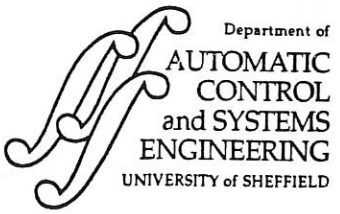
Monograph:
Tokhi, M.O. and Hossain, M.A. (1994) CISC, Risc and DSP Processors in Real-Time Signal Processing and Control. Research Report. ACSE Research Report 550 . Department of Automatic Control and Systems Engineering



CISC, RISC AND DSP PROCESSORS 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: (0742) 825136.
Fax: (0742) 731 729.
E-mail: O.Tokhi@sheffield.ac.uk.

Research Report No. 550

November 1994

Abstract

This paper presents an investigation into the nature of advanced high performance complex instruction set computer (CISC) processors, reduced instruction set computer (RISC) processors and digital signal processing (DSP) devices. Several DSP and control algorithms of regular and irregular nature are considered to explore the real-time characteristics of the different processors. The algorithms are implemented on several CISC, RISC and DSP processors. The hardware and software resources and capabilities of the processors and the characteristics of the algorithms are discussed to provide a matching between the algorithms and the architectures. Finally, a comparison of the results of the implementations is made, on the basis of real-time computation performance, to lead to merits of development of fast processing techniques for real-time DSP and control applications.

Key words: *Digital signal processing, complex instruction set computer, reduced instruction set computer, active vibration control.*

CONTENTS

LIST OF TABLES AND FIGURES

1 Introduction

Microprocessor - the brain of the computer is now a part of the development of human life. In modern real-time digital signal processing (DSP) and control applications involving complex and computationally intensive algorithms, however, performances are limited as sampling times are becoming shorter with increasing performance demands. The technology is developing at a rapid pace to overcome these high performance demands. The basis of the development of microprocessors technology is based on the (i) processing speed, (ii) processing ability, (iii) communication ability, and (iv) control ability. Every year brings new devices, new functions, and new possibilities. An imaginative and effective architecture for today could be klunker for tomorrow, and likewise, a ridiculous proposal for today may be ideal for tomorrow. There are no absolute rules to say that one architecture is better than another. In terms of design strategy, performance and facility every microprocessor possesses its own speciality (Carr, 1990; Stone, 1990).

For microprocessors with widely different architectures, performance measurements such as MIPS (million instructions per second) and MFLOPS (million floating-point operations per second) are meaningless. Of more importance is to rate the performance of each microprocessor on the type of program likely to be encountered in a typical application. The different architectures of microprocessors and their different clock rates, memory cycle times etc. all confuse the issue of attempting to rate the processors. This is an inherent difficulty to select microprocessors, for better performance in signal processing and control system development applications. The ideal performance of a microprocessor demands a perfect match between processor capability and the program behaviour. Processor capability can be enhanced with better hardware technology, innovative architectural features and efficient resources management. From the hardware point of view, currently performance varies due to the fact that whether the processor possesses pipeline facility, superscalar facility, is microcode operated, has internal cache or internal RAM, built-in math-coprocessor, floating point unit etc. In contrast, program behaviour is difficult to predict due to its heavy dependence on application and run-time conditions.

There are also many other factors affecting program behaviour, including algorithm design, data structures, languages efficiency, programmer skill, and compiler technology (Anderson, 1991; Hwang, 1993).

Conventional processors such as the Intel 486, 386, M68040, VAX/8600 and IBM 390 fall into the family known as CISC architecture. The typical clock rate of today's CISC processors ranges from 33 to 50 MHz. With microprogrammed control, the cycles per instructions (CPI) of different CISC instructions varies from 1 to 20. The CISC processors possess 8-24 general purpose registers, mostly with a unified cache for instructions and data, recent designs also use split caches. The addressing modes are normally within 12 to 24. Some modern CISC central processing units (CPUs) use hardwired control instead of microprogrammed control. In contrast, today's RISC processors, such as the Intel i860, SPARC TMS390S10, MIPS R3000 and IBM RS/6000 have faster clock rates ranging from 20 to 120 MHz determined by the implementation technology employed. With the use of hardwired control, the CPI of most RISC instructions has been reduced to 1 to 2 cycles. The RISC processors possess limited addressing modes (typically, 3 to 5) and large numbers of general purpose registers (typically 32 to 192) with mostly split data cache and instruction cache. The superscalar processors, which allow multiple instructions to be issued simultaneously during each cycle, form a special subclass of RISC processors. Thus, the effective CPI of a superscalar processor should be lower than that of a generic scalar RISC processor. The clock rate of a superscalar processor matches that of a scalar RISC processor. The very long instruction word (VLIW) architecture uses even more functional units than a superscalar processor. Thus, the CPI of a VLIW processor can be further lowered. Due to the use of very long instructions, VLIW processors have been mostly implemented with microprogrammed control. Thus, the clock rate is slow with the use of read-only memory (ROM). A large number of microcode access cycles may be needed for some instructions (Hwang, 1993). In contrast, an important goal in DSP hardware design is to optimise both the hardware architecture and the instruction set for DSP operations. This is achieved by making extensive use of the concepts of parallelism. In particular, the key architectural features used are (i) Harvard architecture, (ii) pipelining, (iii) fast,

dedicated hardware multiplier/accumulator, (iv) special instructions dedicated to DSP, (v) replication (more than one ALU, multiplier or memory unit) and (vi) on-chip memory/cache. The main advantage of DSP devices over general purpose microprocessors is that they contain dedicated circuitry which provides high resolution and high speed arithmetic operations. In some cases, for instance TMS320C40, the device possesses special features, e.g. parallel high speed communication links.

All microprocessors possess their own speciality for specific applications. This leads to inherent difficulties to explore comparative performance of different microprocessors. To explore the real-time performance in particular applications it is essential to implement the algorithm of that application into the processors. This paper presents an investigation into the performance evaluation of currently available high performance CISC, RISC and DSP processors in real-time applications. It explores the comparative hardware and software resources and real-time computational performances in implementing several complex and demanding algorithms in control and signal processing applications. These include, a fast Fourier transform (FFT) algorithm, a second order correlation algorithm, two different adaptive filter algorithms, a simulation algorithm of a flexible manipulator system and simulation, identification and active vibration control algorithms for a flexible beam system. The algorithms considered are described and classified according to their degree of regularity. The classifications are made on the basis of algorithms structure and in comparison to each other.

The algorithms are implemented on a number of different CISC, RISC and DSP processors, namely, an Intel 80860 (i860) RISC processor, a Texas Instruments TMS320C40 (C40) DSP processor, a SPARC TMS390S10 RISC processor, an Inmos T805 (T8) transputer RISC processor, a 486DX2 CISC processor and a 386DX CISC processor. The hardware and software resources and capabilities of the processors and the characteristics of the algorithms are discussed to explore the matching between the algorithms and architectures. Finally, a comparison of the results of the implementations is made, on the basis of real-time computation performance, to lead to merits of developing fast processing techniques for real-time applications.

