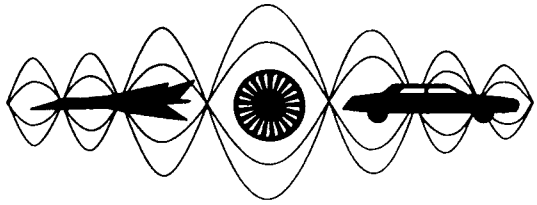


FIFTH INTERNATIONAL CONGRESS ON SOUND AND VIBRATION
DECEMBER 15-18, 1997
ADELAIDE, SOUTH AUSTRALIA

Specialist Keynote Paper

HIGH-PERFORMANCE REAL-TIME COMPUTING METHODS

M. O. Tokhi §, D. N. Ramos-Hernandez § and M. A. Hossain ‡

§ Department of Automatic Control and Systems Engineering,
The University of Sheffield, UK.
‡ Department of Computer Science, University of Dhaka, Bangladesh.

ABSTRACT

The performance demands in practical realisation of signal processing and control strategies have motivated a trend towards utilisation of complex algorithms. This, in turn, has resulted in a resurgence in the development of high-performance processors to make real-time implementation of such algorithms feasible in practice. However, due to inefficient mapping of algorithms on processors, to take account of the computing capabilities of processors in relation to the computing requirements of the application, such a goal can still be difficult to reach. This paper presents an investigation into the development of sequential and parallel computing methods for real-time signal processing and control. Several algorithms encountered in acoustics and vibration applications are considered. These are implemented on a number of high-performance processors including the TMS320C40 parallel digital signal processing device, the Intel i860 vector processor and the Inmos T805 transputer. A comparative assessment of the performance of the processors in implementing the algorithms, revealing the capabilities of the processors in relation to the nature of the algorithms, is presented. This is used as the basis of development of new performance metrics and task to processor mapping strategies for parallel architectures. The performance metrics and mapping strategies thus developed are verified by implementing the algorithms on a number of homogeneous and heterogeneous parallel architectures.

Keywords: Heterogeneous architectures, homogeneous architectures, parallel processing, real-time signal processing and control, sequential processing.

1. INTRODUCTION

A real-time system can be regarded as one that has to respond to externally-generated 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 signal processing and 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.

Digital signal processing (DSP) devices are designed in hardware to perform concurrent add and multiply instructions and execute irregular algorithms efficiently, typically finite-impulse response (FIR) and infinite-impulse response (IIR) filter algorithms. Vector processors are designed to efficiently process regular algorithms involving matrix manipulations. However, many demanding complex signal processing and control algorithms can not be satisfactorily realised with conventional computing methods. Alternative strategies where high-performance sequential and parallel computing methods are employed, could provide suitable solutions in such applications (Tokhi and Hossain, 1996).

Parallel processing (PP) is a subject of widespread interest for real-time signal processing and control. In a conventional parallel system all the processing elements (PEs) are identical. This architecture can be described as homogeneous. However, many 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. To exploit the heterogeneous nature of the hardware it is required to identify the heterogeneity of the algorithm so that a close match can be forged with the hardware resources available (Baxter *et al.*, 1994).

For sequential and parallel processing with widely different architectures and different PEs, performance measurements such as million instructions per second (MIPS), million operation per second (MOPS) and million floating-point operations per second (MFLOPS) of the PEs are meaningless. Of more importance is to rate the performance of each architecture with its PEs on the type of program likely to be encountered in a typical application. The different architectures and their different clock rates, memory cycle times of the PEs, inter-processor communication speed, optimisation facility and compiler performance etc. all confuse the issue of attempting to rate the architecture. This is an inherent difficulty in selecting a parallel architecture, for better performance, for algorithms in signal processing and control system development applications. The ideal performance of a parallel architecture demands a perfect match between the capability of the architecture and the program behaviour. Capability of the architecture can be enhanced with better hardware technology, innovative architectural features and efficient resources management. In contrast, program behaviour is difficult to predict due to its heavy dependence on application and run-time conditions. Moreover, there are many other factors that influence program behaviour. These include algorithm design, partitioning and mapping of an algorithm, inter-processor communication, data structures, language efficiency, programmer skill, and compiler technology (Hwang, 1993; Tokhi *et al.*, 1995). The purpose of this investigation is to provide a coherent analysis and evaluation of the performance of sequential and parallel

computing techniques for real-time signal processing and control applications. The paper is structured as follows

Section 2 provides a brief description of the computing platforms and software resources utilised. Section 3 presents a brief description of the signal processing and control algorithms utilised in this work. Section 4 presents performance metrics in sequential and parallel processing. Section 5 gives a generalised task to processor allocation in parallel architectures. Section 6 presents results and discussion of implementation of the algorithms on the processors and performance evaluation of the computing platforms. The paper is finally concluded in Section 7.

2. HARDWARE AND SOFTWARE PLATFORMS

The hardware architectures utilised incorporate the Intel 80860 (i860) RISC processor, the Texas Instruments TMS320C40 (C40) DSP device and the Inmos T805 (T8) transputer. These are used as uni-processor architectures as well as in devising various heterogeneous and homogeneous parallel architectures. These are briefly described below.

