Department of
AUTOMATIC
CONTROL
and SYSTEMS
ENGINEERING
UNIVERSITY of SHEFFIELD

PERFORMANCE EVALUATION IN SEQUENTIAL REAL-TIME PROCESSING

M O Tokhi ‡, M A Hossain § and C Chambers ¤

‡ Department of Automatic Control and Systems Engineering,
The University of Sheffield, Mappin Street, Sheffield, S1 3JD, UK.

§ Department of Computer Science, University of Dhaka, Dhaka, Bangladesh.

¤ Department of Computer Science, The University of Sheffield, Sheffield, UK.

Tel: + 44 (0)114 282 5136.
Fax: + 44 (0)114 273 1729.
E-mail: O.Tokhi@sheffield.ac.uk.

Research Report No. 645

October 1996

Abstract

This paper presents an investigation into the performance evaluation of high-performance processors in real-time applications. The performance evolution of the processors in relation to task size, compiler efficiency for numerical computation and code optimisation are investigated. A quantitative measure of performance evaluation of processors is introduced. An adaptive filtering algorithm and a beam simulation algorithm are considered. These are implemented on several high-performance processors incorporating transputers and digital signal processing devices. A comparative performance evaluation of the architectures is made, demonstrating the critical issues encountered with fast processing techniques in real-time signal processing and control.

Key words: Digital signal processing, performance evaluation, real-time signal processing and control.

CONTENTS

LIST OF TABLES AND FIGURES

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 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 computing methods are employed, could provide suitable solutions in such applications (Tokhi and Hossain, 1996). Not much work has been reported on such methods in real-time signal processing and control applications (Leitch and Tokhi, 1986; Tokhi, 1992; Tokhi and Hossain, 1995; Tokhi *et al.*, 1995).

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 (Tokhi and Hossain, 1995). 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 microprocessor 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 (Anderson, 1991; Hwang, 1993). The work reported in this paper attempts to investigate such issues within the framework of real-time applications.

An investigation into the computing capabilities of several high-performance processors and their suitability in real-time applications is presented in this paper. Three computing domains, namely the Texas Instruments TMS320C40 (C40) DSP device, an Intel 80i860 (i860) vector processor, and an Inmos T805 (T8) transputer are studied. The algorithms chosen to highlight the main characteristics of these processors are the least mean square (LMS) based adaptive filtering algorithm and finite difference (FD) simulation of a flexible beam in transverse vibration. Previous investigations have established a comparative performance evaluation of these processors in implementing the above algorithms on the basis of fixed task sizes. Not much work, however, is known of the performance evolution of such processors in relation to task size in real-time applications (Tokhi *et al.*, 1996). Moreover, similar studies involving code optimisation and compiler efficiency have not been reported. This work aims at addressing these points.

2 ALGORITHMS

The performance of a processor is largely affected by the computing requirements of an application. This in turn is determined by the degree of regularity of the algorithm. Regularity is used to describe the degree of uniformity in the execution thread of the computation. An algorithm consisting of varying loops and conditional jumps, for instance, is considered to be highly irregular. 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 an RI type algorithm, a vector processor will, principally, be expected to perform better. Moreover, if a large amount of data is to be handled for computation in these type of algorithms, the performance will further be enhanced if the processor has more internal data cache, instruction cache and/or a built-in math co-processor.

The algorithms considered in this investigation consist of the FD simulation of a flexible beam structure and an LMS adaptive filter algorithm. These are briefly described below.

2.1 Beam simulation

Consider a cantilever beam system with a force $U(x,t)$ 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 stationery position at the point where the force has been applied. In this manner, the governing dynamic equation of the beam is given by

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

where, μ is a beam constant and m is the mass of the beam. Discretising the beam in time and length using central FD methods, a discrete approximation to Equation (1) can be obtained as (Tokhi and Hossain, 1994)

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

where, $\lambda^2 = \left[(\Delta t)^2 / (\Delta x)^4\right]\mu^2$ with Δt and Δx representing the step sizes in time and along the beam respectively, S is a pentadiagonal matrix (the so called stiffness matrix of the beam), Y_i $(i = k+1, k, k-1)$ is an $n \times 1$ matrix representing the deflection of end of sections (grid-points) 1 to n of the beam at time step i (beam divided into $n-1$

sections). Equation (2) is the required relation for the simulation algorithm that can be implemented on a digital processor easily.

2.2 The LMS adaptive filter

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

$$\mathbf{W}_{k+1} = \mathbf{W}_k - 2e_k\eta\mathbf{X}_k \tag{3}$$

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

$$e_k = y_k - \mathbf{W}_k^T\mathbf{X}_k \tag{4}$$

where, y_k is the current contaminated signal sample. Equations (3) and (4) constitute the LMS adaptive filter algorithm.

Note in the above that the beam simulation algorithm can be considered as an RI algorithm whereas the LMS adaptive filter constitutes an irregular algorithm.

3 HARDWARE

The architectures considered include an i860 vector processor, a C40 DSP device and a T8 transputer. These are described below.

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. It is the Intel's first superscalar RISC processor possessing separate integer, floating-point, graphics, adder, multiplier and memory management units. The i860 executes 82 instructions, including 42 RISC integer, 24 floating-point, 10 graphics, and 6 assembler pseudo operations in one clock cycle. All external or internal address buses are 32-bit wide and the external data path or internal data bus is 64-bits wide.