2 Hardware

Six different RISC, CISC and DSP processors are considered in this investigation for real-time performance evaluation in implementing several signal processing and control algorithms. These are described below.

2.1 The 386DX (40)

The 386DX (40) is the Intel's 80386DX microprocessor. It consists of 275,000 transistors with external and internal 40 MHz clock speed and nearly 12 MIPS power. This is a CISC processor, possessing 32 - bit data bus and 32 address lines, allowing to address up to 4 gigabytes of physical memory. Moreover, the chip can handle up to 16 terabytes of virtual memory. It incorporates 16 bytes of pre-fetch cache memory. This special on board memory area is used to store the next few instructions of the program the chip is executing. Independently of the calculating portion of the chip, a special circuitry loads software code into this memory before it is needed. This small cache helps the 386 run more smoothly, with less waiting as code is retrieved from system memory. The virtual mode facility of the 386 processor gives freedom in running DOS program. This mode enables a single 386 microprocessor to divide its memory into many virtual machines, each machine alike entirely separate computer equipped with an 8086 microprocessor. This implies the multitasking facility of the 386 processor. The processor, however, does not have any internal or external math-coprocessor, floating point unit and pipelining facility. Moreover, it does not have internal cache or internal memory (Rosch, 1993).

2.2 The 486DX2 (50)

The 486DX2(50) is the Intel's CISC 80486DX2 processor. It consists of 1,200,000 transistors with internal and external 50 MHz clock speed and 54 MIPS. From a software standpoint, the 486 is distinguished from the 386 by one flag, one exception, two page-table entry bits, six instructions, and nine control register bits. The hardware of the 486,

however, differs substantially from the 386 and the changes mean more speed. Most important of these changes are a streamlined hardware design, tighter silicon design rules, an integral math-coprocessor, instruction pipelining, a built-in floating point unit and internal memory cache.

The streamlined hardware design (particularly its pipelining) means that the 486 can think faster than a 386 microprocessor when the two are operating at the same clock speed. Therefore, a 33 MHz 486 is faster than a 33 MHz 386. On most applications, the 486 is about twice as fast as a 386 at the same clock rate. Because of its improved internal design, the 486 reduces the number of clock cycles for most instructions.

Inside the chip, size has a more important influence. The larger a logic circuit element, the more power it can handle, and the more power it takes to make it work. Inside today's microprocessors, the primary limit to speed is heat dissipation. Running a chip too fast will heat its silicon until it boils its life away. Smaller circuits require less power, so they generate less heat and can operate faster. The 486 pioneered one-micron design rules, which means that the finest details etched into chip measure one micron across (Rosch, 1993).

The 80486 incorporates all the necessary coprocessor circuitry on the same slice of silicon. This internal coprocessor nearly doubles the performance of the processor. The faster a microprocessor operates, the more it suffers from the shortcomings of today's slow DRAM chips. In some systems, microprocessors spend one-third or more of their time waiting for memory to catch up. The 486 helps minimise the effect of this memory slowdown by incorporating its own high-speed memory cache. The cache in a 486 is organised as a four-way set associative design which essentially splits up its 8K total size as four smaller 2K caches, an arrangement that further enhances its performance, particularly in multi-threaded applications.

2.3 *SPARC TMS390S10*

SPARC is an acronym for Scalable Processor ARChitecture. Despite its independence from hardware implementation, the "scalable" part of the SPARC name refers to chip technology; specifically to the size of the smallest lines on the chip. The simple design of SPARC enables the chip design rules to be tightened easily (making the lines smaller) as fabrication technology improves. The result is a chip with finer details and more compact layout that enables faster operation. The Texas Instruments TMS390S10 is a RISC processor, possessing individual floating-point unit, integer unit and memory management unit (MMU) with 50 MHz clock speed, on-chip data and instruction cache. This is a processor within multi-tasking SUN system for which the performance at any time depends on the number of users.

2.4 *The T805 transputer*

The transputer (TRANSistor comPUTER) is high a performance microprocessor designed by INMOS Ltd. to facilitate interprocess and inter-processor communication and is targeted at the efficient exploitation of very large scale integration (VLSI) technology. The most important feature of the transputer is its external links which enables it to be used as a building block in the construction of low cost, high performance multiprocessing systems. Communication takes place (via these links) only between pairs of devices and it is distributed throughout the system, thus, overcoming the classic Von Neumann bottle-neck which is often encountered in bus-based systems. The particular technology used for the construction data bus effectively dictates an upper bound on the number of communications in these systems, whereas in a transputer based system, further processors can be added indefinitely. However, it should be noted that the efficient use of the processors in an arbitrary transputer network for an arbitrary application, is still a topic of intense research where definite solutions are not yet available. Moreover, even single processor applications can make use of the concurrent operation of the CPU and link processors. For example, at any one instant, the CPU might be processing one item of

data, one of the links might be transferring the next item of data from disk to memory and a further link might be transferring the previous calculated result from memory to disk.

The transputer family consists of several types of VLSI devices including the 16-bit T212, T225, the 32-bit T414, T425 and the floating-point T800, T805 and T9000 processors. The architectural features of T805, capable of operating concurrently with other features, are as follows (Transtech Parallel Systems Ltd, 1993).

- The T805 is 84 pin VLSI microchip measuring 27 x 27 x 2mm and a typical architecture can incorporate a single T805 floating point transputer with 2MBytes of external memory on a board measuring a mere 90 x 53 x 15mm.
- It is a 32-bit RISC processor, with 25MHz clock speed and is able to yield up to 20MIPS performance, including hardware support for simulating concurrence on a single processor by time slicing the CPU.
- It has fast on-chip 4KBytes static RAM .
- It incorporates external memory controller with either multiplexed address and data buses for economy of device pins or non-multiplexed for performance.
- It has, typically, four high speed bi-directional links for communication between pairs of devices within the family. The links operate at speeds of 20Mbits/sec and can achieve data transfer rates of up to 1.7MBytes/sec unidirectionally or 2.3 MBytes/sec bidirectionally.

2.5 The i860 microprocessor

The Intel i860 has been designed for numerically and vector intensive applications. Many of the design principles used have been adopted from supper computer technology enabling the i860 to deliver a peak arithmetic performance of 80MFLOPS (single precision) and 60MFLOPS (double precision) in conjunction with a peak integer performance of 40MIPS. In particular, its high throughput is achieved from a combination of RISC design techniques, pipelined processing units, wide data paths and large on-chip

caches. Implemented with a single chip with over 1 million transistors, the i860 supports a 64-bit architecture and is capable of executing up to three operations each clock cycle (25ns @ 40MHz). On a single chip the architecture supports (i) integer operations, (ii) floating point operation, (iii) graphics operations, (iv) memory-management unit and (v) data cache and instruction cache.

