Robotica (1998) volume 16, pp. 445–456. Printed in the United Kingdom © 1998 Cambridge University Press

Sequential and parallel real-time simulation of a flexible manipulator system

M.O. Tokhi*, M.A. Hossain† and A.K.M. Azad*

(Received in Final Form: August 6, 1997)

SUMMARY

This paper presents an investigation into the utilisation of sequential and parallel processing techniques for the real-time simulation of a flexible manipulator system. A finite dimensional simulation of the system is developed using a finite difference approximation to the governing dynamic equation of the manipulator. The developed algorithm is implemented on a number of uni-processor and multi-processor, homogeneous and heterogeneous, parallel architectures. A comparison of the results of these implementations is made and discussed, on the basis of real-time processing requirements in the simulation and control of flexible manipulator systems.

KEYWORDS: Digital signal processing; Finite difference method; Parallel processing; Performance evaluation; Real-time simulation; Sequential processing.

1. INTRODUCTION

Flexible manipulator systems offer several advantages in contrast to the traditional rigid ones. These include faster system response, lower energy consumption, requiring relatively smaller actuators, less overall mass and, in general, less overall cost. However, due to the distributed nature of the governing equations describing system dynamics, the control of flexible manipulators has traditionally involved complex processes.[1–3] Moreover, to compensate for flexure effects and thus yield robust control the design focuses primarily on noncolocated controllers.[4,5] It is important initially to recognise the flexible nature of the manipulator and construct a mathematical model for the system that accounts for the interactions with actuators and payload. Such a model can be constructed using partial differential equations (PDEs). A commonly used approach for solving a PDE representing the dynamics of a manipulator, sometimes referred to as the separation of variables method, is to utilise a representation of the PDE, obtained through a simplification process, by a finite set of ordinary differential equations. Such a model, however, does not always represent the fine details of the system.[6] A method in which the flexible manipulator is modelled as a massless spring with a lumped mass at one end and lumped rotary inertia at the other end has previously been proposed.[7,8] Unfortunately, the solution obtained through this method is also not accurate and suffers from similar problems as in the case of the method of separation of variables. The finite element (FE) method has also been previously utilised to describe the flexible behaviour of manipulators.[9,10] The computational complexity and consequent software coding involved in the FE method is a major disadvantage of this technique. However, as the FE method allows irregularities in the structure and mixed boundary conditions to be handled, the technique is found suitable in applications involving irregular structures. In applications involving uniform structures, such as the manipulator system considered here, the finite difference (FD) method is found to be more appropriate. Previous simulation studies of flexible beam systems have demonstrated the relative simplicity of the FD method as compared to the FE method.[11] Real-time simulation is important from a system design and evaluation viewpoint. It provides a characterisation of the system in the real sense as well as allows on-line evaluation of controller designs in real-time. An investigation into the real-time simulation of a flexible manipulator system is thus presented in this paper, on the basis of FD methods, using sequential and parallel processing techniques.

A real-time system is one that has to respond to externally generated input stimuli within a finite and specified period. Despite the vastly increased computing power which is now available there can still be limitations in computing capability of digital processors in real-time control applications for two reasons: (a) sample times have become shorter as greater performance demands are imposed on the system (b) algorithms are becoming more complex as the development of control theory leads to an understanding of methods for optimising system performance. To satisfy these high performance demands, microprocessor technology has developed at a rapid pace in recent years. This is based on (i) processing speed, (ii) processing ability, (iii) communication ability, and (iv) control ability. However, many demanding complex signal processing and control algorithms can not be satisfactorily realised with conventional uni-processor systems. Alternative strategies where high-performance computing and parallel processing (PP) methods are employed, could provide suitable solutions in such applications.[12] Not much work has been reported on such methods in the real-time simulation of flexible manipulator systems.[13–15]

* Department of Automatic Control and Systems Engineering, The University of Sheffield, Mappin St, Sheffield S1 3JD (UK).
† Department of Computer Science, University of Dhaka (Bangladesh).

Parallel processing is a growing technology with the potential to give virtually unlimited computational power, provided that effective parallel software development and application techniques and tools can be produced to be able to take advantage of such potential. 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.[16] It is evidenced that throughout the years, there has been a continuing research effort to understand parallel computation.[17] Such effort has intensified dramatically in the last few years involving various applications, including signal processing, control, artificial intelligence, pattern recognition, computer vision, computer aided design and discrete event simulation.

In a conventional parallel system all the processing elements (PEs) are identical. This architecture can be described as homogeneous. However, many 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 types 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.[18] 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.[19]

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 a number of 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.[20] It is essential to note that in implementing an algorithm on a parallel computation platform, a consideration of (i) the interconnection scheme issues, (ii) the scheduling and mapping of the algorithm on the architecture, and (iii) the mechanism for detecting parallelism and partitioning the algorithm into modules or sub-tasks, will lead to a computational speedup.[21]

For microprocessors with widely different architectures, performance measurements such as MIPS (million instructions per second), MOPS (million operations per second) and MFLOPS (million floating-point operations per second) are meaningless. Of more importance is to rate the performance of a processor on the type of program likely to be encountered in a particular application. The different microprocessors and their different clock rates, memory cycle times etc. all confuse the issue of attempting to rate the processors. In particular, there is an inherent difficulty in selecting microprocessors in real-time signal processing and control applications. The ideal performance of a computing architecture demands a perfect match between processor capability and program behaviour. Processor capability can be enhanced with better hardware technology, innovative architectural features and efficient resource management. From the hardware point of view, current performance varies according to whether the processor possesses a pipeline facility, is microcode/hardwired operated, has an internal cache or internal RAM, has a built-in math co-processor, floating point unit etc. Program behaviour, on the other hand, is difficult to predict due to its heavy dependence on application and run-time conditions. Other factors affecting program behaviour include algorithm design, data structure, language efficiency, programmer skill and compiler technology.[22-25] The work reported in this paper attempts to investigate such issues within the framework of sequential and parallel real-time simulation of a flexible manipulator system.