However, the internal RISC integer ALU is only 32 bits wide. The instruction cache transfers 64 bits per clock cycle, equivalent to 320 Mbytes/sec at 40 MHz. In contrast, the data cache transfers 128 bits per clock cycle. There are two floating-point units, namely, the multiplier and the adder units, which can be used separately or simultaneously under the co-ordination of the floating point control unit. Special dual-operation floating-point instructions such as add-and-multiply and subtract-and-multiply use both the multiplier and adder units in parallel. Furthermore, both the integer and the floating-point control units can execute concurrently (Hwang, 1993).

The C40 is a high-performance Texas Instruments 32-bit DSP processor with 40 MHz clock speed, 8 Kbytes on-chip RAM, 512 bytes on-chip instructions cache and is capable of 275 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 co-processor buses for support of massive concurrent I/O of program and data throughput, thereby maximising sustained CPU performance (Brown, 1991; Texas Instruments, 1991a).

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

4 SOFTWARE

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 (Hwang, 1993). Minimising the run-time memory management within the program and selecting 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.

Compilers have a significant impact on the performance of the system. This means that some high-level languages have advantages in certain computational domains and some have advantages in other domains. The compiler itself is critical to the performance of the system as the efficiency of the mechanism for taking a high-level description of the application and transforming it into a hardware dependent implementation differs from compiler to compiler. Identifying the foremost compiler for the application in hand is, therefore, especially challenging due to the unpredictable run-time behaviour of the program and memory management capabilities using different compilers. In signal processing and control applications it is important to select a suitable programming language that can support highly numerical computation for real-time implementation. The investigation here involves performance evaluation issues of some commonly used compilers with the computing platforms considered.

The algorithms are coded in high-level languages as appropriate for the hardware used, with the code structure kept as similar as possible for a given application. 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.

Performance is also related to code optimisation facility of the compiler, which may be machine dependent. The goal of program optimisation is, in general, to maximise the speed of code execution. This involves several factors such as minimisation of code length and memory accesses, exploitation of parallelism, elimination of dead code, in-line function expansion, loop unrolling and maximum utilisation of registers. The optimisation techniques include vectorisation using pipelined hardware and parallelisation using multiprocessors simultaneously (Hwang, 1993). The i860 and the C40 processors with their respective compilers are considered in investigating their optimisation facility.

5 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). The speedup thus defined provides a relative performance measure of a processor for fixed load (task size) and thus can be referred to as fixed-load speedup. This can also be used to obtain a comparative performance measure of a processor in an application with fixed task sizes under different processing conditions, for example, with and without code optimisation. However, this performance metric does not provide a measure of performance of the processor over a wide range of computational demands of the application.

It has been observed previously and demonstrated later in this investigation 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). With some processors, however, anomalies in the form of change of gradient (slope) of execution time to task size are observed. These are mainly due to run-time memory management conditions of the processor where up to a certain task size the processor may find the available cache sufficient, but beyond this it may require to access lower level memory. Despite this the variation in the slope is relatively small and the execution time to task size relationship can be considered as linear. 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 in an application can be obtained.

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

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

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

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

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

The concept of speed, assumed to be constant for a processor in an application, has previously been utilised to derive an expression for the generalised speedup as the ratio of parallel speed (of a parallel architecture) over sequential speed (of a single processor) (Sun and Gustafson, 1991; Sun and Rover, 1994). The generalised speedup introduced above, however, reflects on the relative performance of two uni-processor architectures in an application and of the same processor under two different processing conditions.

6 IMPLEMENTATIONS AND RESULTS

In the following investigations involving the beam simulation algorithm an aluminium type cantilever beam of length $l = 0.635$ m , mass $m = 0.037$ kg and $\mu = 1.351$ was considered.

6.1 Compiler efficiency

In this section results of investigations of the performance evaluation of several compilers are presented and discussed. The compilers involved are the 3L Parallel C version 2.1, Inmos ANSI C and Occam. All these compilers can be utilised with the computing platforms considered in this investigation. It has previously been reported that, although Occam is more hardware oriented and a straight-forward programming language for parallel processing, it may not be as suitable as the Parallel C or ANSI C compilers for numerical computations (Bader and Gehrke, 1991). To obtain a comparative performance evaluation of these compilers, the flexible beam simulation algorithm was coded, for 19 equal-length beam sections, into the three programming languages and run on a T8. Figure 1 shows the execution times in implementing the simulation algorithm, over 20000 iterations, using the three compilers. 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. This was further investigated with a linear algebraic equation. Table 1 shows the performances with integer and floating point operations with and without array. It is noted 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. As compared to ANSI C, better performance is achieved with Parallel C for both integer and floating type computation with array. Better performance is achieved with Occam compiler for floating type computation as compared to Parallel C. It is also noted that for the amount of double data handling 1.9 times more execution time was required with Occam. In contrast, 1.87 times more execution time was required with each ANSI C and Parallel C. This implies that for large amounts of data handling, run-time memory management problem can be solved with Parallel C and ANSI C more efficiently than with the Occam compiler.

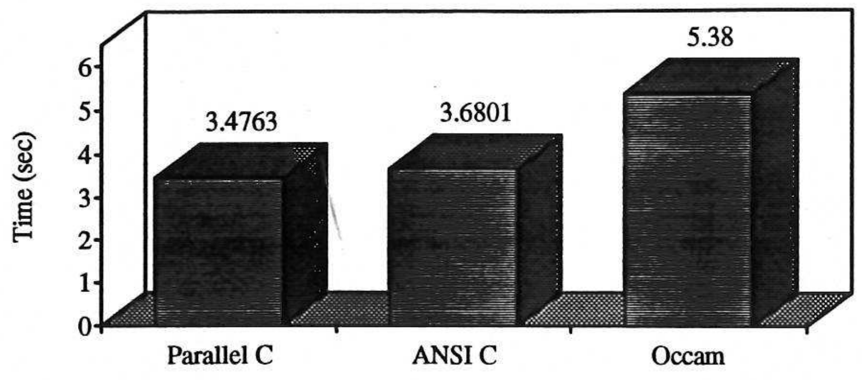


Figure 1: Performance of the compilers in implementing the simulation algorithm on the T8.