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 unit and the adder unit, 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 unit and the floating-point control unit can execute concurrently. In this sense, the i860 is also a superscalar RISC processor capable of executing two instructions, one integer and one floating-point, at the same time. The i860 executes 82 instructions, including 42 RISC integer, 24 floating-point, 10 graphics and 6 assembler pseudo operations. All the instructions are executed in one cycle each. This equals 25 ns for a 40 MHz clock rate (Hwang, 1993).

The ability to provide all these facilities, on the same silicon, enables hardware developing systems that are less dependent on many external components normally associated with sophisticated computer systems. The i860 is an ideal candidate for integration into highly parallel computer environments with high computational performance, modularity and real-state requirements.

2.6 The TMS320C40

This is a high performance Texas Instruments 32-bit DSP device with 40 MHz clock speed, 8KBytes internal memory, 512 bytes instructions cache and is capable of 40

MFLOPS. This DSP processor possesses six parallel high speed communication links for inter-processor communication; 20Mbytes/sec asynchronous transfer rate at each port for maximum data throughput. The CPU is capable of 275 MOPS with the following key features

- Eleven operations/cycle throughput, resulting in massive computing and sustained CPU performance.
- 25ns instruction cycle times.
- 40/32-bit single-cycle floating-point/integer multiplier for high performance in computationally intensive algorithms.
- Single-cycle IEEE floating-point conversion for efficient interface to IEEE-compatible processors.
- Hardware divide and inverse square root support for high performance.
- Byte and half-word manipulation capabilities for fast data (un)packing.
- Support for linear, circular, and bit-reversed addressing for high performance.
- Single cycle branches, calls, and returns for fast program control.
- Relocatable reset and interrupt vectors for easy integration into parallel processing systems.

In contrast, the device possesses two identical external data and address buses supporting shared memory systems and high data rate, single-cycle transfers. 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. It possesses a primary register file containing 32 registers and an expansion register file consisting of two registers for coping with interrupts. As part of the primary register file there are twelve extended precision, 40-bit registers designed to maintain extended floating point precision. In the normal sense of a microprocessor these act as accumulators. Eight auxiliary 32-bit registers support a variety of addressing modes and are used to generate a 32-bit address for local or global memory. The remaining registers support other system

functions such as stack management, processor status, interrupt management, block instruction repeats and various addressing modes. The components in the CPU are serviced by two 32-bit register buses and two 40-bit CPU busses enabling the CPU to achieve a very high degree of parallelism - one of the principal attributes of the C40 (Brown, 1991; Hwang, 1993; Texas Instruments, 1991).

3 Software support

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 efficiency of the mechanism for 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 in this investigation were coded in high-level languages consisting of ANSI C, 3L Parallel C and Borland C as appropriate for the hardware used. Table 1 shows the compilers with the corresponding computing platforms used.

4 Algorithms

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

4.1 FFT algorithm

A real periodic discrete-time signal $x(n)$ of period N can be expressed as a weighted sum of complex exponential sequences. Because of the fact that sinusoidal sequences are unique only for discrete frequencies from 0 to 2π, the expansion contains only a finite number of complex exponentials. The complex discrete Fourier transform (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 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 (Proakis and Manolakis, 1988).

4.2 Cross-correlation

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 $r_{xy}(l)$ indicates 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 (Ifeachor and Jervis, 1993; Proakis and Manolakis, 1988).

4.3 Simulation and active vibration control of a flexible beam structure

Consider a cantilever beam system 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 and $y(x,t)$ is the deflection 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)

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

where μ is a beam constant given by $\mu^2 = \frac{EI}{\rho A}$, with ρ, A, I and E representing the mass density, cross-sectional area, moment of inertia of the beam and the Young's 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

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

Note that the model, thus, utilised incorporates no damping. To construct a suitable platform for test and verification of the control mechanism (introduced later), a method of obtaining numerical solution of the PDE in equation (6) is required. 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 $\frac{\partial^2 y(x,t)}{\partial x^2}$

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

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

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, $\lambda^2 = \frac{(\Delta t)^2}{(\Delta x)^4}\mu^2$. Equation (9) gives the deflection of point i along the beam at time step $j+1$ in terms of the deflections of the point at time steps j and $j-1$ and deflections of points $i-1$, $i-2$, $i+1$ and $i+2$ at time step j. Note that in evaluating the deflection at the grid point $i=1$ the fictitious deflection $y_{-1,j}$ will be required. Similarly, in evaluating the deflection at the free end of the beam, $i=n$ (n representing the total number of sections along the beam), the fictitious deflections $y_{n+1,j}$ and $y_{n+2,j}$ will be required. To obtain these, the boundary conditions in equation (7) are used. In a similar manner as above, the boundary conditions in equation (7) can be expressed in terms of the FD approximations as

$$y_{0,j} = 0 \quad , \quad \frac{y_{1,j} - y_{-1,j}}{2\Delta x} = 0$$
$$\frac{y_{n+1,j} - 2y_{n,j} + y_{n-1,j}}{(\Delta x)^2} = 0 \quad , \quad \frac{y_{n+2,j} - 2y_{n+1,j} + 2y_{n-1,j} - y_{n-2,j}}{2(\Delta x)^3} = 0 \tag{10}$$

Solving equation (10) for the deflections $y_{-1,j}$ and $y_{0,j}$ at the fixed end and $y_{n+1,j}$ and $y_{n+2,j}$ at the free end 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 the construction of the simulation algorithm. 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 along the beam 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 $0<\lambda^2\leq 0.25$ (Virk and Kourmoulis, 1988).

A schematic diagram of an active vibration control (AVC) structure is shown in Figure 1. 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 1 is to achieve total (optimum) vibration suppression at the observation point. Synthesising the controller on the basis of this objective yields (Tokhi and Letich, 1991)

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

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. An RLS parameter estimation algorithm is used here to estimate Q_0 and Q_1 in the discrete-time domain in parametric form. The process of control, on the other hand, involves designing the controller according to equation (13) and implementing this in real-time.