The i860 is a 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. This is a superscalar RISC processor (Hwang, 1993). The C40 is a 32-bit DSP device with 40 MHz clock speed, 8 kBytes on-chip RAM, and 512 bytes on-chip instructions cache, and is capable of 275 MOPS and 40 MFLOPS. The device possesses six parallel high-speed communication links for inter-processor communication (Texas Instruments, 1991a).

The T8 is a general-purpose medium-grained 32-bit Inmos parallel PE with 25 MHz clock speed, yielding up to 20 MIPS performance, 4 kBytes on-chip RAM and is capable of 4.3 MFLOPS. The T8 is a RISC processor possessing an on-board 64-bit floating-point unit and four serial communication links (Transtech Parallel Systems Ltd, 1991).

The homogeneous architectures considered include a network of C40s and a network of T8s. A pipeline topology is utilised for these architectures, on the basis of the algorithm structure, which is simple to realise and is well reflected as a linear farm (Irwin and Fleming, 1992). 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 C40s communicate with each other via parallel communication links. The homogeneous architecture of T8s comprises a network of T8s resident on a Transtech TMB08 motherboard. The serial links of the processors are used for communication with one another.

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 an IBM compatible PC, A/D and D/A conversion facility, a TMB16 motherboard and a TTM110 board incorporating a T8 and an i860. The i860 and the T8 processors communicate with each other via the shared memory. In the C40+T8 architecture 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.

The compilers used consist of the Inmos ANSI C (for T8), Portland Group ANSI C (for i860) 3L Parallel C (for C40 and T8) and Occam (for T8). For the implementations involving the T8, as will be noted later, the ANSI C compiler is used in investigations involving performance evaluations of the hardware architectures in implementing the algorithms

whereas the 3L Parallel C and Occam are used in investigations involving the performance evaluation of the compilers.

3. ALGORITHMS

The algorithms considered in this investigation are briefly described in this section.

3.1 The fast Fourier transform

Fast Fourier transform (FFT) constitutes a class of algorithms devised for the efficient computation of discrete Fourier transforms (DFT) of sequences. A real periodic discrete-time signal $x(n)$ of period N can be expressed as a weighted sum of complex exponential sequences. Since sinusoidal sequences are unique only for discrete frequencies from 0 to 2π, the expansion contains only a finite number of complex exponentials. The complex DFT series $X(k)$ of $x(n)$ can be written as

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

where $W_N = \exp(-j2\pi/N)$. Using the divide-and-conquer approach, equation (1) can be simplified as

$$X(p,q) = \sum_{l=0}^{L-1} \left\{ W_N^{lq} \left[\sum_{m=0}^{M-1} x(l,m) W_M^{mq} \right] \right\} W_L^{lp}. \tag{2}$$

Equation (2) involves the computation of DFT of sequences of lengths M and L respectively. In this manner, the total computation will be half of that of a direct DFT computation (Ifeachor and Jervis, 1993).

3.2 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

$$r_{xy}(l) = \sum_{n=-\infty}^{\infty} x(n) y(n-l); \quad l = 0, \pm 1, \ldots \tag{3}$$

or, equivalently, as

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

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 in equations (3) and (4) each yield identical cross-correlation sequences (Ifeachor and Jervis, 1993).

3.3 LMS filter

The least mean square (LMS) adaptive filter algorithm is based on the steepest descent method where the weight vector is updated according to (Widrow *et al.*, 1975)

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

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

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

where y_k is the current contaminated signal sample. The weights obtained by the LMS algorithm are adjusted so that the filter learns the characteristics of the signal leading to a convergence of the weights.

3.4 RLS filter

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

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

where Θ and Ψ represent the parameter vector and the observation matrix of the filter respectively. The new parameter vector is given by

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

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

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

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

3.5 The DOT algorithm

The DOT algorithm is a basic linear algebraic operation, which incorporates floating-point add and multiply operations and takes one processor cycle to finish. It is assumed that each memory operation, including read and write, takes one processor cycle to finish (Sun and Gustafson, 1991). The DOT algorithm is given as

$$y(i) = a + b(i) \times c(i)$$

where a, b and c represent real numbers.

3.6 Simulation and active vibration control of a flexible beam structure

Consider a cantilever beam system with a force $F(x,t)$ applied at a distance x from its fixed (clamped) end at time t. This will result in a deflection $y(x,t)$ of the beam from its stationary position at the point where the force has been applied. In this manner, the governing dynamic equation of the beam is given by

$$\mu^2 \frac{\partial^4 y(x,t)}{\partial x^4} + \frac{\partial^2 y(x,t)}{\partial t^2} = \frac{1}{m} F(x,t) \tag{5}$$

where, μ is a beam constant and m is the mass of the beam. Discretising the beam into a finite number of sections (segments) of length Δx and considering the deflection of each section at time steps Δt using the central finite difference (FD) method, a discrete approximation to equation (5) can be obtained as (Tokhi and Hossain, 1994)

$$Y_{k+1} = -Y_{k-1} - \lambda^2 S Y_k + \frac{(\Delta t)^2}{m} F(x,t) \tag{6}$$

where $\lambda^2 = \mu^2 (\Delta t)^2 / (\Delta x)^4$, S is a pentadiagonal matrix, entries of which depend on the physical properties and boundary conditions of the beam, and Y_i $(i = k+1, k, k-1)$ is a vector representing the deflection of end of sections 1 to n of the beam at time step i. Equation (6) is the required relation for the simulation algorithm that can easily be implemented on a digital computer.