Table 1: Comparison of compilers for different type of data processing.

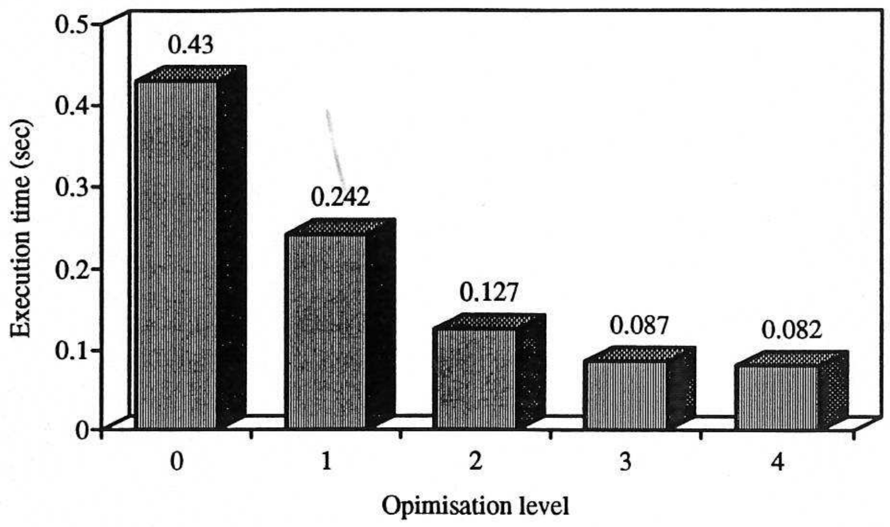
Equation used: $z = (x + i * y - x * i) / (x * x + y * y)$; $i = 0,1,2,\ldots,20000$;
The same equation is repeated twice for 40000 declaring another variable z_1, i.e.
$z_1 = (x + i * y - x * i) / (x * x + y * y)$; $i = 0,1,2,\ldots,20000$
Values used: Integer *x=55, y=25,* Floating: *x=55.02562, y=25.13455.*

Compiler	Floating type data processing		Integer type data processing	
	With array 20000/40000	Without array 20000/40000	With array 20000/40000	Without array 20000/40000
3L Parallel C	0.1327 / 0.2488	0.1263 / 0.2333	0.1327 / 0.2488	0.1263 / 0.2333
ANSI C	0.1328 / 0.2444	0.0227 / 0.0227	0.1328 / 0.2444	0.0226 / 0.0226
Occam	0.1078 / 0.2052	0.1078 / 0.2044	0.1960 / 0.3825	0.1905 / 0.3698

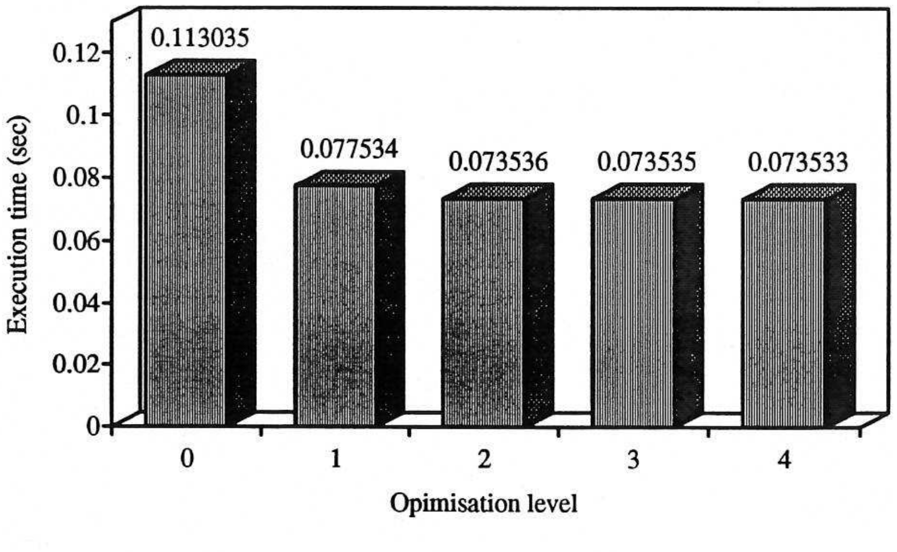
6.2 Code optimisation

Code optimisation facility of compilers for hardware is another important component affecting the real-time performance of a processor. Almost always optimisation facilities enhance the real-time performance of a processor. The i860 and the C40 have many optimisation features (Portland Group Inc., 1991; Texas Instruments, 1991a,b). The TMS320 floating-point DSP optimising C compiler is the TMS320 version of the 3L Parallel C compiler (Texas Instruments, 1991c). 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.

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. Figure 2 shows the execution times achieved in implementing the LMS and the beam simulation algorithms, where level 0 corresponds to no optimisation. The corresponding execution time speedups achieved with each optimisation level in implementing the LMS and beam simulation algorithms are shown in Figure 3. It is noted in Figures 2 and 3 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. As the LMS algorithm has multiple nested loops, the compiler is able to recognise the structures involved and restructure the actual code so that it is still functionally equivalent but suits the CPU architecture better. Branches to sub-routines, for instance, are in-lined which cuts out procedure call overheads, hence speeding up the execution time. The beam simulation algorithm, however, is already in a matrix format and thus does not need as much room for improvement (Portland Group Inc., 1991).