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 for estimation of parameters of the system models Q_0 and Q_1. This is based on the well known least squares method. 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 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 2. 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

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

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 Young's 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

$$\begin{aligned} &y(0,t) = 0 \\ &I_h\frac{\partial^3 y(0,t)}{\partial t^2 \partial x} - EI\frac{\partial^2 y(0,t)}{\partial x^2} = \tau(t) \end{aligned} \qquad (21)$$

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 be written 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, $a = 2 - \frac{6\Delta t^2 EI}{\rho \Delta x^4}$, $b = \frac{4\Delta t^2 EI}{\rho \Delta x^4}$ and $c = \frac{\Delta t^2 EI}{\rho \Delta x^4}$. 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,

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

$$A = \begin{bmatrix} m_1 & m_2 & m_3 & 0 & 0 & \dots & 0 & 0 & 0 & 0 & 0 \\ b & a & -b & -c & 0 & \dots & 0 & 0 & 0 & 0 & 0 \\ -c & b & a & b & -c & \dots & 0 & 0 & 0 & 0 & 0 \\ \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots \\ 0 & 0 & 0 & 0 & 0 & \dots & -c & b & a & b & -c \\ 0 & 0 & 0 & 0 & 0 & \dots & 0 & m_{11} & m_{12} & m_{13} & m_{14} \\ 0 & 0 & 0 & 0 & 0 & \dots & 0 & m_{21} & m_{22} & m_{23} & m_{24} \end{bmatrix}, \quad F = \begin{bmatrix} \tau(i,j) \\ 0 \\ \vdots \\ 0 \end{bmatrix}, \quad B = \frac{\Delta t^2}{\rho}$$

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). The displacement is obtained, here, for two sets of boundary conditions, first by ignoring hub inertia and payload and then including hub inertia and payload. The results obtained from these two sets of boundary conditions are also compared.

4.5 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 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 $x(k)$ (Ifeachor adn Jervis, 1993; 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

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

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

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

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

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

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

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

5 Implementations and Results

The algorithms described in the previous section are of various degrees of regularity. Regularity is a term used to describe the degree of uniformity in the execution thread of the computation. Many of the signal processing algorithms can be expressed by matrix computations. These are called regular iterative (RI) due to their very regular structure. An algorithm incorporating loops and conditional jumps, amounting to varying execution times from one iteration to another, is referred to as irregular. The algorithms considered are implemented on the various computing platforms in this section and the results obtained are assessed and compared on the basis of real-time performance. The matching and mismatching between the regular/irregular algorithms and hardware are also explored.

5.1 The FFT algorithm

Figure 3 shows the real-time performance of the different computing platforms for a 512-point FFT algorithm. As noted, the FFT algorithm is a regular DSP process and highly matrix based. Thus, the i860, with its powerful vector processing resources, achieves the best performance among the processors. In contrast, the C40 achieves 10-15 times slower execution time as compared to the i860. The SPARC RISC processor achieves a better performance than the C40, 486DX2 and the T8 processor, but slower execution time as compared to the i860. The 386DX takes the highest execution time among the processors.

The 486DX2 achieves slower execution time as compared to the SPARC, but is faster as compared to a single T8 and the 386DX.

5.2 Cross-correlation

To investigate the real-time implementation of the correlation algorithm, two waveforms of 1000 samples each were used. Figure 4 shows the execution time of the different processors. Note that the correlation algorithm is an RI type signal processing algorithm. Thus, the superscalar vector processor i860 appears to achieve the lowest and the 386DX the highest execution time among the computing domains. On the other hand, the T8 and the C40 perform about 5.8 and 2.8 times slower than the i860, respectively. The 486DX2 and the SPARC processor achieve similar execution times. The 386DX is the slowest of all, possibly, due to the floating point operations being evaluated in software rather than using dedicated hardware. Moreover the 386DX does not have math-coprocessor, cache or internal memory making it slower to handle large amounts of data calculation.

5.3 Simulation, identification and control of the flexible beam

5.3.1 Simulation

Figure 5 shows the execution time for the different microprocessors in implementing the simulation algorithm for 20 000 iterations. It is noted that in this case the superscalar i860 RISC processor performs as the fastest whereas the 386DX machine is the slowest of the processors used. 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 lowest execution time. The C40 does not have such vector processing resources making it about 6 times slower than the i860. This implies that the C40 is not performing well in a situation where the algorithm is of a matrix type and where extensive run time memory management is involved. The transputer, on

the other hand, is performing about 9.864 times slower than the i860. The SPARC processor and the 486DX2 appear to achieve similar performances, the SPARC being slightly faster due to its RISC processor. The 386DX performance was the slowest of all the processors in this case. This, as compared to the 486DX2 machine, is mainly due to the floating point operations evaluated in software rather than using dedicated hardware since it has no maths co-processor. Moreover, this machine does not have cache or internal memory making it slower to handle calculation of large amounts of data.

5.3.2 Identification

As discussed earlier, the identification algorithm with its irregular nature 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. Figure 6 shows the total execution times of the architectures used in implementing the identification algorithm over 1000 iterations. It is noted that among the processors used the C40 is performing as the fastest and the 386DX as the slowest. The algorithm does incorporate some matrix manipulation. However, as a result of the irregular nature of the algorithm, the i860 is found to perform even slower than the C40. In contrast, the T8 is performing well and at similar level as the SPARC and 486DX2 processors. The 386DX processor, on the other hand, is performing slower than these.

5.3.3 Control

Figure 7 shows the total execution times achieved by the various architectures used in implementing the control algorithm for 20 000 iterations. It is noted that among these processors the i860 is the fastest and the 386DX is the slowest. Note that the beam simulation forms a large proportion of the control algorithm. This makes the algorithm mainly a very regular iterative type. Thus, as in the case of the simulation algorithm, the powerful vector processing resources of the i860 are utilised to achieve the smallest execution time among the processors. The SPARC processor, 486DX2 machine and

386DX machine appear to perform at similar levels as in the case of the simulation algorithm.

5.4 RLS filter

The execution times achieved by the architectures, in implementing the RLS filter algorithm for 1000 iterations are shown in Figure 8. It is noted that among these the C40 is performing as the fastest whereas the 386DX is performing as the slowest. The i860 is performing about 1.323 times slower than the C40. This is due to the irregular nature of this algorithm. In contrast, the T8 achieves better performance as compared to the 486DX2 processor. The SPARC RISC processor achieves slower execution time as compared to the C40 but similar to the i860 RISC processor.

5.5 LMS filter

To investigate the real-time implementation of the LMS algorithm, the parameter $\mu = 0.04$ was used. The execution times achieved by the architectures, in implementing the LMS algorithm for 1000 iterations, are shown in Figure 9. It is noted that in this case the C40 performs as the fastest whereas the 386DX machine as the slowest of the processors used. 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 processes, on the other hand, provides some degree of regularity in the algorithm as well. The exploitation of these two aspects of the process are evident in the performance of the C40 DSP device and the T8. The i860 vector processor does not have the resources necessary to be exploited in implementing the highly irregular nature of the LMS algorithm and thus is found to perform 2.83 times slower than the C40, slightly faster than the T8 and slower than the 486DX2 and the SPARC RISC processor.