A single-input single-output active vibration control system is considered for vibration suppression of the beam. The unwanted (primary) disturbance is detected by a detection sensor, processed by a controller to generate a cancelling (secondary, control) signal so as to achieve cancellation at an observation point along the beam. The objective is to achieve total (optimum) vibration suppression at the observation point. This requires the primary and secondary signals at the observation point to be equal in amplitudes and to have a 180° phase difference. Synthesising the controller on the basis of this objective will yield the required controller transfer function as (Tokhi and Hossain, 1994)

$$C = [1 - Q_1/Q_0]^{-1} \tag{7}$$

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 (7) is the required controller design rule which can easily be implemented on-line. This will involve estimating Q_0 and Q_1, using a suitable system-identification algorithm, designing the controller using equation (7) and implementing the controller to generate the cancelling signal. An RLS parameter-estimation algorithm is used to estimate Q_0 and Q_1 in the discrete-time domain in parametric form. Thus, the beam simulation algorithm comprises equation (6), the identification algorithm is referred to as estimation of Q_0 and Q_1 with calculation of the controller parameters according to equation (7) and implementation of the controller constitutes the beam control algorithm. Note that the simulation algorithm is an integral part of the control algorithm.

4. PERFORMANCE METRICS

A commonly used measure of performance of a processor in an application is speedup. This is defined as the ratio of execution time of the processor in implementing the application algorithm relative to a reference time or execution time of a reference processor (Tokhi and Hossain, 1995).

For PP, 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 speed that can be achieved with a parallel architecture of N identical processors working concurrently on a problem is N. This is known as the "ideal speedup". In practice, the speedup is much less, since some processors are idle at times due to conflicts over memory access, communication delays, algorithm inefficiency and mapping for exploiting the natural concurrency in a computing problem (Hwang and Briggs, 1985). But, in some cases, the speedup can be obtained above the ideal speedup, due to anomalies in programming, compilation 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.

When speed is the goal, the power to solve problems of some magnitude in a reasonably short period of time is sought. Speed is a quantity that ideally would increase linearly with system size. Based on this reasoning, The isospeed approach, described by the average unit speed as the achieved speed of a given computing system divided by the number of processors N, has previously been proposed (Sun and Rover, 1994). This provides a quantitative measure of describing the behaviour of a parallel algorithm-machine combination as sizes are varied.

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 different number of processors. It has also been illustrated that the value of efficiency is directly related to the granularity of the system. Although, speedup and efficiency and their variants have widely been discussed in relation to homogeneous parallel architectures. Not much has been reported on such performance measures for heterogeneous parallel architectures.

4.1 Sequential processing

It has previously been reported that the performance of a processor, as execution time, in implementing an application algorithm generally evolves linearly with the task size (Tokhi *et al.*, 1996). This means that a quantitative measure of performance of a processor in an application can adequately be given by the average ratio of task size to execution time or the average speed. Alternatively, the performance of the processor can be measured as the average ratio of execution time per unit task size, or the average (execution time) gradient. In

this manner, a generalised performance measure of a processor relative to another processor in an application can be obtained.

Let the average speeds with two processors p_1 and p_2 in an application, over a range of task sizes, be denoted by V_1 and V_2 respectively. The generalised sequential (execution time) speedup $S_{1/2}$ of p_1 relative to p_2 in implementing the application algorithm can thus be defined as

$$S_{1/2} = V_1/V_2$$

Alternatively, if the corresponding average gradients with p_1 and p_2 for the application, over a range of task sizes, are given by G_1 and G_2 respectively, $S_{1/2}$ can be expressed as

$$S_{1/2} = G_2/G_1$$

The concept of generalised sequential speedup described above can also be utilised to obtain a comparative performance evaluation of a processor for an application under various processing conditions.

4.2 Parallel processing

Due to substantial variation in computing capabilities of the PEs, the traditional parallel performance metrics of homogeneous architectures are not suitable for heterogeneous architectures in their current form. Note, for example, that speed and efficiency provide measures of performance of parallel computation relative sequential computation on a single processor node. In this manner, the processing node is used as reference node. In a heterogeneous architecture, such a reference node, representing the characteristics of all the PEs, is not readily apparent. In this investigation such a reference node is identified by proposing the concept of virtual processor. Moreover, it is argued that a homogeneous architecture can be considered as a sub-class of heterogeneous architectures. In this manner, the performance metrics developed for heterogeneous architectures should be general enough to be applicable to both classes of architectures.

