the speedup achieved with two T8s as compared to a single T8 is 1.269. This implies that the longer execution time achieved with the C40+T8 in comparison to the C40+C40 can mainly be attributed to the performance of the T8 as well as the serial to parallel communication link in this architecture. Similarly, the longer execution time achieved with the i860+T8 as compared to that with the i860 can mainly be attributed to the performance of the T8 and the shared memory communication overheads in this architecture.

6.8 Comparative performance of the computing platforms

In this section a summary of the performances of the computing platforms in implementing the algorithms is presented. Figures 17 and 18 show the execution times achieved with the uni-processor and parallel architectures in implementing the algorithms respectively. The algorithms along the horizontal axis have been arranged according to their degree of regularity; the FFT on the left is of a highly regular nature, whereas, the LMS filter algorithm on the right is of a highly irregular nature among the algorithms considered. It is noted that, among the uni-processor architectures, the i860 is performing as the fastest in implementing regular and matrix based algorithms. In contrast, the C40 is performing as the fastest of processors in implementing algorithms of irregular nature. In a similar manner, the performance of the T8 enhances with increasing degree of irregularity in an algorithm.

It is noted in Figure 18 that among the parallel architectures the suitability of the i860 for regular and matrix based algorithms is well reflected in the shortest execution times achieved with the i860+T8 in implementing the FFT, beam simulation, beam control, manipulator simulation and correlation algorithms. In contrast, the suitability of the C40 is

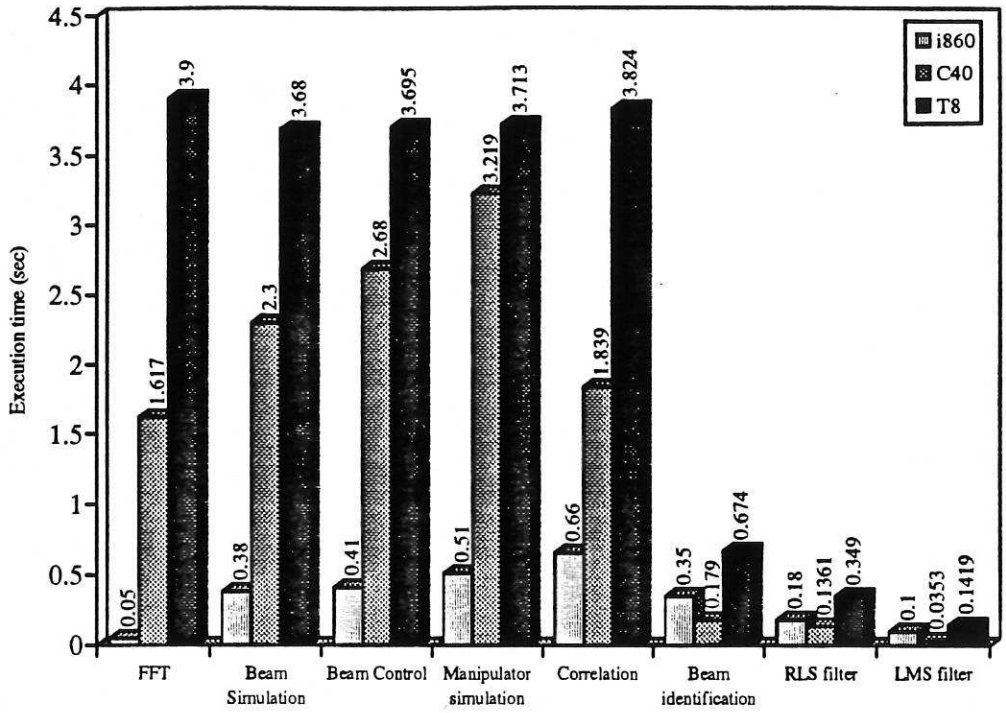

Figure 17: Execution times of the uni-processor architectures in implementing the algorithms.