5.6 *Simulation of the flexible manipulator*

Figure 10 shows the performance of the different processors in implementing the flexible manipulator simulation algorithm for 27 536 iterations. The algorithm as described earlier, is a very regular matrix based, similar to the flexible beam simulation algorithm. Thus, in a similar manner, the i860 processor achieves the fastest performance and the 386DX processor as the slowest of the processors. The SPARC processor and the 486DX2 perform at a similar level to one another, but faster than the C40 DSP processor. The C40 DSP device is about 6.3 times slower as compared to the i860 processor. This further implies that the C40 does not have resources to exploit in implementing matrix based algorithms.

5.7 *Comparative performance of processors with different algorithms*

To explore the comparative characteristics of the processors in implementing the various algorithms, the six processors are divided into two classes. As noted and discussed above, the i860, SPARC and the C40 dominantly outperform the rest of the processors. Thus, the i860, SPARC and the C40 are grouped into one category and the rest into another category. A comparative performance of the processors in the first category is shown in Figure 11. For better presentation the execution time for identification, RLS and LMS algorithms are magnified into 2, 4 and 8 times respectively. It is noted that the i860 processor performs as the fastest of the processors in implementing regular and vector based algorithms. In contrast, the C40 performs as the fastest of the processors in implementing algorithms of an irregular nature. As compared to the C40, the SPARC processor performs faster in implementing regular and vector based algorithms but slower in implementing the irregular type algorithms.

Figure 12 shows a comparative performance of the processors in the second category, namely, 486DX2, T8 and 386DX in implementing the algorithms used. It is noted that the 486DX2 performs as the fastest of the processors in implementing all, but the identification and RLS algorithms. The T8 RISC processor performs slower than the

486DX2 in implementing regular and matrix based algorithms. In contrast, the T8 performs relatively better in implementing irregular type algorithms (identification and RLS) than matrix based and regular algorithms. The 386DX processor performs as the slowest of the processors in implementing the algorithms considered. It is noted that the superscalar i860 RISC processor is best in implementing regular matrix based algorithms and the C40 in implementing regular or irregular but non-matrix based algorithms.

6 Conclusion

This paper has explored the real-time implementation of several signal processing and control algorithms on a number of different computing architectures. The i860 vector processor and the C40 DSP device with their hardware and software architectures optimised to achieve fast processing in DSP applications are found to perform relatively better in implementing regular and irregular DSP operations. Special features, such as vector processing resources in an i860 RISC processor, are exploited to give even better performance in applications involving matrix manipulations. It has been demonstrated that, in practice, there is generally a mismatch between hardware requirements of an algorithm and the hardware resources of an architecture leading to a disparity in their relative performance. Therefore, to fully exploit the architectures a close match needs to be forged between the algorithm and the underlying hardware, with due consideration of the suitable programming language for the application, and issues such as algorithmic regularity and granularity. It follows from the investigation presented that, equally clearly, there is no one processor for the solution of best real-time performance in terms of computation in implementing algorithms of different nature. Computational performance varies with granularity of hardware and with granularity and regularity of an algorithm.

7 References

ANDERSON, A. J. (1991). "A performance evaluation of microprocessors, DSPs and the transputer for recursive parameter estimation", Microprocessors and Microsystems, **15,** (3), pp. 131-136.

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.

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

CARR, A. G. (1990). "Real-time signal processing a review of applications and technology", Computing and Control Engineering Journal, **1**, pp. 77-80.

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

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

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

ROSCH, W. L. (1993). "Hardware bible", Brady Publishing, Indianapolis.

STONE, H. S. (1990). "High performance computer architecture", Addison - Wesley, USA

TEXAS INSTRUMENTS, (1991), " TMS320C4x User's Guide", Texas Instruments, USA.

TOKHI, M. O. and LEITCH, R. R. (1991). "Design and implementation of self-tuning active noise control systems", IEE proceedings-D, **138**, (5), 421-430.

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

TRANSTECH PARALLEL SYSTEMS LIMITED, (1993). "Transtech parallel technology", UK.

VIRK, G. S. and KOURMOULIS, P. K. (1988). On the simulation of systems governed by partial differential equations, Proceedings of IEE Control-88 Conference, Oxford, 13-15 April, pp. 318-321.

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 of IEEE, **63**, pp. 1692 - 1696.

Table 1: Compilers used for different computing platforms

Processor	i860	C40	T8	SPARC	486DX2	386DX
Compiler	Portland Group ANSI C	3L Parallel C, V. 1.0.1	3L Parallel C, V. 2.1	ANSI C for UNIX	Borland C	Borland C

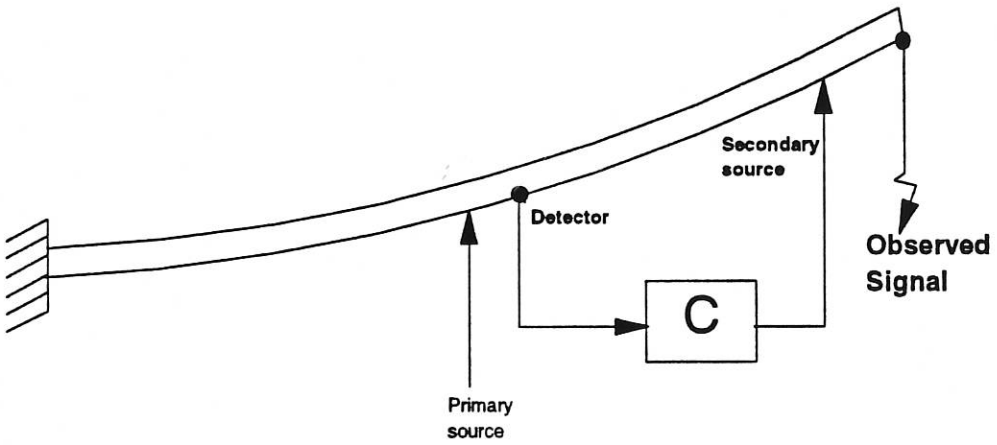


Figure 1: Active vibration control structure

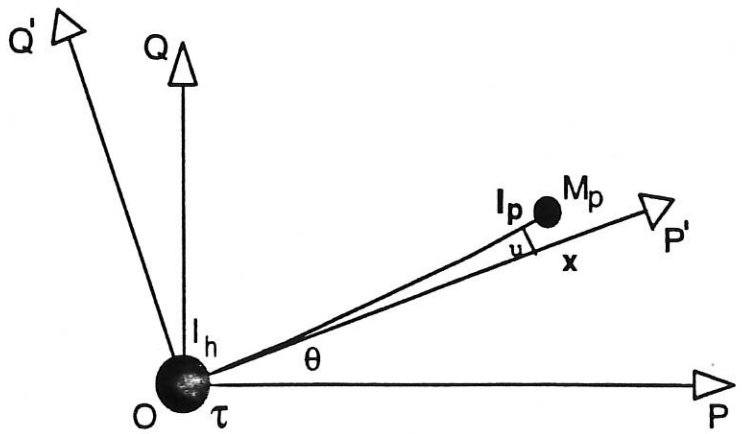


Figure: 2 Schematic representation of the flexible manipulator system.