Consider a heterogeneous parallel architecture of N processors. To allow define speedup and efficiency of the architecture, assume a virtual processor is constructed that would achieve a performance in terms of average speed equivalent to the average performance of the N processors. Let the performance characteristics of processor i ($i = 1,\ldots,N$) over task increments of ΔW be given by

$$\Delta W = V_i \Delta T_i \tag{8}$$

where ΔT_i and V_i represent the corresponding execution time increment and average speed of the processor. Thus, the speed V_v and average execution time increment ΔT_v of the virtual processor executing the task increment ΔW are given as

$$V_v = \frac{\Delta W}{\Delta T_v} = \frac{1}{N}\sum_{i=1}^{N} V_i = \frac{1}{N}\sum_{i=1}^{N} \frac{\Delta W}{\Delta T_i} = \frac{\Delta W}{N}\sum_{i=1}^{N} \frac{1}{\Delta T_i}, \quad \Delta T_v = N\left(\sum_{i=1}^{N} \frac{1}{\Delta T_i}\right)^{-1} \tag{9}$$

Thus, the fixed-load increment parallel speedup S_f and generalised parallel speedup S_g of the parallel architecture, over a task increment of ΔW, can be defined as

$$S_f = \Delta T_v / \Delta T_p \quad , \quad S_g = V_p / V_v$$

where ΔT_p and V_p are the execution time increment and average speed of the parallel system. In this manner, the (fixed-load) efficiency E_f and generalised efficiency of the parallel architecture can be defined as

$$E_f = \left(S_f / N\right) \times 100\% \quad , \quad E_g = \left(S_g / N\right) \times 100\%$$

Note in the above that the concepts of parallel speedup and efficiency defined for heterogeneous architectures are consistent with the corresponding definitions for homogeneous architectures. Thus, these can be referred to as the general definitions of speedup and efficiency of parallel architectures.

5. TASK TO PROCESSOR ALLOCATION IN PARALLEL ARCHITECTURES

The concept of generalised sequential speedup can be utilised as a guide to allocation of tasks to processors in parallel architectures so as to achieve maximum efficiency and maximum (parallel) speedup. Let the generalised sequential speedup of processor i (in a parallel architecture) to the virtual processor be $S_{i/v}$;

$$S_{i/v} = V_i / V_v \quad ; \quad i = 1, \ldots, N \tag{10}$$

Using the processor characterisations of equations (8) and (9) for processor i and the virtual processor, equation (10) can alternatively be expressed in terms of fixed-load execution time increments as

$$S_{i/v} = \Delta T_v / \Delta T_i \quad ; \quad i = 1, \ldots, N$$

Thus, to allow 100% utilisation of the processors in the architecture the task increments ΔW_i allocated to processors should be so that the execution time increment of the parallel architecture in implementing the task increment ΔW is given by

$$\Delta T_p = \Delta T_i = \frac{\Delta W_i}{V_i} = \frac{\Delta T_v}{N} = \frac{1}{N} \frac{\Delta W}{V_v} \quad ; \quad i = 1, \ldots, N \tag{11}$$

or, using equation (10),

$$\Delta W_i = \frac{V_i}{V_v} \frac{\Delta W}{N} = S_{i/v} \frac{\Delta W}{N} \quad ; \quad i = 1, \ldots, N \tag{12}$$

It follows from equation (11) that, with the distribution of load among the processors according to equation (12) the parallel architecture is characterised by

$$\Delta W = (NV_v)\Delta T_p = V_p \Delta T_p$$

having an average speed of

$$V_p = NV_v$$

Thus, with the distribution of load among the processors according to equation (12), the speedup and efficiency achieved with N processors are N and 100% respectively. These are the ideal speedup and efficiency. In practice, however, due to communication overheads and run-time conditions the speedup and efficiency of the parallel architecture will be less than these values.

Note in the above that, in developing the performance metrics for a heterogeneous parallel architecture of N processors, the architecture is conceptually transformed into an equivalent homogeneous architecture incorporating N identical virtual processors. This is achieved by the task allocation among the processors according to their computing capabilities to achieve maximum efficiency. For a homogeneous parallel architecture the virtual processor is equivalent to a single PE in the architecture.

6. IMPLEMENTATIONS AND RESULTS

In this section, in addition to evaluation of performance of the architectures in implementing the algorithms, hardware and software related influential factors such as inter-processor communication, compiler efficiency and code optimisation are also looked at.

6.1 Inter-processor communication and overhead

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. These are (i) T8-T8: serial communication, (ii) C40-C40: parallel communication, (iii) T8-C40: serial to parallel communication and (iv) T8-i860: shared memory communication. 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. The inter-processor communication times (in sec) achieved were 0.0018 with C40-C40 (double-line), 0.018 with T8-T8 (double-line), 0.0268 with T8-i860 (double-line), 0.0316 with T8-C40 (double-line), 0.1691 with C40-C40 (single-line) and 0.208 with T8-C40 (single-line). Thus, among these the C40-C40 double-line parallel communication was the fastest, whereas the T8-C40 single-line serial-to-parallel communication was the slowest. It is noticed that as compared to serial communication, parallel communication offers a substantial advantage. In shared-memory communication, additional time is required in accessing and/or writing into the shared memory. In serial-to-parallel communication, on the other hand, an additional penalty is paid during the transformation of data from serial to parallel and vice versa.