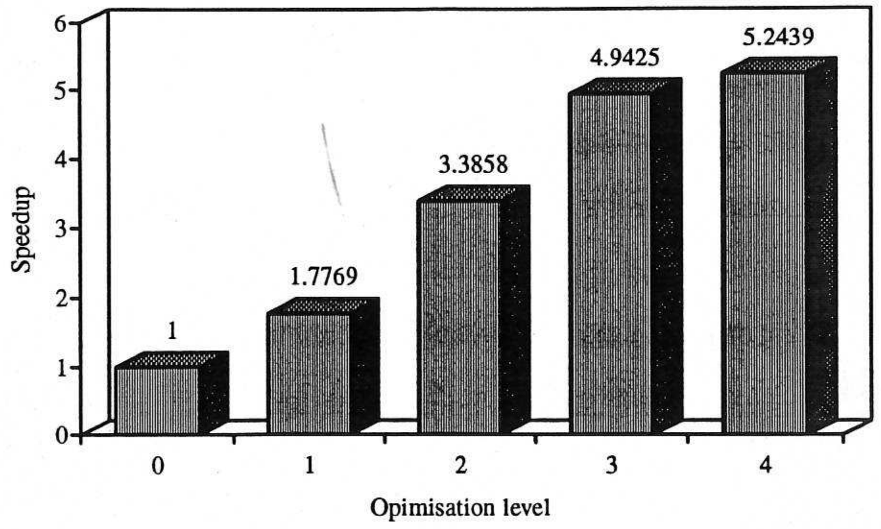


(a)

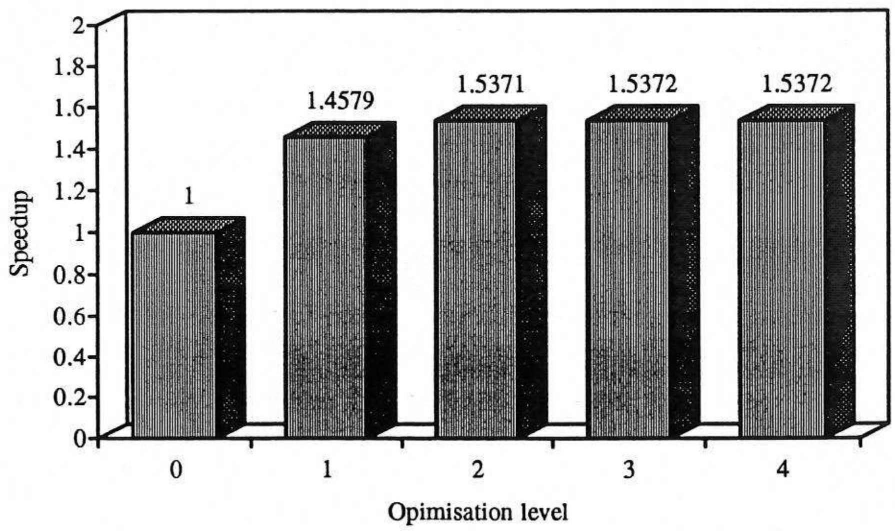


(b)

Figure 2: Execution times of the i860 in implementing the algorithms with the Portland Group C compiler optimiser;
(a) The LMS algorithm.
(b) The beam simulation algorithm.



(a)

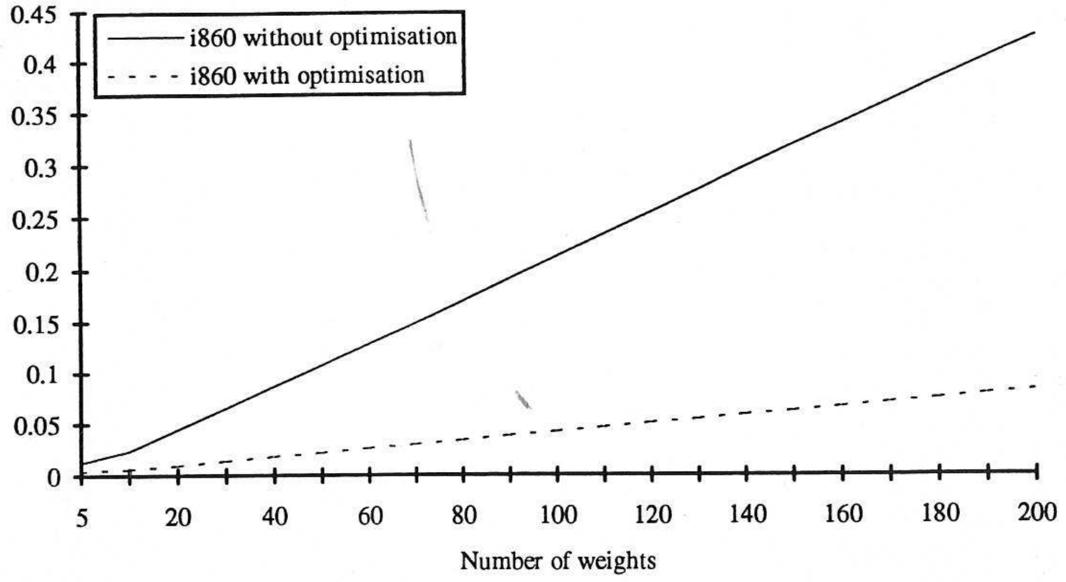


(b)