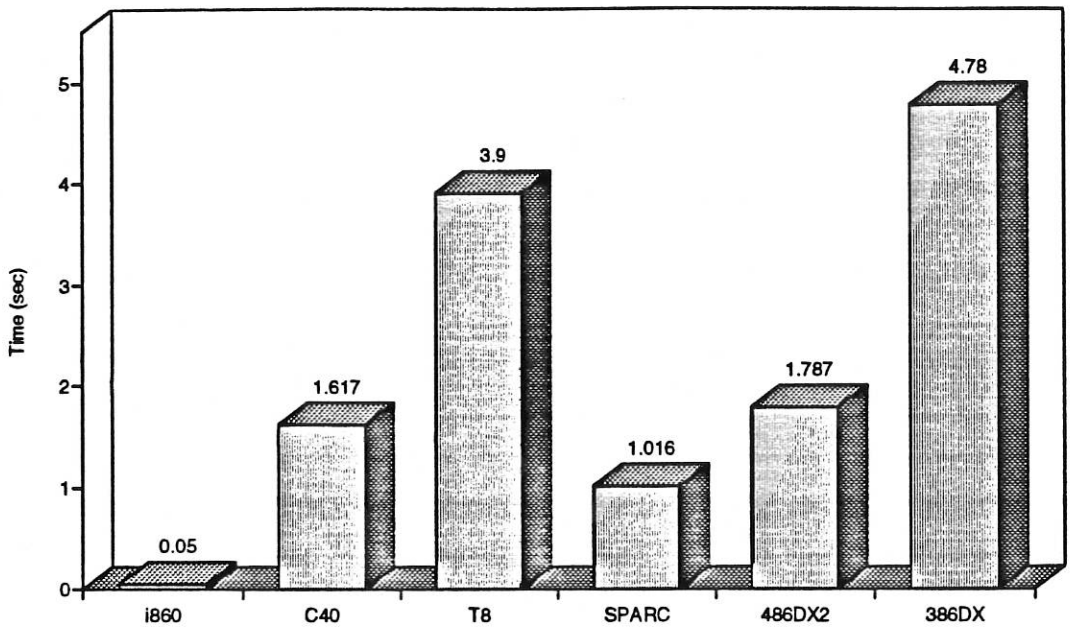


Figure 3: Execution times of processors in implementing the FFT algorithm.

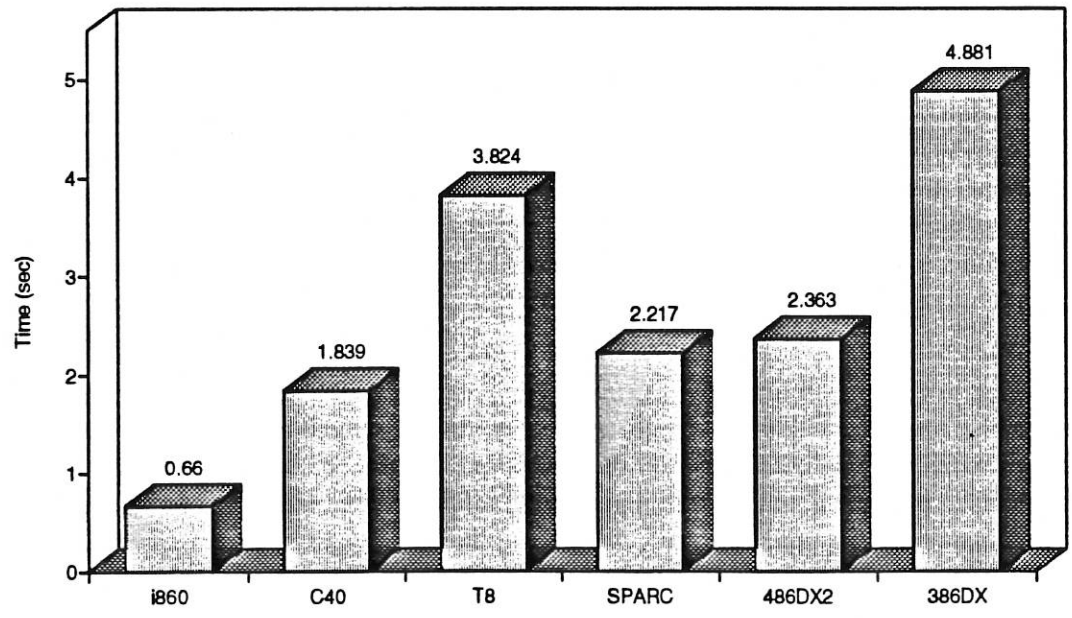


Figure 4: Execution times of processors in implementing the correlation algorithm.

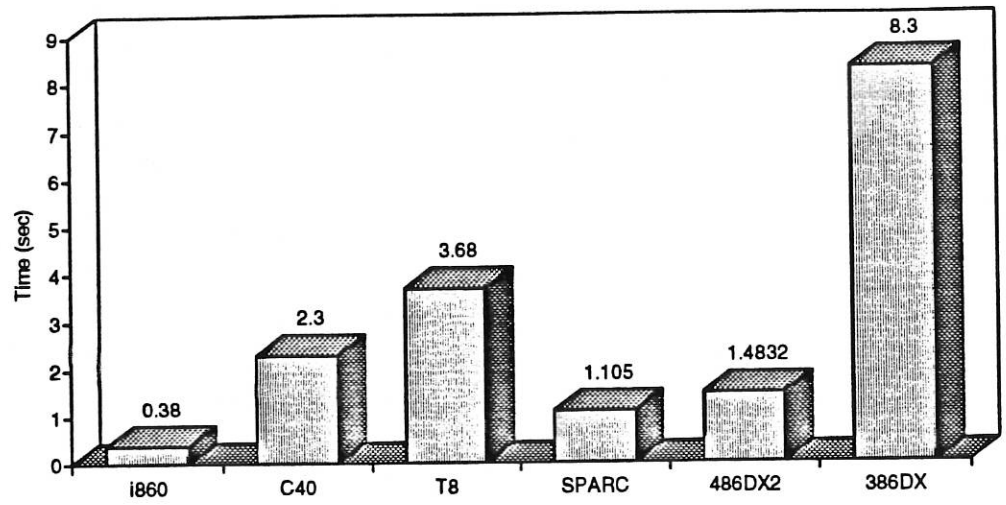


Figure 5: Execution times of processors in implementing the beam simulation algorithm.