To investigate performance with communication overhead, the beam simulation algorithm was chosen. An aluminium type cantilever beam of length $L = 0.635$ m, mass $m = 0.037$ kg and $\mu = 1.351$ was considered. The beam was discretised into 19 segments and a sample period of $\Delta t = 0.3$ msec was used. In this investigation, the total execution times achieved by the architectures, in implementing the algorithm over 20000 iterations was

considered. The algorithm, thus, consists of computation of deflection of nineteen equal-length segments. The computation for each segment requires information from two previous and two forward segments. The algorithm was implemented on networks of up to nine T8s. Figure 1 shows the real-time performance (i.e. computation with communication overhead) and the actual computation time with one to nine PEs. The difference between the real-time performance and the actual computation time is the communication overhead. It is noted that, due to communication overheads, the computing performance does not increase linearly with increasing number of PEs. The performance remains nearly at a similar level with a network having more than six PEs. Note that the increase in communication overhead at the beginning, with less than 3 PEs, is more pronounced and remains nearly at the same level with more than five PEs. This is due to the communication overheads among the PEs which occur in parallel. Such a trend will also be observed in the corresponding speedup and efficiency for the network.

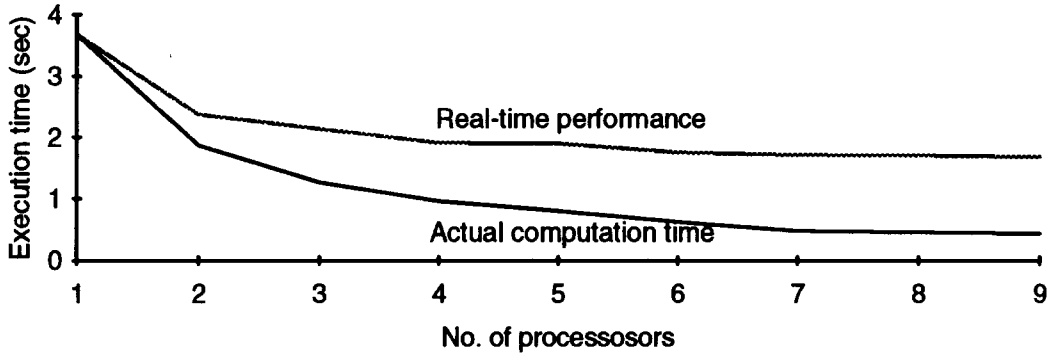


Figure 1: Execution time of the simulation algorithm on the transputer network.

6.2 Efficiency of compilers

To evaluate the efficiency of compilers the 3L Parallel C version 2.1, INMOS ANSI C and Occam were used. The flexible beam simulation algorithm was coded into the three programming languages and run on a T8. The execution times (in sec) achieved were 3.4763, 3.6801 and 5.36 with Parallel C, ANSI C and Occam respectively. It is noted that the performances with Parallel C and ANSI C are nearly at a similar level and at about 1.5 times faster than the Occam. To further investigate this, the simulation algorithm was implemented on networks of one to nine T8s using ANSI C and Occam. The execution times corresponding to this investigation are shown in Figure 2. This further demonstrates that Occam produces slower executable code, for this particular numerical computation, in a PP environment, as compared to ANSI C. This is due to the involvement of integer/floating point operations and run-time memory management of the transputer for which the C compiler is more efficient than Occam. Further investigations confirmed that better performance is achieved, throughout, with the ANSI C compiler than the Occam, except, in a situation, where, the computation involved is floating type data processing with declaring array.

6.3 Code optimisation

Almost always code optimisation facilities of compilers enhance the real-time performance of a processor. The i860 and the C40 have many optimisation features. The TMS320 floating-point DSP optimising C compiler is the TMS320 version of the 3L Parallel C compiler (Texas Instruments, 1991b). It has many options, constituting three levels of optimisation, which aid the successful optimisation of C source code files on the C40. The Portland Group (PG) C compiler is an optimising compiler for the i860 (Portland Group Inc., 1991). It incorporates four levels of optimisation.