An investigation into a comparative performance evaluation of hardware and software resources of several architectures in the real-time simulation of flexible robot manipulators is presented in this paper. A single-link flexible manipulator system incorporating hub inertia and payload mass which can bend freedly in the horizontal plane and is stiff in vertical bending and torsion is considered. A fourth-order PDE model of the system obtained through the utilisation of the Lagrange equation and the modal expansion method is considered. A finite-dimensional simulation of the system is developed using an FD discretisation of the dynamic equation of the manipulator. The algorithm thus developed is implemented on a number of general-purpose and special-purpose high-performance architectures incorporating complex instruction set computer (CISC) processors, reduced instruction set computer (RISC) processors and digital signal processing (DSP) devices. The hardware and software resources, capabilities of the processors and characteristics of the algorithm are discussed to explore the matching between the algorithm and the architectures. Finally, a comparison of the results of the implementations, on the basis of real-time computation performance, is made to establish merits of development of fast processing methods in the real-time simulation of flexible manipulator systems.

2. THE FLEXIBLE MANIPULATOR SYSTEM

A schematic representation of a single link flexible manipulator system is shown in Figure 1, where, a manipulator of length l, linear mass density ρ, moment of inertia I_b, hub inertia I_h, carrying a payload of mass M_p with associated inertia I_p is considered. A control torque $\tau(t)$ is applied at the hub of the manipulator by an actuator motor. The angular displacement of the link in the (fixed) POQ—coordinates is denoted by $\theta(t)$. u

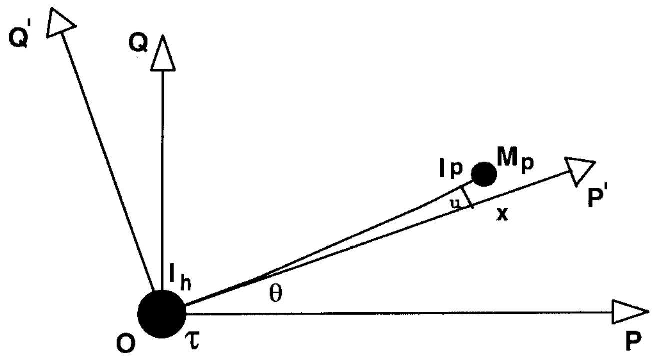

Fig. 1. Schematic representation of the flexible manipulator system.

represents the elastic deflection of the manipulator at a distance x from the hub, measured in the moving coordinates $P'OQ'$. The height of the link is assumed to be much greater than its depth, thus, allowing the manipulator to vibrate (be flexible) dominantly in the horizontal direction (POQ-plane) and be stiff in vertical bending and torsion.

2.1 System model

The flexible manipulator system is modelled as a pinned-free flexible beam, incorporating an inertia at the hub and payload mass at the end-point. The model is developed through the utilisation of Lagrange equation and modal expansion method.[26,27] To avoid the difficulties arising due to time-varying length, the length of the manipulator is assumed to be constant. Moreover, shear deformation, rotary inertia and effect of axial force are neglected.

For an angular displacement θ and an elastic deflection u the total displacement $y(x, t)$ of a point along the manipulator at a distance x from the pinned end can be described as a function of both the rigid body motion $\theta(t)$ and elastic deflection $u(x, t)$;

$$y(x, t) = x\theta(t) + u(x, t)$$

Thus, the net deflection at x is the sum of a rigid body deflection and an elastic deflection. Note that the elastic deflections of the manipulator are assumed to be confined to the horizontal plane only. This is achieved by allowing the manipulator to be dominantly flexible to horizontal deflection. In general, the motion of a manipulator will include elastic deflections in both the vertical and horizontal planes. Motion in the vertical plane due to gravity forces, for example, can cause permanent elastic deflections. This effect is neglected here as the manipulator is assumed to be dominantly flexible in the horizontal plane. This can be ensured by physical construction of the manipulator and by keeping the payload within a certain range.

The dynamic equations of motion of the manipulator can be obtained using the Hamilton's extended principle[28] with the associated kinetic, potential and dissipated energies of the system. This yields the dynamic equation of motion of the manipulator as[29,30]

$$\rho \frac{\partial^2 y(x,t)}{\partial t^2} + EI \frac{\partial^4 y(x,t)}{\partial x^4} = \tau(x,t) \qquad (1)$$

where, E is Young's modulus, I is the second (area) moment of inertia and $\tau(x, t)$ is the applied torque. The product EI represents the flexural rigidity of the manipulator. The associated boundary conditions, at the hub and end-point of the manipulator, and the initial conditions are given by

$$\begin{gathered}
y(0,t) = 0, \\
I_h \frac{\partial^2}{\partial t^2}\frac{\partial y(0,t)}{\partial x} - EI\frac{\partial^2 y(0,t)}{\partial x^2} = \tau(t) \\
M_p \frac{\partial^2 y(l,t)}{\partial t^2} - EI\frac{\partial^3 y(l,t)}{\partial x^3} = 0, \\
I_p \frac{\partial^2}{\partial t^2}\frac{\partial y(l,t)}{\partial x} + EI\frac{\partial^2 y(l,t)}{\partial x^2} = 0 \\
y(x,0) = 0, \\
\frac{\partial y(x,0)}{\partial t} = 0
\end{gathered} \qquad (2)$$

The fourth order PDE in equation (1) with the associated boundary and initial conditions in equation (2) describe the dynamic equations of motion of the flexible manipulator system.