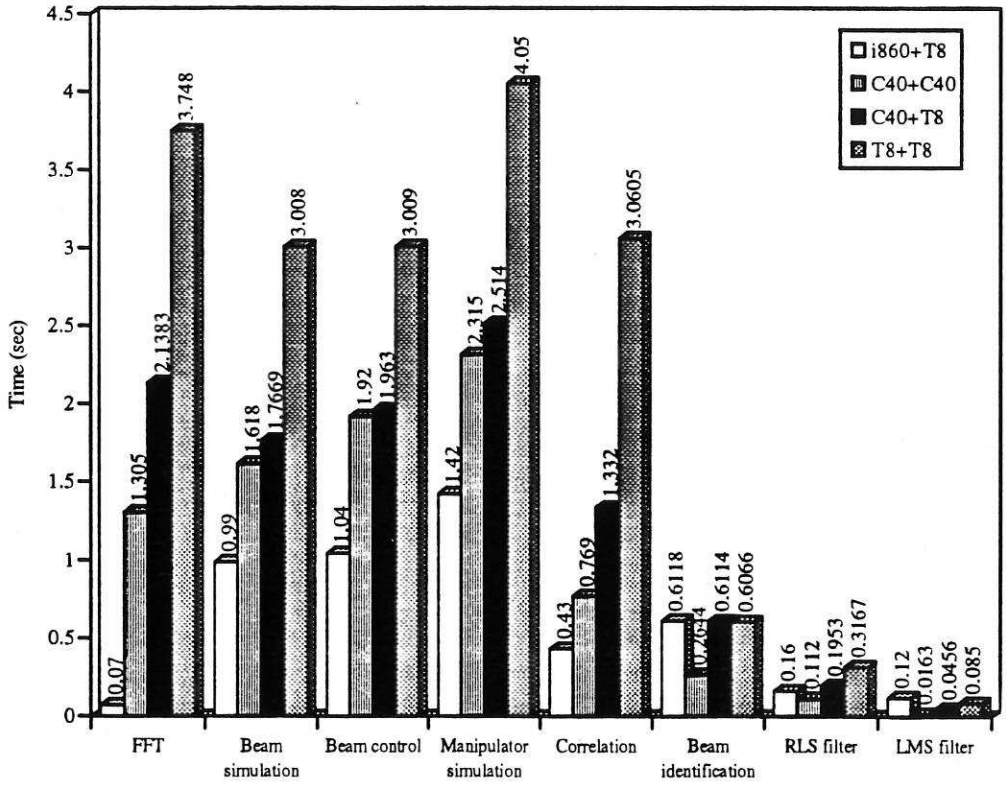

Figure 18: Execution times of the parallel architectures in implementing the algorithms.

reflected in achieving the shortest execution times in implementing the beam identification, RLS filter and LMS filter algorithms. Although, the T8+T8 is performing as the slowest of the parallel architectures in implementing the algorithms, its performance is consistent with the corresponding uni-processor architectures; the architecture is performing relatively better in implementing algorithms of irregular nature than those of regular and matrix nature. This also implies that the slower performance of the C40+T8 throughout in comparison to the parallel architectures in implementing the algorithms can mainly be attributed to the performance of the T8 and the serial to parallel communication link in this architecture.

7 Conclusion

This paper has been explored the real-time performance of a number of uni-processor and multi-processor, homogeneous and heterogeneous, parallel architectures, with PEs of contrasting features, in signal processing and control applications and the implementation issues of algorithms encountered in these applications. The inter-processor communication speed for four different homogeneous and heterogeneous architectures have been investigated and presented. Partitioning and mapping, granularity and regularity of algorithms have been discussed for better load distribution among the PEs in parallel architectures. A comparison of the results of the implementations has been made revealing the capabilities of the architectures and their suitability in the efficient implementation of signal processing and control algorithms.

This investigation has revealed that principally there is no one architecture achieving the best performance in implementing algorithms of various nature due to regularity of the algorithm, granularity of the algorithm and the hardware and inter-processor communication. For PP, performance measures such as MIPS and MFLOPS of the PEs are meaningless. The architectures and their clock rates, memory cycle times of the PEs, inter-processor communication speed, optimisation facility and compiler performance etc. all confuse the issue of attempting to rate the architecture. Of more importance is to rate

the performance of an architecture with its PEs on the type of program likely to be encountered in a typical application. The desired performance of a parallel architecture, thus, demands a close match between the capability of the architecture and the nature of the algorithm.

8 References

AGRAWAL, D. P., JANAKIRAM, V. K. and PATHAK, G. C. (1986). "Evaluating the performance of multicomputer configuration", IEEE - Computer, **19**, (5), pp. 23-37.

AZAD, A. K. M. (1994). "Analysis and design of control mechanisms for flexible manipulator systems", PhD Thesis, Department of Automatic Control and Systems Engineering, The University of Sheffield, UK.