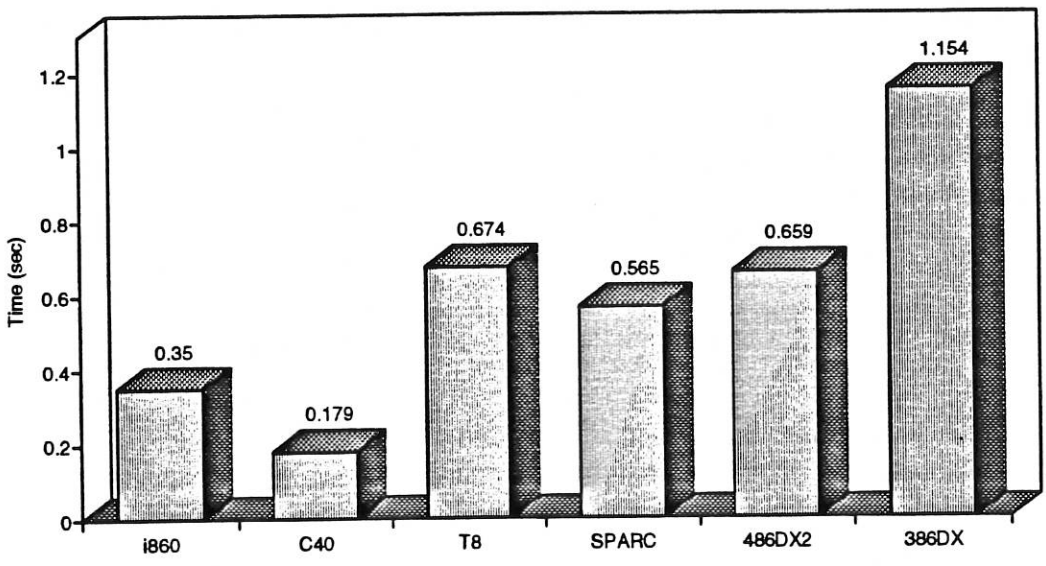


Figure 6: Execution times of processors in implementing the identification algorithm.

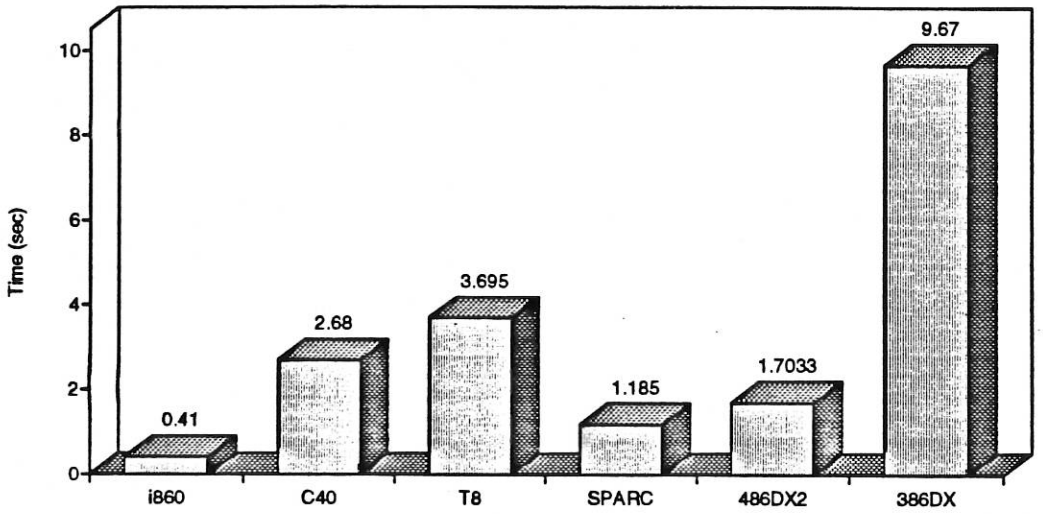


Figure 7: Execution times of processors in implementing the control algorithm.

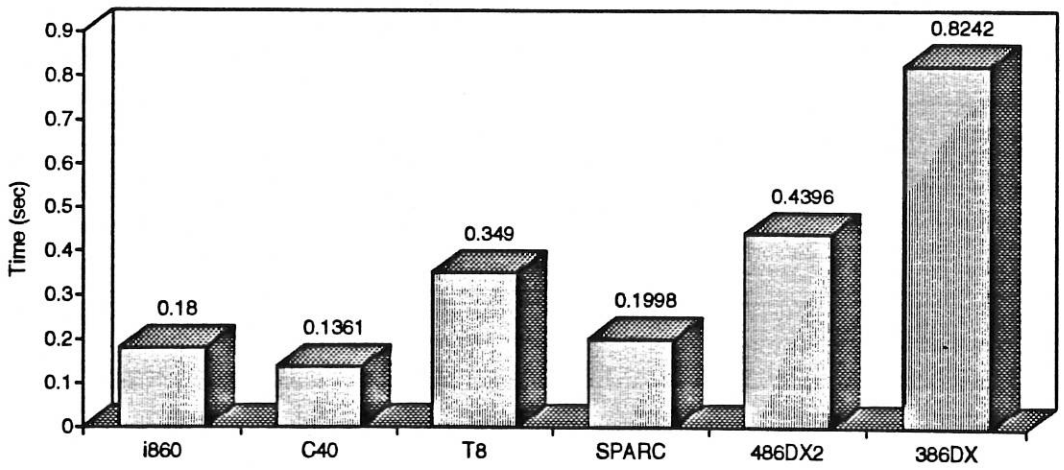


Figure 8: Execution times of the processors in implementing the RLS filter algorithm.

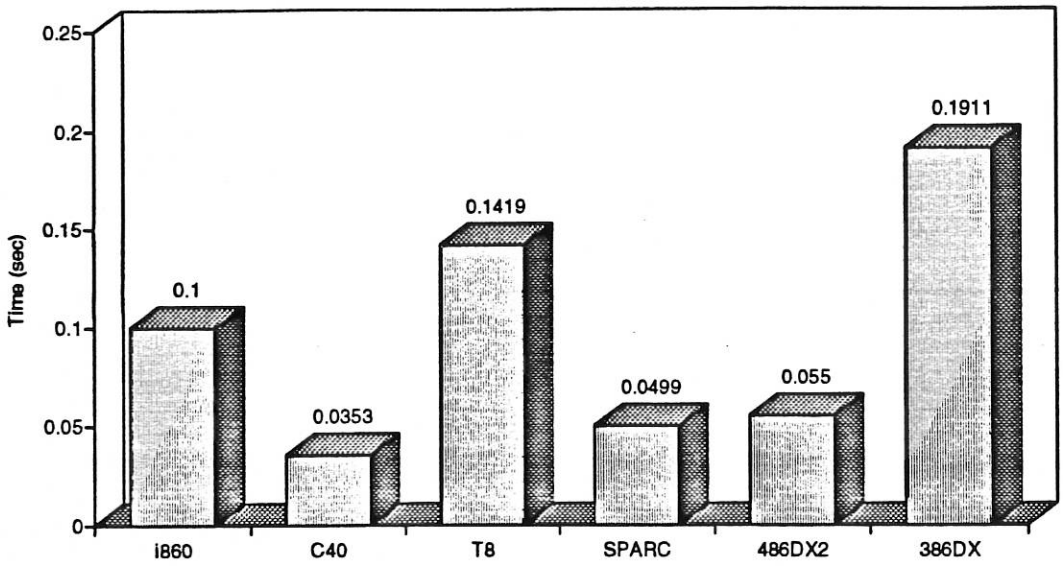


Figure 9: Execution times of the processors in implementing the LMS filter algorithm.