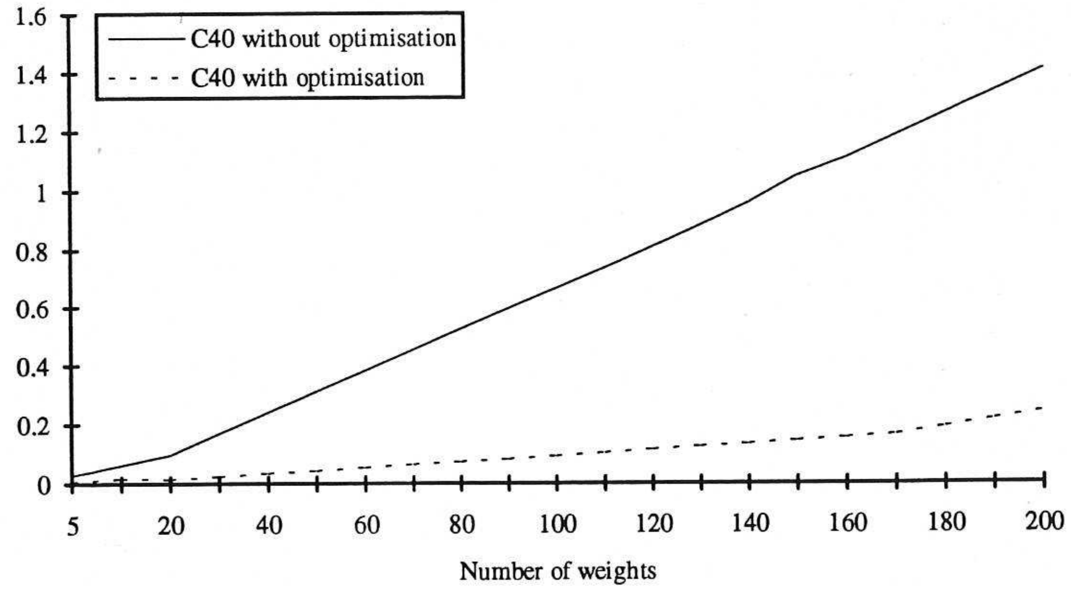
Figure 3: Speedup with the Portland Group C compiler optimiser in implementing the algorithms on the i860;
(a) The LMS algorithm.
(b) The beam simulation algorithm.

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. Figures 4 and 5 show the execution times achieved by the processors in implementing the LMS and beam simulation algorithms over 1000 and 20000 iterations respectively. It is noted that the execution time in each case varies approximately linearly as function of the task size. With the LMS algorithm, as noted, the relation has a relatively smaller gradient for less than 10 and 20 weights with the i860 and the C40 respectively. For the number of weights greater than these, the gradient is larger. Such a phenomenon is more likely associated with dynamic memory management conditions of the processor in each case. In case of the beam simulation algorithm, as noted in Figure 5, such a situation is not clearly evident. This suggests that with the amount of data handling involved in implementing this algorithm both processors appear to have required utilising the lower level memory throughout. The slight variation in gradient noted in Figure 5(b) with the algorithm implemented on the C40 is more likely due to computational error.

It is noted in Figures 4 and 5 that the enhancement in performance of the processors in implementing the LMS algorithm with optimisation is 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. This is further evidenced in Figure 6 with the corresponding speedups achieved with optimisation in implementing the algorithms on the i860 and the C40. It is important to note that the optimisation with the C40 offers significant enhancement in implementing the LMS algorithm. However, the enhancement occurs within a specific bandwidth of the task size (number of filter weights). A similar trend will also be expected with the speedup achieved

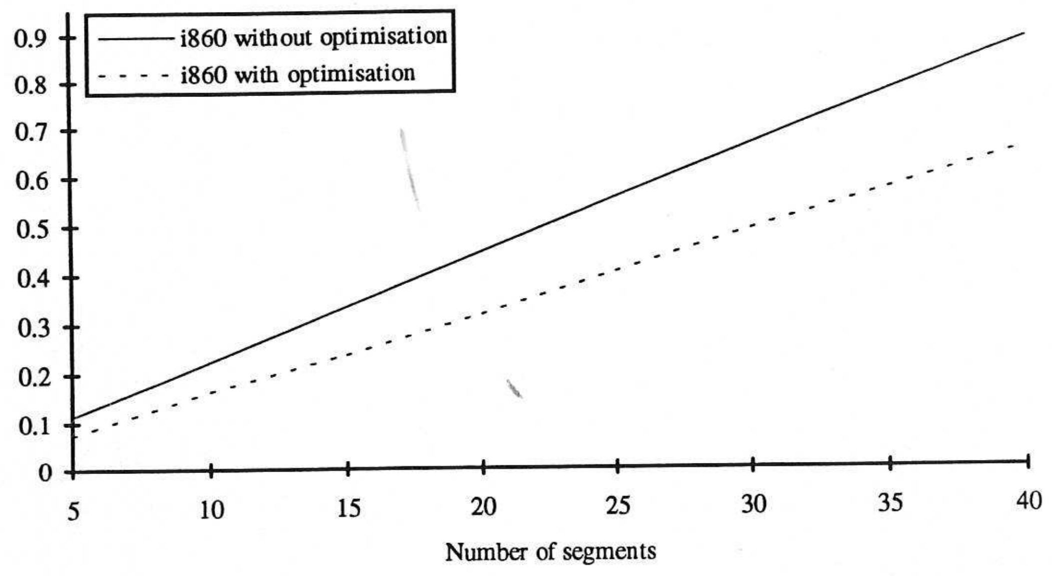


(a)