2.2 Simulation algorithm

The PDE in equation (1) describing the dynamics of the flexible manipulator system is of a hyperbolic type and can be classified as a boundary value problem. This can be solved using an FD method. This involves obtaining a set of equivalent difference equations defined by the central FD quotients to replace the PDE. 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 ($i = 0, 1, \ldots, n$; $j = 0, 1, \ldots, m$). A difference equation, corresponding to each point of the grid is, thus, developed. The known boundary conditions are then utilised to eliminate the displacements of the fictitious points outside the defined interval. In this manner, the displacement, $y_{i,j+1}$, of section i of the manipulator at time step $j + 1$ can, thus, be obtained as[31]

$$\begin{aligned}
y_{i,j+1} = {} & -c[y_{i-2,j} + y_{i+2,j}] + b[y_{i-1,j} + y_{i+1,j}] \\
& + a y_{i,j} - y_{i,j-1} + \frac{\Delta t^2}{\rho}\tau(i,j)
\end{aligned} \qquad (3)$$

where, $a = 2 - 6c$, $b = 4c$, $c = EI(\Delta t)^2/\rho(\Delta x)^4$. Using matrix notation, equation (3) can be written in a compact form as

$$\mathbf{Y}_{i,j+1} = \mathbf{A}\mathbf{Y}_{i,j} - \mathbf{Y}_{i,j+1} + \mathbf{BF} \qquad (4)$$

where $\mathbf{Y}_{i,k}$ represents the displacement of grid points $i = 1, \ldots, n$ of the manipulator at time step $k = j + 1$, j, $j - 1$, $\mathbf{A}$ is a constant $n \times n$ matrix entries of which depend on the flexible manipulator specification and the number of sections the manipulator is divided into, **F** is an $n \times 1$ matrix known as the forcing matrix and **B** is a scalar constant related to the time step Δt and mass per unit length, ρ, of the flexible manipulator;

$$\mathbf{Y}_{i,j+1} = \begin{bmatrix} y_{1,j+1} \\ y_{2,j+1} \\ \vdots \\ y_{n,j+1} \end{bmatrix}$$

$$\mathbf{Y}_{i,j} = \begin{bmatrix} y_{1,j} \\ y_{2,j} \\ \vdots \\ y_{n,j} \end{bmatrix}$$

$$\mathbf{Y}_{i,j-1} = \begin{bmatrix} y_{1,j-1} \\ y_{2,j-1} \\ \vdots \\ y_{n,j-1} \end{bmatrix}$$

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

$$\mathbf{B} = (\Delta t)^2/\rho, \quad \mathbf{F}^T = [\tau(1,j),\ \tau(2,j), \dots, \tau(n,j)].$$

The element values m_1, m_2, m_3, $m_{11}, \dots, m_{24}$ in matrix **A** depend on the boundary conditions of the manipulator. Equation (4) gives, as a general solution of the PDE, the displacement of section i of the manipulator at time step $j+1$ that can be implemented on a digital processor easily. Note in equation (4) that the displacement of the hub-end of the manipulator ($i = 0$) has not been included, as this is already known from the boundary conditions in equation (2). For the algorithm to be stable the iterative procedure in equation (3) for every grid-point is to converge to a solution. A necessary and sufficient condition for the algorithm to be stable is given by $0 \leq c \leq 0.25$.[31]

3. HARDWARE

The hardware utilised comprises three general-purpose processors namely, an Intel 80386DX (40 MHz) (386DX), an Intel 80486DX2 (50 MHz) (486DX) and a Texas Instruments TMS390S10 SAPRC processor, three special-purpose parallel PEs namely, an Inmos T805 (T8) transputer, an Intel 80860 (i860) RISC processor and a Texas Instruments TMS320C40 (C40) DSP device and four different heterogeneous and homogeneous parallel architectures incorporating the parallel PEs. 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 features of the PEs and the parallel architectures. Table I shows some of the comparative features of the uni-processor architectures utilised. Other features of these with those of the parallel architectures are described below.

3.1 General-purpose processors

Among the general-purpose processors, the 386DX is a CISC processor, possessing a 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. 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 programs. This mode enables a single 386 microprocessor to divide its memory into many virtual machines, each machine as a separate 8086 microprocessor. This implies that the 386 has a multitasking facility. The processor, however, does not have any internal or external math co-processor and floating point unit facility. Moreover, it does not have internal cache or internal memory.[32]

The 486DX is a general-purpose CISC processor. From a software standpoint, the 486 is distinguished from

Table I. Comparative features of the processors

Processors	386DX	486DX2	SPARC	T805	i860	C40
Manufacturer	Intel	Intel	Texas Inst.	Inmos	Intel	Texas Inst.
Type	32-bit CISC	32-bit CISC	32-bit RISC	32-bit RISC	64-bit RISC	32-bit DSP
Clock speed	40 MHz	50 MHz	50 MHz	25 MHz	40 MHz	40 MHz
Internal Cache/RAM	None	8 kbytes cache (unified)	—	4 kbytes RAM	8 kbytes data & 4 kbytes instruction	8 kbytes data & 512 bytes instruction
MIPS/MFLOPS/MOPS	12 MIPS	54 MIPS	—	20 MIPS, 2.4 MFLOPS	40 MIPS, 80(1) & 60(2) MFLOPS	275 MOPS & 40 MFLOPS
Transistors	275,000	1,200,000	—	—	Over one million	—

(1): Single precision.
(2): Double precision.

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 streamlined hardware design, tighter silicon design rules, integral math co-processor, instruction pipelining, built-in floating point unit and internal memory cache. The streamlined hardware design (particularly its pipelining) means that the 486 can process faster than a 386 microprocessor when the two are operating at the same clock speed. Thus, a 33 MHz 486 is faster than a 33 MHz 386. In 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. The 486 incorporates all the necessary coprocessor circuitry on the same slice of silicon. This internal coprocessor nearly doubles the performance of the processor.

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 a 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 and 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.