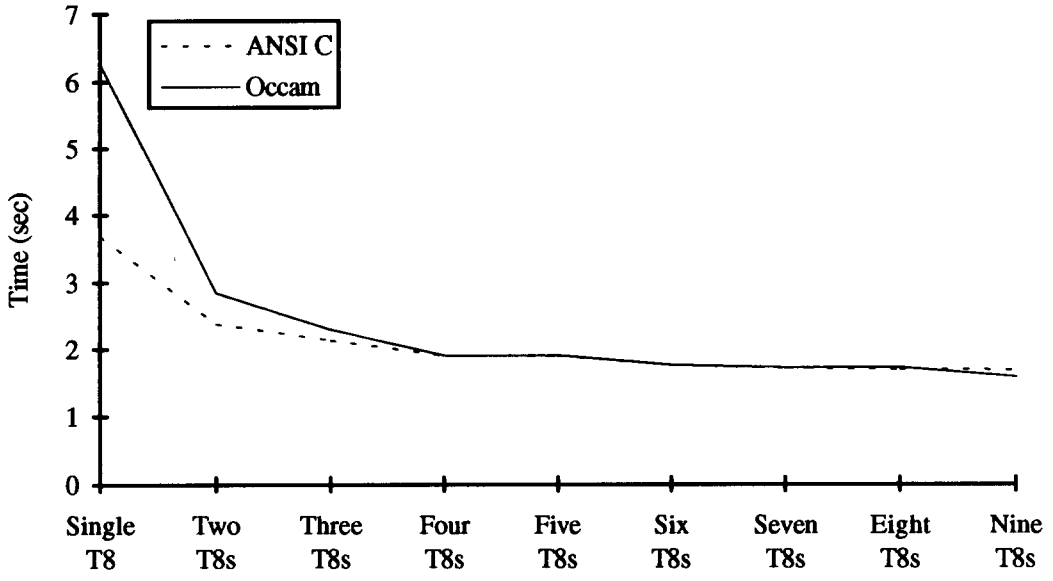


Figure 2: Performance of ANSI C and Occam in implementing the simulation algorithm.

To measure the performance attainable from the compiler optimisers, so as to fully utilise the available features, experiments were conducted to compile and run the LMS and the beam simulation algorithms on the i860 and the C40. To study the effect of the PG optimising compiler, the LMS algorithm was compiled with the number of weights set at 5 and $\eta = 0.04$. The algorithm was implemented on the i860 with four levels of optimisation and the execution time of the processor in implementing the algorithm over 1000 iterations was recorded. Similarly, the beam simulation algorithm was compiled and implemented on the i860 with 5 beam segments and $\Delta t = 0.055$ ms. The execution time of the processor in implementing the algorithm over 20000 iterations was recorded with each of the four levels of optimisation. Table 1 shows the execution times and the corresponding speedups achieved in implementing the algorithms, where level 0 corresponds to no optimisation. It is noted that the performance of the processor in implementing the LMS algorithm has enhanced significantly with higher levels of optimisation. The enhancement in case of the beam simulation algorithm, on the other hand, is not significant beyond the first level. The disparity in the speedups in case of the two algorithms is thought to be due to the type of operations performed by the optimiser.

To study the effect of the optimisers further on the performance of the system, optimisation level 0 (no optimisation) and level 2 were used with the 3L optimiser in

implementing the algorithms on the C40. Similarly, with the i860, using the PG compiler, optimisation level 0 and level 4 were utilised. The LMS and the beam simulation algorithms were coded for various task sizes, by changing the number of weights in case of the LMS algorithm and number of segments in case of the beam simulation algorithm. The algorithms were thus implemented on the i860 and the C40. The enhancement in performance of the processors in implementing the LMS algorithm with optimisation was noted to be substantially greater than in implementing the beam simulation algorithm. This, as discussed above, is due to the structure of the algorithms where for the LMS algorithm the features of the optimisation are well exploited and not much for the beam simulation algorithm. It was further noticed with the corresponding speedups achieved with optimisation in implementing the algorithms on the i860 and the C40, that the enhancement occurs within a specific bandwidth of the task size (number of filter weights). However, due to its better data handling capability, the upper roll-off point for the i860 occurs at a larger task size in comparison to that with the C40 implementation.

Table 1: Execution times and speedup of the i860 with various optimisation levels.

Optimisation level	LMS algorithm		Simulation algorithm	
	Execution time (sec)	Speedup	Execution time (sec)	Speedup
Level 0	0.43	1	0.113035	1
Level 1	0.242	1.7769	0.077534	1.4579
Level 2	0.127	3.3858	0.073536	1.5371
Level 3	0.087	4.9425	0.073535	1.5372
Level 4	0.082	5.2439	0.073533	1.5372

6.4 Algorithms and architectures

To provide a comparative performance evaluation of the architectures, several algorithms, namely a 512-point FFT, cross-correlation with two waveforms each of 1000 samples, the RLS filter with a second-order IIR structure over 1000 iterations, the LMS filter with an FIR structure and $\mu = 0.04$ over 1000 iterations, the beam simulation and control each over 20000 iterations and the beam identification with second-order models over 1000 iterations were considered. Table 2 shows the execution times thus achieved with the architectures in implementing the algorithms. The algorithms have been listed according to their degree of regularity; the FFT at the top is of a highly regular nature, whereas the LMS filter algorithm at the bottom is of a highly irregular nature. It is noted that, among the uni-processor architectures, the i860 performs as the fastest in implementing regular and matrix-based algorithms. In contrast, the C40 performs the fastest of the uni-processors in implementing algorithms of irregular nature. In a similar manner, the performance of the T8 is enhanced by an increasing degree of irregularity in an algorithm.

It is noted in Table 2 that among the parallel architectures the suitability of the i860 for regular and matrix-based algorithms is well reflected in the shortest execution times achieved with the i860+T8 in implementing the FFT, beam simulation, beam control and correlation algorithms. In contrast, the suitability of the C40 is reflected in achieving the shortest

execution times with C40+C40 in implementing the beam identification, RLS filter and LMS filter algorithms. However, due to an unbalanced load distribution among the processors, especially in case of the i860+T8 and C40+T8, the enhancement in the performance of the architecture in relation to the corresponding uni-processors is not significant.