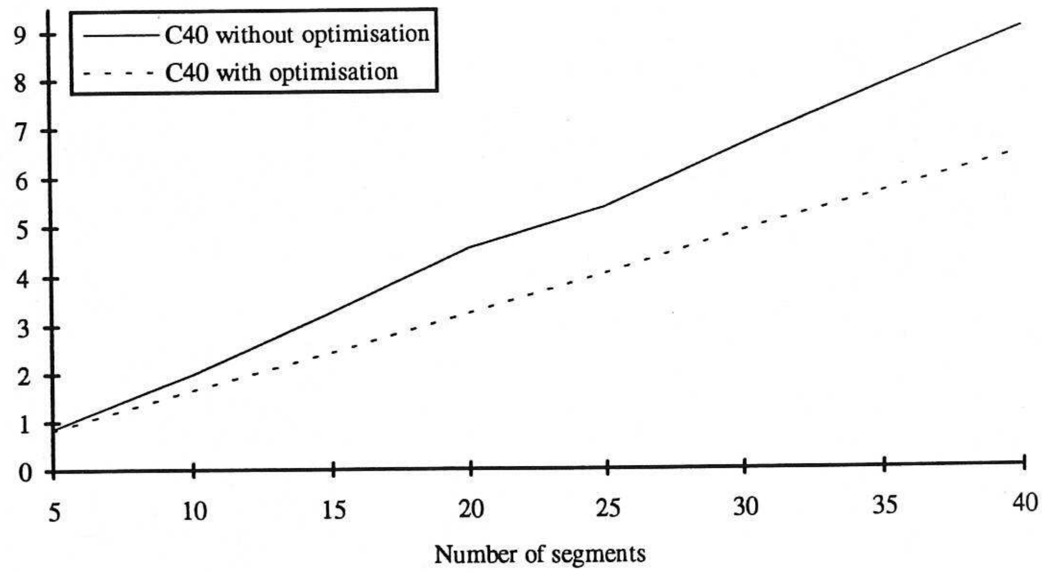


(b)

Figure 4: Execution times of the processors in implementing the LMS algorithm;
(a) With the i860.
(b) With the C40.

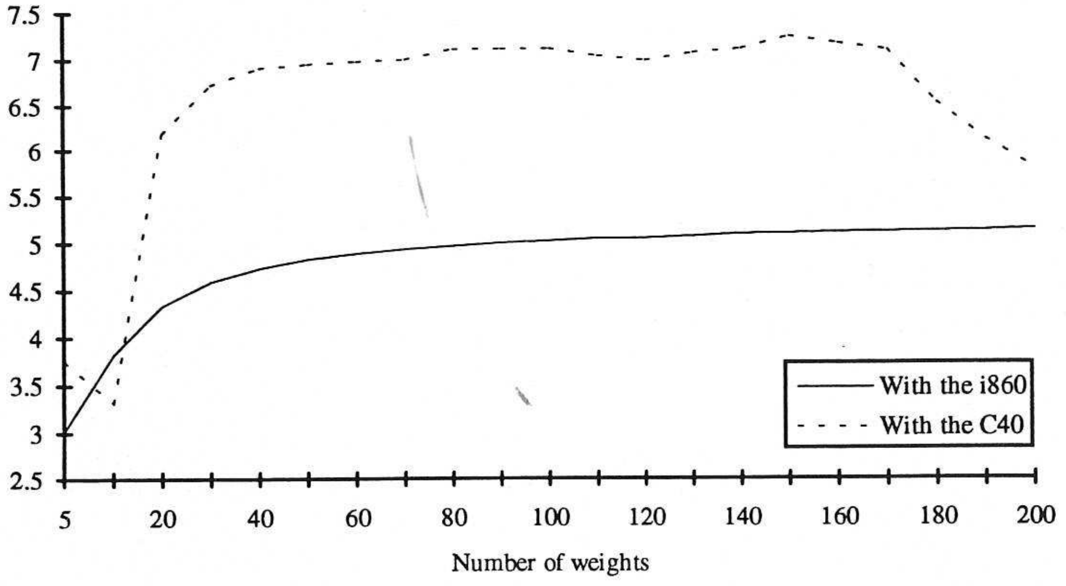


(a)

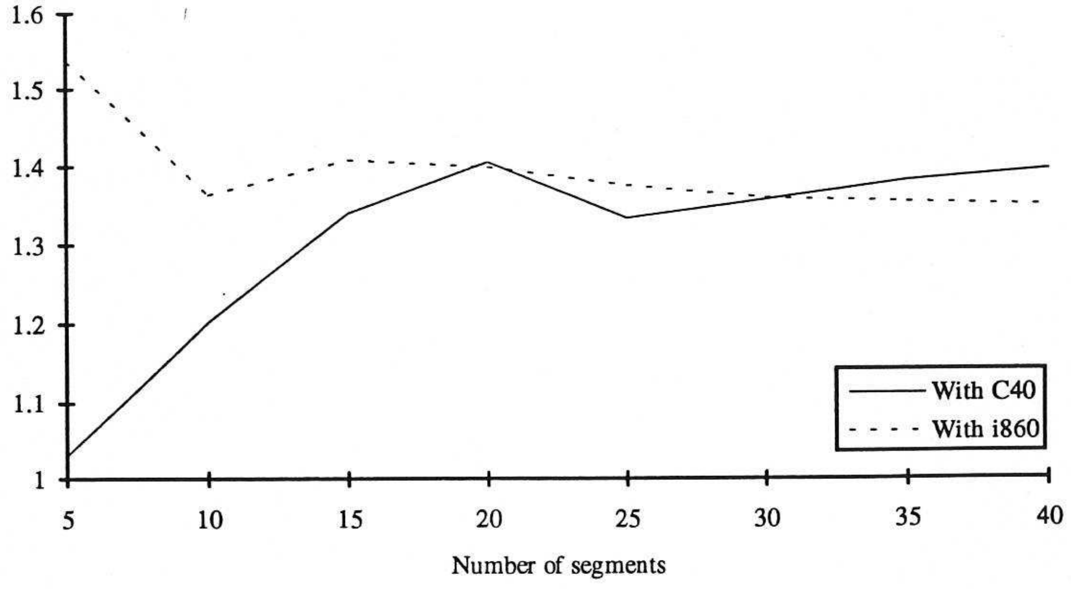


(b)