3.2 Parallel processing elements

The transputer is a high-performance microprocessor designed to facilitate inter-process 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 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 of 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. 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. Among these the T805 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 speeds of 20 Mbits/sec, achieving data rates of up to 1.7 Mbytes/sec uni-directionally 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.[33,34]

The i860 is a high-performance 64-bit vector processor with 40 MHz clock speed, a peak integer performance of 40 MIPS, 8 kbytes data cache and 4 kbytes instruction cache and is capable of 80 MFLOPS. The Intel i860 has been designed for numerically and vector intensive applications.[35] In particular, its high throughput is achieved from a combination of RISC design techniques, pipeline processing units, wide data paths and large on-chip caches. On a single chip the architecture supports integer, floating point and graphics operations and incorporates memory-management unit, 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. 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, which equals 25 nsec for a 40 MHz clock rate.[25] The i860 is an ideal candidate for integration into highly parallel computer environments with high computational performance, modularity and low real-estate requirements.

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 cache and is capable of 275 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.[36,37]

3.3 Homogeneous architectures

The homogeneous architectures considered include a network of T8s and a network of C40s. The homogeneous architecture of T8s comprises a network of T8s resident on a Transtech TMB08 motherboard. The root T8 incorporates 2 Mbytes of 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. A pipeline topology is utilised for this architecture. This is shown in Figure 2. This topology has been utilised as it is simple to realise. Moreover, it

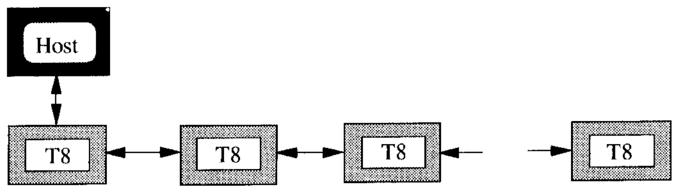

Fig. 2. Operational configuration of the homogeneous architecture incorporating a network of T8s.

reflects the structure of the algorithm considered in this study.

The homogeneous architecture of C40s comprises a network of C40s resident on a Transtech TDM410 motherboard and a TMB08 motherboard incorporating a T8 as a root processor. The T8 possesses 1 Mbyte local memory and communicates with the TDM410 (C40s network) via a 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. A pipeline topology is utilised for this architecture. Figure 3 shows the topology of the homogeneous architecture of C40s. The T8 in the network is used as the root processor providing 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 is well reflected as a linear form.[33]

3.4 Heterogeneous architectures

Two heterogeneous parallel architectures, namely, an integrated i860 and T8 system and an integrated C40 and T8 system, are considered in this study. The i860 + T8 architecture comprises a TMB16 motherboard 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 operational configuration of the architecture is shown in Figure 4. The i860 and the T8 processors communicate with each other via the shared memory.

The operational configuration of the C40 + T8 architecture is shown in Figure 5 in which the T8 is used both as the root processor providing an interface with the host, and as an active PE. The C40 and the T8 communicate with each other via serial-to-parallel or parallel-to-serial links.

Note in the above that the SPARC, 486DX and

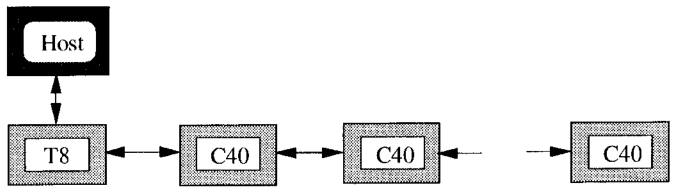

Fig. 3. Operational configuration of the homogeneous architecture incorporating a network of C40s.

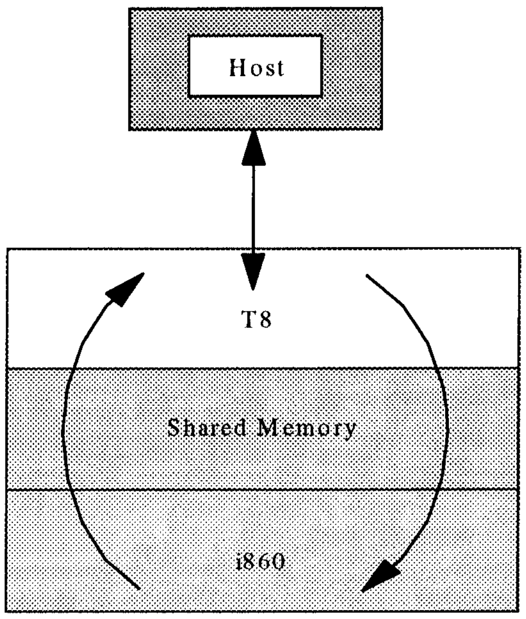

Fig. 4. Operational configuration of the heterogeneous architecture incorporating an i860 and a T8.

386DX are not commonly used in real-time implementations. These are utilised here, however, for benchmarking and comparison of various features with the DSP and transputer-based architectures. Moreover, note that, wherever indicated, the host is utilised for developing and downloading programmes. Thus, the host does not take part in the real-time implementation process.

4. SOFTWARE SUPPORT

Software support is needed for the development of efficient programs in high-level languages. The ideal performance of a computer system demands a perfect match between machine capability and program behaviour. Program performance is the turnaround time, which includes disk and memory access, input and output activities, compilation time, operating system overhead and CPU time. To shorten the turnaround time, one can reduce all these time factors.[25] Minimising the run-time memory management within the program and selecting

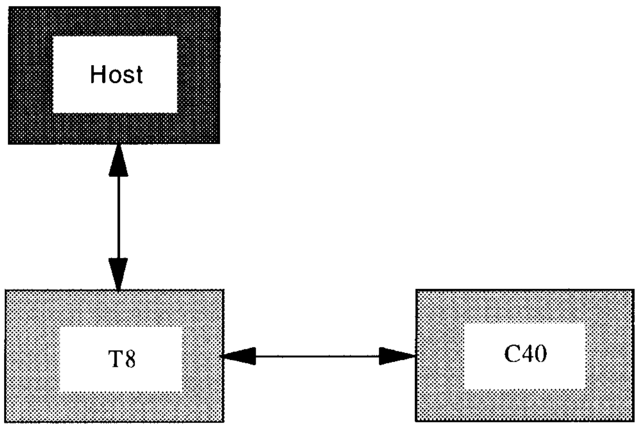

Fig. 5. Operational configuration of the heterogeneous architecture incorporating a C40 and a T8.

an efficient compiler for a specific computation demand, could enhance the performance.