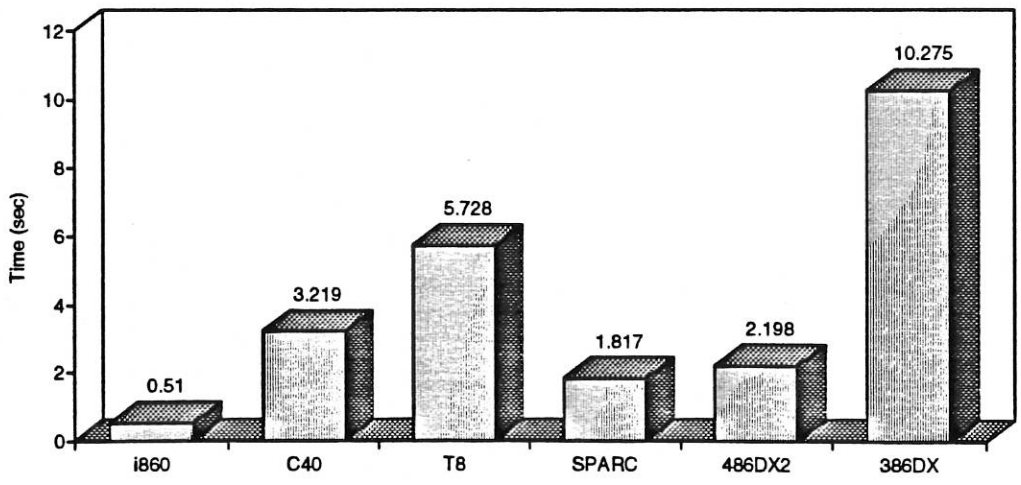


Figure 10: Execution times of processors in implementing the flexible manipulator simulation algorithm.

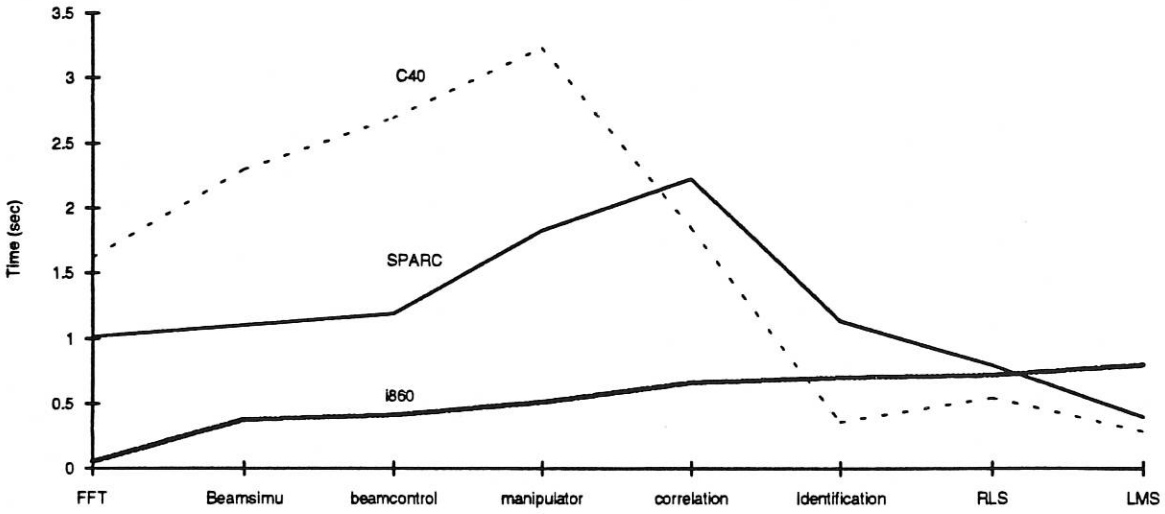


Figure 11: Performance of the i860, SPARC and C40 processors in implementing the algorithms.

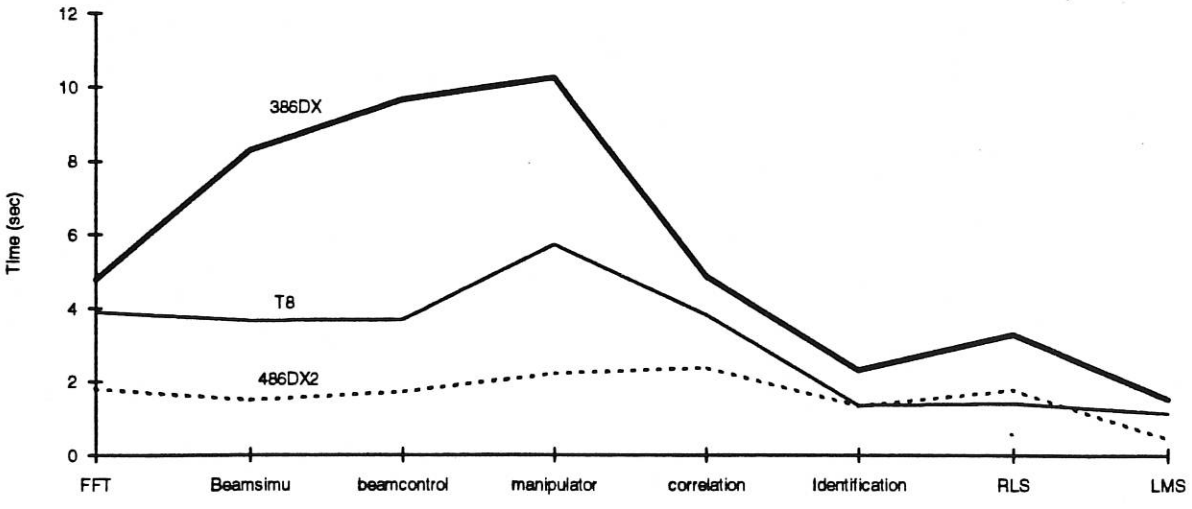


Figure 12: Performance of the 486DX2, T8 and 386DX processors in implementing the algorithms.