Figure 5: Execution times of the processors in implementing the beam simulation algorithm;
(a) With the i860.
(b) With the C40.



(a)



(b)

Figure 6: Optimisation speedup with the processors in implementing the algorithms;
(a) The LMS algorithm.
(b) The beam simulation algorithm.

in case of the i860 in implementing the LMS algorithm. However, due the better data handling capability of the i860 the upper roll-off point is expected to occur at a larger task size in comparison to that with the C40 implementation.

The execution time speedups achieved with code optimisation in implementing the beam simulation algorithm on the processors, as noted in Figure 6(b), reach a similar level with both the i860 and the C40. At the lower end of task sizes the speedup with the C40 is relatively larger and continues to decrease with an increase in the task size. The speedup with the implementation on the i860, on the other hand, increases with the task size rapidly at the lower end and slowly beyond 20 beam segments. This suggests that the optimisation for the i860 is performing better than that for the C40 in this type of application.

6.3 Comparative performances of the processors

To evaluate the performance of the processors in implementing the algorithms relative to one another, the algorithms were implemented on the i860, the C40 and the T8 with various task sizes and the execution times recorded accordingly. With the i860 and the C40 implementations the two levels of optimisation considered above were utilised. The execution times recorded in implementing the LMS algorithm in these exercises correspond to 1000 iterations and those for the beam simulation algorithm correspond to 20000 iterations.

Figure 7 shows the execution times of the processors in implementing the LMS algorithm. It is noted that the i860 with optimisation has performed the fastest and the T8 as the slowest of the processors. The performance of the C40, on the other hand, appears to have enhanced significantly with optimisation as compared to its performance without optimisation. This is further demonstrated by the average computational speed of the processors shown in Figure 8 in implementing the LMS algorithm. It is evident from Figure 8 that the speedup achieved with the i860 including optimisation is 5.262, 2.2728, 16.274 and 35.996 relative to the i860 with no optimisation, the C40 with optimisation, the C40 with no optimisation and the T8 respectively. The C40 with optimisation, on the other

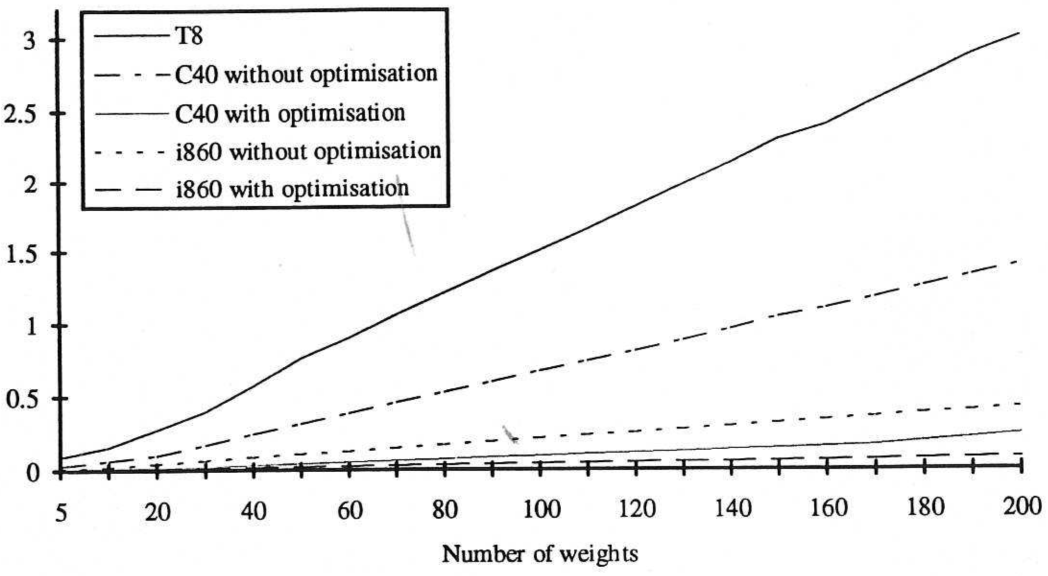


Figure 7: Execution times of the processors in implementing the LMS algorithm.