Compilers have a significant impact on the performance of the system. This is not to say that any particular high-level language dominates another. Most languages have advantages in certain computational domains. The compiler itself is critical to the performance of the system, since the mechanism for taking a high-level description of the application and transforming it into hardware dependent machine language differs from one compiler to another. Identifying the foremost compiler for the application in hand is, therefore, especially challenging due to unpredictable run-time behaviour of the compilers and memory management capability.

In signal processing and control applications, it is especially important to select a suitable programming language that supports highly numerical computations. For instance, the Occam compiler is more straight forward for parallel processing, although it is not efficient in numerical calculations.[38,39] Such features are important to consider in the selection of a compiler for a particular application. The algorithm considered was coded in high-level languages consisting of Inmos ANSI C, 3L Parallel C and Borland C as appropriate for the hardware used. Among these the ANSI C and Parallel C programming languages support PP. In the process of implementing the algorithm on different platforms, the coding of the algorithm was kept to a similar form with the different compilers.

5. PRACTICAL ISSUES IN REAL-TIME PARALLEL 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.

Parallelism is beneficial when it successfully yields higher performance with reasonable hardware and software cost. Whether parallelism is worthwhile depends on the application. 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.

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;

$$S_N = T_1/T_N$$

The theoretical maximum speedup 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 idle at times due to conflicts over memory access, communication delays, algorithm inefficiency and mapping for exploiting the natural concurrency in a computing problem.[40] But, in some cases, the speedup can be obtained above the ideal speedup, due to anomalies in programming, compilation and architecture usage. For example, a single-processor system may store all its data off-chip, whereas the multi-processor 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

$$E_N = (S_N/N) \times 100\% = (T_1/NT_N) \times 100\%$$

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 a different number of processors. It has also been illustrated that the value of efficiency is directly related to the granularity of the system.[41]

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 the suitability of an algorithm evaluated. When several processors are required to work co-operatively on a single task, one expects a frequent exchange of data among sub-tasks 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-processor 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. 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.[21]

There are three different problems to be considered in implementing signal-processing and control algorithms on PP systems; (1) identifying parallelism in the algorithm, (2) partitioning the algorithm into sub-tasks, and (3) allocating the tasks to processors. These include inter-processor communication, 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 communication demand of the task. Performance benefits of parallel architectures strongly depend on these ratios.[41,42]

Task granularity can also be viewed in terms of computation time per task. When this is large, it is a coarse-grained task implementation. When it is small, it is a fine-grained 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 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.[43]

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 this type of algorithms, a vector processor will, principally, be expected to perform better. Moreover, if a large amount of data is to be handled, the performance will be further 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 can therefore be detrimental to the performance of the computing platform in both homogeneous and heterogeneous architectures.

To investigate the performance of the computing platforms in the real-time implementation of the algorithm considered in this study, the features discussed above are considered in the process of partitioning the algorithm so that the capabilities of the parallel hardware platforms are efficiently exploited and the load distribution is balanced so as to minimise inter-processor communications.

6. IMPLEMENTATIONS AND RESULTS

In the experimental investigations to follow an aluminium type flexible manipulator of physical characteristics shown in Table II was considered. In implementing the algorithm, the manipulator was divided into 19 sections and a sample time of $\Delta t = 0.218$ ms was utilised. To investigate the real-time performance of the simulation algorithm, a bang-bang torque of amplitude 0.1 Nm was utilised.

Table II. Parameters of the flexible manipulator.

Parameter	value
Length	960 mm
Depth	3.2004 mm
Height	19.230 mm
Area of cross section	6.1544×10^{-5} m^2
Mass density per volume	2710 kgm^{-3}
Young's Modulus	7.11×10^{10}
Area moment of inertia	5.1924×10^{-11} m^2
Hub inertia	5.86×10^{-4} kgm^2
Manipulator inertia	0.0495 kgm^2

6.1 Inter-processor communication

Inter-processor communication in a PP network is an important factor in determining the real-time performance of the network in implementing an algorithm. The inter-processor communication techniques involved are (a) C40-C40: parallel communication, (b) T8-T8: serial communication, (c) i860-T8: shared memory communication and (d) T8-C40: parallel to serial communication.

To investigate the performance of the inter-processor communication links, a 4000-point 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 cases of the C40 to T8 and the C40 to C40 communications, the speed of single lines of communication was 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. The performance of inter-processor communication links is shown in Figure 6. 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 evident that as compared to serial communication, parallel communication offers a substantial advantage. The relatively slower performance of the single line of communication is due to the utilisation of a single bi-directional line of communication in which, in addition to the sequential nature of the process of sending and receiving data, extra time is required for altering the direction of the link (data flow). Moreover, there is an initialisation delay for each communication performed which for small amounts of data becomes relatively a significant proportion of the communication time. 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, an additional penalty is paid during the transformation of data from serial to parallel and vice versa.