For the performance with heterogeneous architectures to be significant, the concept of virtual processor was utilised with the corresponding task allocation strategy. In this process the beam simulation and the DOT algorithms were used with the C40+T8 architecture. The former was implemented over 20000 iterations for each task size and for the latter was realised with $b = 4.0$, $c = 6.5$ and a as an integer. Figure 3 shows the execution times of the architectures in implementing the algorithms. In these diagrams the characteristics of the virtual processor are also shown. The actual execution times achieved with the C40+T8 in implementing the DOT algorithm are similar to those of the corresponding theoretical (100% efficient) model. In case of the beam simulation algorithm, however, the actual execution time of the architecture is more than that of the corresponding theoretical model. This is due to communication overheads in implementing this algorithm. However, these results demonstrate that the utilisation of the concept of virtual processor in allocating tasks among the processors in a heterogeneous architectures results in maximum efficiency and hence enhances the real-time performance of the architecture substantially.

Table 2: Execution times (sec) of the architectures in implementing the algorithms.

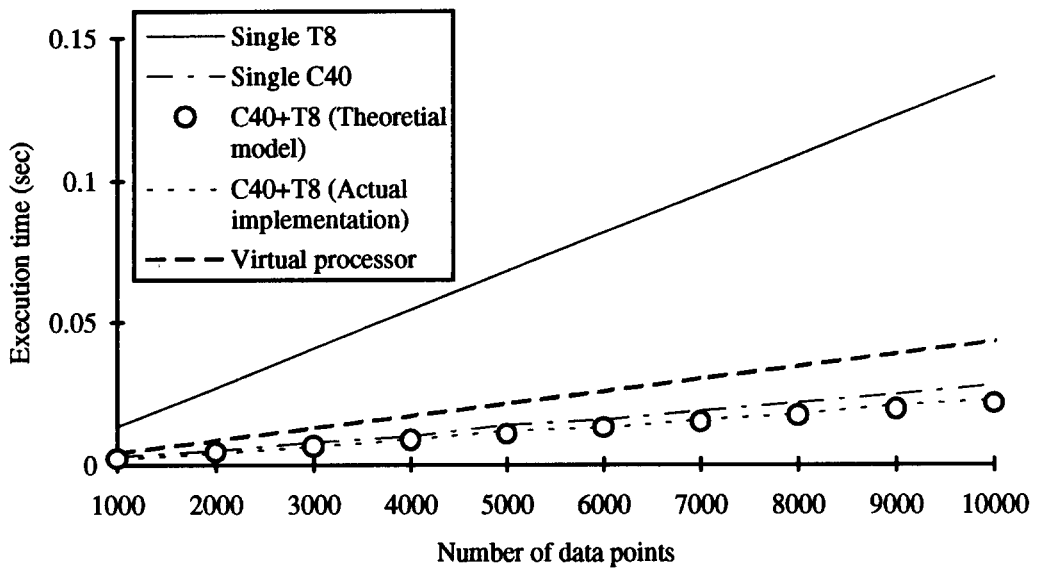
Algorithm	i860	C40	T8	i860+T8	C40+C40	C40+T8	T8+T8
FFT	0.05	1.617	3.9	0.07	1.305	2.1383	3.748
Simulation	0.38	2.3	3.68	0.99	1.618	1.7669	3.008
Control	0.41	2.68	3.695	1.04	1.92	1.963	3.009
Correlation	0.66	1.839	3.824	0.43	0.769	1.332	3.0605
Identification	0.35	0.179	0.674	0.6118	0.2644	0.6114	0.6066
RLS filter	0.18	0.1361	0.349	0.16	0.112	0.1953	0.3167
LMS filter	0.1	0.0353	0.1419	0.12	0.0163	0.0456	0.085

7. CONCLUSION

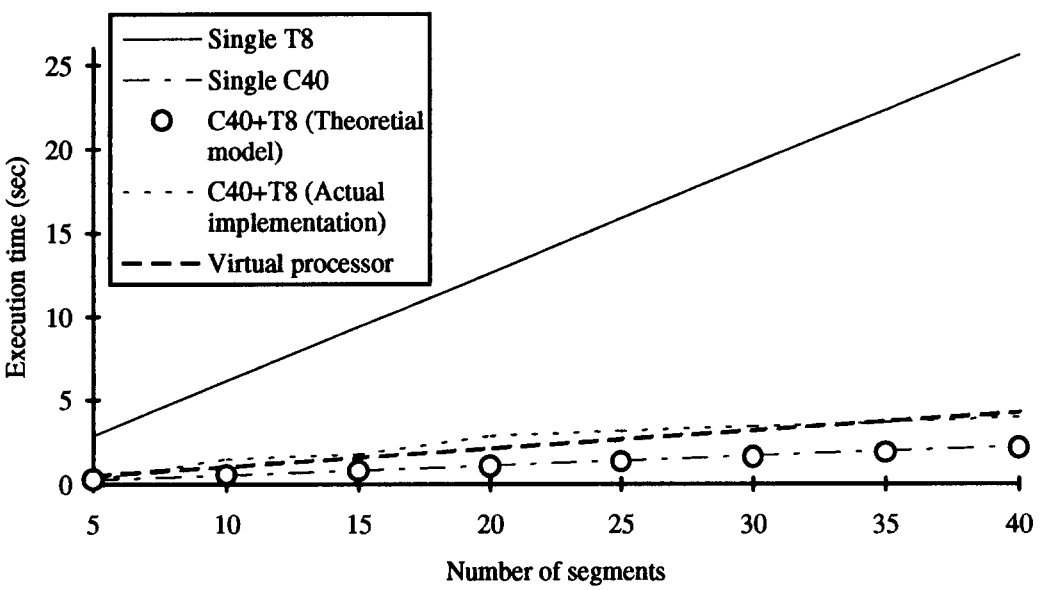
An investigation into the development of high-performance computing methods within the framework of real-time applications has been presented. Influential factors such as inter-processor communication, compiler efficiency and code optimisation affecting the performance of a computing plat form have been investigated. A comparative performance evaluation of several unit-processor and multi-processor architectures with contrasting features in implementing signal processing and control algorithms has been carried out. It has been demonstrated that due to the heterogeneous nature of an application algorithm a close match needs to be made between the computing requirements of the algorithm and the computing capabilities of the architecture.