BAXTER, M. J., TOKHI, M. O., FLEMING, P. J. (1994). "Parallelising algorithms to exploit heterogeneous architectures for real-time control systems", Proceedings of IEE Control-94 Conference, Coventry, 21-24 March 1994, **2**, pp. 1266-1271.

BROWN, A. (1991). "DSP chip with no parallel ?", Electronics world + Wireless world, October, pp 878-879.

CHING, P.C. and WU, S. W. (1989). "Real-time digital signal processing system using a parallel architecture", Microprocessors and Microsystems, **13**, (10), pp. 653-658.

CRUMMEY, T. P, JONES, D. J, FLEMING, P. J and MARNANE, W. P. (1994). "A hardware scheduler for parallel processing in control applications", Proceedings of IEE Control-94 Conference, Coventry, 21-24 March 1994, **2**, pp. 1098-1103.

CVETANVOIC, Z. (1987). "The effects of problem partitioning, allocation, and granularity on the performance of multiple processor systems", IEEE transactions on Computers, **C-36**, (4), pp. 421-432.

DENNING, P. J. (1986). "Parallel computing and its evolution", Communications of the ACM, **29**, (12), pp. 1163-1167.

HOCKNEY, R. W. and JESSHOPE, C. R. (1988). "Parallel computers 2. Architecture Programming and Algorithms", Adams Hilger, Bristol.

HWANG, K. (1993). "Advanced computer architecture - parallelism scalability programmability", McGraw-Hill, California.

HWANG, K. and BRIGGS, F. A. (1985). "Computer architecture and parallel processing", McGraw and Hill, California.

IFEACHOR, E. C. and JERVIS, B. W. (1993). "Digital signal processing - A practical approach", Addison - Wesley publishing company, UK.

IRWIN, G. W. and FLEMING, P. J. (1992). "Transputer in real-time control", John Wiley & Sons Inc, England.

JONES, D. I. (1989). "Parallel architectures for real-time control", Electronics and Communications Engineering Journal, **1**, (5), pp. 217-224.

KHOKHAR, A. A., PRASANNA, V. K., SHAABAN, M. E. and WANG, C. (1993). "Heterogeneous computing: challenges and opportunities", IEEE Computer, **26**, (6), pp. 18-27.

KOURMOULIS, P. K. (1990). "Parallel processing in the simulation and control of flexible beam structure system", PhD. Thesis, Department of Automatic Control and Systems Engineering, University of Sheffield, UK.

MAGUIRE, L. P. (1991). "Parallel architecture for Kalman filtering and self-tuning control", PhD thesis, The Queen's University of Belfast, UK.

MEGSON, G. M. (1992). "Practical steps towards algorithmic engineering", IEE Digest No. 1992/204: Colloquium on Applications of Parallel and Distributed Processing in Automation and Control, London.

NOCETTI, G. D. F and FLEMING, P. J. (1991). "Performance studies of parallel real-time controllers", Proceedings IFAC workshop on Algorithms and Architectures for Real-time Control, Bangor, UK, pp. 249-254.

PROAKIS, J. G. and MANOLAKIS, D. G. (1988). "Introduction to digital signal processing", Macmillan Publishing Company, USA.

STONE, H. S. (1987). "High performance computer architecture", Addison Wesley, USA.

TEXAS INSTRUMENTS. (1991). "TMS320C40 User's Guide", Texas Instruments, USA.

TOKHI, M. O. and HOSSAIN, M. A. (1994). "Self-tuning active vibration control in flexible beam structures ", Proceedings of IMechE-I: Journal of Systems and Control Engineering, **208**, (14), pp. 263-277.

TOKHI, M. O. and LEITCH, R. R. (1992). "Active noise control", Oxford Science Publications, Clarendan Press, Oxford.

TOKHI, M. O., VIRK, G. S. and HOSSAIN, M. A. (1992). "Integrated DSP^3 systems for adaptive active control", IEE Digest No. 1992/185: Colloquium on Active Techniques for Vibration Control - Sources, Isolation and Damping, London, 28 October, pp. 6/1-6/4.

TRANSTECH PARALLEL SYSTEMS LIMITED. (1991). "Transtech parallel technology", Transtech Parallel Systems Ltd, UK.

WIDROW B., GLOVER J. R., McCOOL, J. M., KAUNITZ, J., Williams, C. S., HEARN, R. H., ZEIDLER, J. R., DONG, E. and GOODLIN, R. C. (1975). "Adaptive noise cancelling: principles and applications", Proceedings IEEE, **63**, pp. 1692 - 96.