6.2 Simulation algorithm

The simulation algorithm considered here is of the RI type. In implementing such an algorithm, a sequential vector processor will, principally, be expected to perform better and faster than any other processor. Moreover, since a large amount of data is to be handled for computation in this algorithm, the performance will further be enhanced if the processor has more internal data cache, instruction cache and/or a built-in math

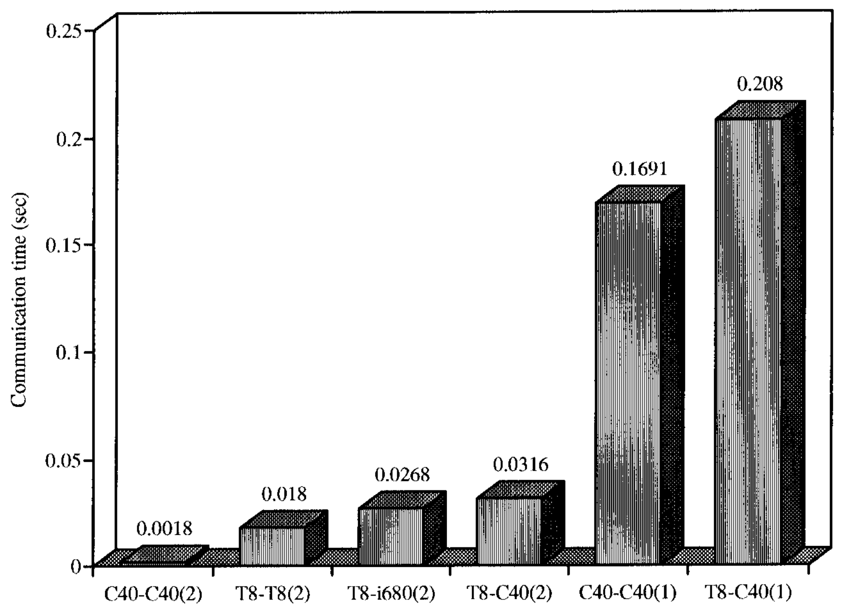

Fig. 6. Speed of inter-processor communication links; (1) single-line; (2) double-line.

coprocessor. In implementing the algorithm on a PP platform, the tasks could be distributed uniformly among the PEs. However, this will require a large amount of communication between the PEs. To calculate the deflection of any one segment of the flexible arm, for instance, information on the deflection of two backward and two forward segments will be required. Thus, each PE will need to send two and receive two data points in every iteration. This heavy communication time can, therefore, be a detriment to the performance of the computing platform in both homogeneous and heterogeneous architectures.

To evaluate the performance of the computing platforms, the algorithm was implemented as a sequential process on uni-processor based architectures. In case of the multi-processor based architectures, on the other hand, the algorithm was partitioned and distributed uniformly among the PEs taking account of the communication load. Thus, with architectures incorporating two PEs the algorithm was partitioned by allocating ten sections (1–10) to one of the PEs and the remaining nine sections to the other PE. In case of architectures incorporating three PEs, it was found that the communication time required with the second (middle) PE was equivalent to the computation time of about two sections. Thus, to obtain enhanced performance, the algorithm was partitioned by allocating seven sections (1–7) to the first PE, five sections (8–12) to the second PE and the remaining seven sections (13–19) to the third PE. The real-time performance of the uni-processor and multi-processor computing domains, thus obtained, are shown in Figures 7 and 8, respectively. The required real-time (Reqrd R/T) indicates the actual time needed to implement the algorithm. This is given by the product of the sample time Δt and the total number of iterations.

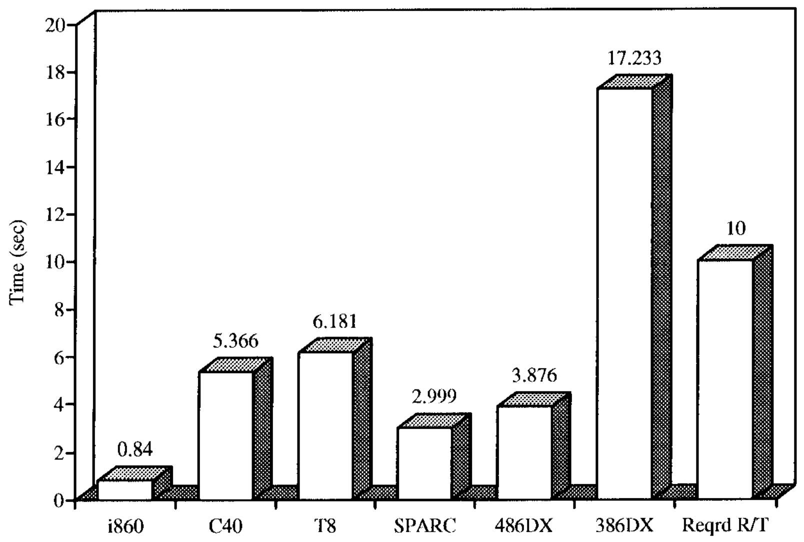

Fig. 7. Execution times of the uni-processor architectures in implementing the algorithm.

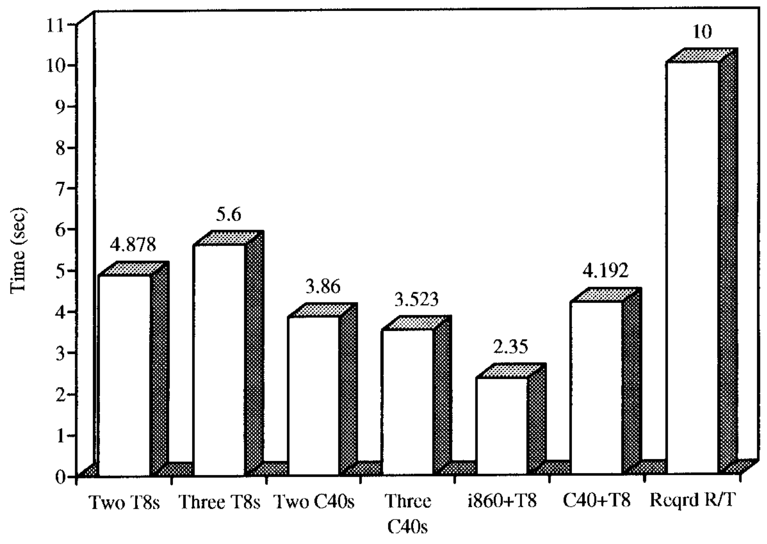

Fig. 8. Execution times of the multi-processor architectures in implementing the algorithm.