A quantitative measure of performance for sequential and parallel computing has been introduced and utilised in a comparative performance evaluation of processors in the implementation of real-time application algorithms. It has been shown that the performance of a processor evolves approximately linearly with task size. The has led to the introduction

of several performance metrics, namely the average speed, average gradient, generalised speedup and efficiency, for a more comprehensive performance evaluation of an architecture in sequential and parallel signal processing and control applications. These have been shown to provide suitable measures of the performance of a processor over a wide range of loading conditions and thus reflect on the real-time computing capabilities of the architectures in a comprehensive manner. The performance metrics developed and verified, for parallel computing, apply to both homogeneous and heterogeneous architectures and are consistent with those of traditional architectures. In this manner, these enable suitable task to processor mapping in a parallel architecture so as to achieve for maximum speedup and efficiency.



(a) The DOT algorithm.



(b) The simulation algorithm.

Figure 3: Execution times of the architectures in implementing the algorithms.

8. REFERENCES

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

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

Hwang, K. and Briggs, F. A. (1985). *Computer architecture and parallel processing*, McGraw-Hill, California.

Ifeachor, E. C. and Jervis, B. W. (1993). *Digital signal processing - A practical approach*, Addison - Wesley publishing company, UK.

Irwin, G. W. and Fleming, P. J. (1992). *Transputers in real-time control*, John Wiley, England.

Portland Group Inc. (1991). *PG tools user manual*, Portland Group Inc.

Sun, X.-H. and Gustafson, J. (1991). Toward a better parallel performance metric. *Parallel Computing*, **17**, (10), pp. 1093-1109.

Sun, X.-H. and Ni, L. (1993). Scalable problems and memory-bounded speedup, *Journal of Parallel and Distributed Computing*, **19**, (September), pp. 27-37.

Sun, X.-H. and Rover, D. T. (1994). Scalability of parallel algorithm-machine combinations. *IEEE Transactions on Parallel and Distributed Systems*, **5**, (6), pp. 599-613.

Texas Instruments. (1991a). *TMS320C40 User's Guide*, Texas Instruments, USA.

Texas Instruments. (1991b). *TMS320 floating-point DSP optimising C compiler user's Guide*, Texas Instruments, USA.

Tokhi, M. O., Chambers, C. and Hossain, M. A. (1996). Performance evolution with DSP and transputer based systems in real-time signal processing and control applications, *Proceedings of UKACC International Conference on Control-96*, Exeter, 02-05 September 1996, **1**, pp. 371-375.

Tokhi, M. O. and Hossain, M. A. (1994). Self-tuning active vibration control in flexible beam structures, *Proceedings of IMechE-I: Journal of Systems and Control Engineering*, **208** (I 4), pp. 263-277.

Tokhi, M. O. and Hossain, M. A. (1995). CISC, RISC and DSP processors in real-time signal processing and control, *Microprocessors and Microsystems*, **19**, (5) pp 291-300.

Tokhi, M. O. and Hossain, M. A. (1996). Real-time active control using sequential and parallel processing methods, In Crocker, M J and Ivanov, N I (eds.) *Proceedings of the fourth International Congress on Sound and Vibration*, St Petersburg, 24-27 June 1996, **1**, pp 391-398.

Tokhi, M. O., Hossain, M. A., Baxter, M. J. and Fleming, P. J. (1995). Real-time performance evaluation issues for transputer networks. In Nixon, P. (ed.), *Transputer and occam Developments, WoTUG-18: Proceedings of 18th International Conference of the World Occam and Transputer User Group*, Manchester, 07-09 April 1995, IOS Press, Amsterdam, pp. 139-150.

Tokhi, M. O. and Leitch, R. R. (1992). *Active noise control*, Oxford Science Publications, Clarendon Press, Oxford.

Transtech Parallel Systems Ltd. (1991). *Transtech parallel technology*, Transtech Parallel Systems Ltd, UK.

Widrow, B., Glover, J. R., McCool, J. M., Kaunitz, J., Williams, C. S., Hearn, R. H., Zeidler, J. R., Dong, E. and Goodlin, R. C. (1975). Adaptive noise cancelling: principles and applications, *Proceedings IEEE*, **63**, pp. 1692-1696.