It is noted in Figures 7 and 8 that the superscalar i860 RISC processor has performed the fastest and the 386DX machine the slowest of the processors used. The algorithm, as noted 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 multi-processor architectures. The C40 does not have such vector processing resources making it about 6.39 times slower than the i860. This implies that the C40 is not performing well in a situation where the algorithm is matrix type and where extensive run time memory management is involved. A single transputer, on the other hand, has performed about 7.36 times slower than the i860.

The SPARC processor and the 486DX appear to have achieved similar performances, the SPARC being slightly faster due to its RISC processor. Both these processors have performed better than a C40 and a T8. The 386DX performance was the slowest of all the processors. This, as compared to the 486DX machine, is mainly due to the floating point operations which are evaluated in software rather than using dedicated hardware since this processor does not have maths coprocessor. Moreover, it does not have cache or internal memory making it slower to handle calculation of large amounts of data.

Among the multi-processor architectures, the main factor influencing the C40 + T8 network to perform slightly slower than the network of two C40s is the serial to parallel communication link utilised and the slower computation performance of the T8 in this architecture. Due to the shared memory communication overhead and incorporation of the slower (T8) processor, the i860 + T8 architecture has achieved longer execution time than a single i860 processor. The performance of the i860 + T8 as compared to the C40 + T8 and the two C40s, however, is significantly better. It is noted in Figures 7 and 8, that, all the architectures except the 386DX has achieved the required real-time performance.

A comparison of the execution times of the T8 and C40 based homogeneous architectures in Figure 8 reveals that the C40 networks have achieved better performances than the T8 networks. Due to an increase in the communication overheads, the performance achieved with three T8s is lower than that with two T8s. In contrast, the performance achieved with three C40s is slightly better than that with two C40s. This is due to the high speed parallel communication links of the C40s network. These can further be demonstrated in terms of speedup and efficiency of the architectures.

The corresponding speedup and efficiency of the execution time for the network of T8s is shown in Table III. As discussed earlier, although an increase in the number of transputers leads to a decrease in the execution time, but the reduction is not linear throughout. This implies that, with an increase in the number of PEs or transformation of the algorithm from course-grain to fine-grains, the communication demand increases. This is further evidenced in Table III which shows a non-linear speedup and efficiency. This could be due to the nature of the algorithm as compared to the capabilities of this hardware which appears not to cope well with the communication overhead and run-time memory management problem.

Table IV shows the corresponding speedup and efficiency of the network of C40s in implementing the algorithm. It is noted that the execution time speedup and efficiency achieved with the network of C40s is not linear. As compared to the network of T8s, the network of C40s has achieved better performance. However, due to a mismatch between the nature of the algorithm with

Table III. Speedup and efficiency of the network of T8s in implementing the simulation algorithm.

Number of T8s	Two	Three
Speedup	1.268	1.104
Efficiency	63.4%	36.8%

Table IV. Speedup and efficiency of the network of C40s in implementing the simulation algorithm.

Number of C40s	Two	Three
Speedup	1.390	1.523
Efficiency	69.5%	50.8%

the architecture, communication overhead and run-time memory management problem, the network of C40s has not achieved an outstanding speedup and efficiency.

7. CONCLUSION

An investigation into the real-time performance evaluation of a number of general-purpose and special-purpose uni-processor and multi-processor parallel architectures in implementing an FD simulation algorithm of a flexible manipulator system has been presented. The inter-processor communication speed for four different homogeneous and heterogeneous architectures have been investigated and presented. Partitioning and mapping, granularity and regularity of the algorithm 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 the algorithm. Real-time performance has been achieved with all the architectures except with the 386DX. Special features, such as vector processing resources in an i860 vector processor, have been exploited to result in relatively better performance with the i860 and the i860 + T8 among the uni-processor and the multi-processor architectures respectively in implementing the algorithm.

For PP, performance measures such as MIPS and MFLOPS of the PEs are meaningless. 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. It has been demonstrated that there is generally a mismatch between the hardware requirements of an algorithm and the hardware resources of the architecture leading to a disparity in their relative performance. Therefore, to fully exploit the architectures 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.

REFERENCES

1. J.-N. Aubrun, "Theory of the structures by low-authority controllers" *J. Guidance and Control* **3,** 441–451 (1980).
2. M.J. Balas, "Feedback control of flexible systems" *IEEE Transaction on Automatic Control* **AC-23,** 673–679 (1987).
3. S. Omatu and J.H. Seinfeld, "Optimal sensor actuator locations for linear distributed parameter systems" *Proceedings of 4th IFAC Symposium on Control of Distributed Parameter Systems* (Los Angeles, 1986) pp. 215–220.
4. R.H. Cannon and E. Schmitz, "Initial experiments on the end-point control of a flexible one-link robot" *Int. J. Robotic Research* **3,** 62–75 (1984).
5. F. Harishima and T. Ueshiba, "Adaptive control of flexible arm using end-point position sensing" *Proceedings of Japan-USA Symposium of Flexible Automation* (Osaka, 1986) pp. 225–229.
6. P.C. Hughes, "Space structure vibration modes: How many exists? Which are important" *IEEE Control Systems Magazine* **7,** 22–28 (1987).
7. V. Feliu, K.S. Rattan and H.B. Brown Jr., "Modelling and control of single link flexible arms with lumped masses" *Transaction of ASME Journal of Dynamic Systems, Measurement and Control* **114,** 59–69 (1992).
8. K. Oosting and S.L. Dickerson, "Simulation of a high-speed lightweight arm" *Proceedings of the IEEE International Conference on Robotics and Automation,* (Philadelphia, 1988) pp. 494–496.
9. M. Dado and A.H. Soni, "A generalized approach for forward and inverse dynamics of elastic manipulators" *Proceedings of IEEE Conference on Robotics and Automation* (San Francisco, 1986) pp. 359–364.
10. P.B. Usoro, R. Nadira and S.S. Mahil, "A finite element/ Lagrange approach to modelling lightweight flexible manipulator" *Transaction of ASME Journal of Dynamic Systems Measurement and Control* **108,** 198–205 (1984).
11. P.K. Kourmoulis, "Parallel processing in the simulation and control of flexible beam structure system" *PhD thesis* (The University of Sheffield, UK, 1990).
12. M.O. Tokhi and M.A. Hossain, "Real-time active control using sequential and parallel processing methods" In M.J. Crocker and N.I. Ivanov (eds.), *Proceedings of the fourth International Congress on Sound and Vibration* **1** (St Petersburg, 1996) pp. 391–398.
13. M.O. Tokhi and A.K.M. Azad, "Real-time finite difference simulation of a single-link flexible manipulator system incorporating hub inertia and payload" *Proceedings of IMechE-I: Journal of Systems and Control Engineering* **209,** 21–33 (1995).
14. M.O. Tokhi, M.A. Hossain and A.K.M. Azad, "Processing requirements in the real-time simulation of flexible manipulator systems" *Proceedings of the Institute of Acoustics* **17** (Part 4), 231–238 (1995).
15. M.O. Tokhi, M.A. Hossain and A.K.M. Azad, "Signal processing and parallel processing in real-time simulation of a flexible manipulator system" **In:** P.J. Fleming and L. Boullart (eds.), *AARTC-95: Preprints of IFAC/IFIP Workshop on Algorithms* and *Architectures for Real-time Control* (Ostend, 1995) pp. 53–58.
16. P.J. Denning, "Parallel computing and its evolution" *Communications of the ACM* **29,** 1163–1167 (1986).
17. R.W. Hocney and C.R. Jessope, *Parallel Computers* (Hilger Publishing Co., Bristol, 1981).
18. G.M. Megson, "Practical steps towards algorithmic engineering" *Colloquium on Applications of Parallel and Distributed Processing in Automation and Control* IEE Digest 1992/204 (London, 1992).
19. M.J. Baxter, M.O. Tokhi and P.J. Fleming, "Parallelising algorithms to exploit heterogeneous architectures for real-time control systems" *Proceeedings of IEE Control-94 Conference* **2** (Coventry, 1994) pp. 1266–1271.
20. T.P. Crummey, D.J. Jones, P.J. Fleming and W.P. Marnane, "A hardware scheduler for parallel processing in control applications" *Proceedings of IEE Control-94 Conference* **2** (Coventry, 1994) pp. 1098–1103.
21. D.P. Agrawal, V.K. Janakiram and G.C. Pathak, "Evaluating the performance of multicomputer configuration" *Computer,* 23–37 (May, 1986).
22. A.J. Anderson, "A performance evaluation of microprocessors, DSPs and the transputer for recursive parameter estimation" *Microprocessors and Microsystems* **15,** 131–136 (1991).
23. P.C. Ching and S.W. Wu, "Real-time digital signal

processing system using a parallel architecture" *Microprocessors and Microsystems* **13,** 653–658 (1989).
24. Z. Cvetanvoic, "The effects of problem partitioning, allocation, and granularity on the performance of multiple processor systems" *IEEE Transactions on Computers* **C-36,** 421–432 (1987).
25. K. Hwang, *Advanced Computer Architecture – parallelism scalability programmability* (McGraw-Hill, USA, 1993).
26. G.G. Hastings and W.J. Book, "A linear dynamic model for flexible robotic manipulator" *IEEE Control Systems Magazine* **7,** 61–64 (1987).
27. V.V. Korolov and Y.H. Chen, "Controller design robust to frequency variation in a one-link flexible robot arm" *J. Dynamic Systems, Measurement and Control* **111,** 9–14 (1989).
28. L. Meirovitch, *Analytical Methods in Vibrations* (Macmillan, New York, 1967).
29. M.O. Tokhi and A.K.M. Azad, "Modelling of a single-link flexible manipulator system: Theoretical and practical investigations" *Robotica* **14,** 91–102 (1996).
30. M.O. Tokhi, A.K.M. Azad, A.S. Morris and M.A. Hossain, "Modelling and simulation of a flexible manipulator system" *IEE Control-94 Conference* **2** (Coventry, 21–24 March 1994) pp. 1400–1405.
31. A.K.M. Azad, "Analysis and design of control mechanisms for flexible manipulator systems" *PhD thesis* (The University of Sheffield, UK, 1994).
32. W.L. Rosch, *Hardware Bible* (Brady Publishing, Indianapolis, 1993).
33. G.W. Irwin and P.J. Fleming, *Transputers in Real-time Control* (John Wiley, England, 1992).
34. Transtech Parallel Systems Ltd, *Transtech Parallel Technology* (Transtech Parallel Systems Ltd., UK, 1991).
35. Transtech Parallel Systems Ltd, *Transtech Parallel Technology* (Transtech Parallel Systems Ltd., UK, 1992).
36. A. Brown, "DSP chip with no parallel" *Electronics World + Wireless World* 878–879 (October, 1991).
37. Texas Instruments, *TMS320 User's Guide* (Texas Instruments, USA, 1991).
38. G. Bader and E. Gehrke, "On the performance of transputer networks for solving linear systems of equation" *Parallel Computing* **17,** 1397–1407 (1991).
39. M.O. Tokhi, M.A. Hossain, M.J. Baxter and P.J. Fleming, "Real-time performance evaluation issues for transputer networks" *Proceedings of WoTUG-18: 18th International Conference of the World Occam and Transputer User Group* (Manchester, 1995) pp. 139–150.
40. K. Hwang and F.A. Briggs, *Computer Architecture and Parallel Processing* (McGraw-Hill, California, 1985).
41. H.S. Stone, *High Performance Computer Architecture* (Addison Wesley, USA, 1987).
42. L.P. Maguire, "Parallel architecture for Kalman filtering and self-tuning control" *PhD thesis* (The Queen's University of Belfast, UK, 1991).
43. G.D.F. Nocetti and P.J. Fleming, "Performance studies of parallel real-time controllers" *Proceedings of IFAC Workshop on Algorithms and Architectures for Real-time Control* (Bangor, 1991) pp. 249–254.