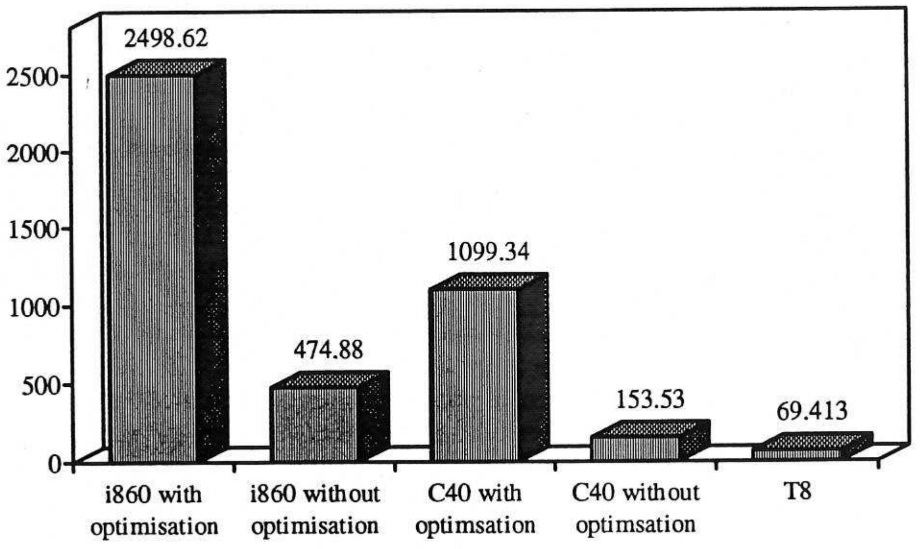
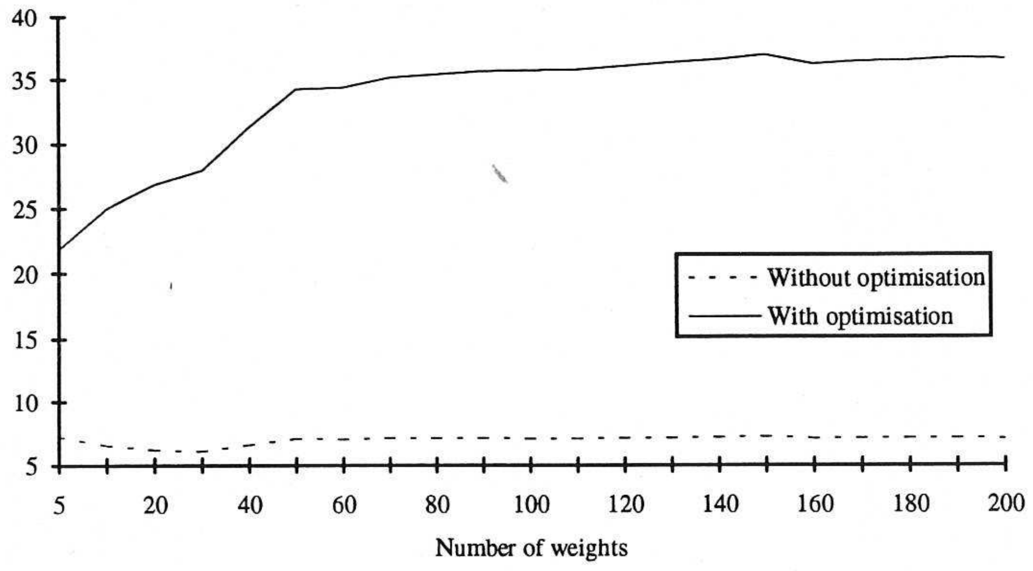


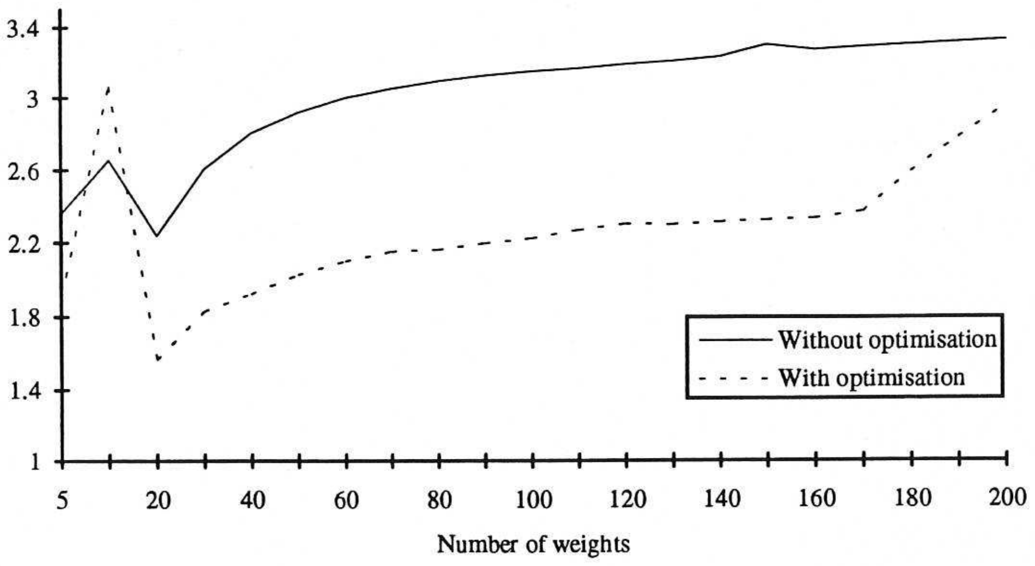
Figure 8: Processor average speed in implementing the LMS algorithm.

hand, appears to have performed 2.315, 7.16 and 15.838 times faster than the i860 without optimisation, the C40 without optimisation and the T8 respectively. Although, the i860 without optimisation appear to have performed slower than the C40 with optimisation, as is further evidenced from the execution time speedup achieved with the i860 relative to the

T8 and the C40 as function of the number of filter weight in Figure 9, the i860 with optimisation has outperformed both the C40 and the T8 significantly in implementing the LMS algorithm.



(a)



(b)

Figure 9: Execution time speedup with the i860 in implementing the LMS algorithm;
(a) Relative to the T8.
(b) Relative to the C40.

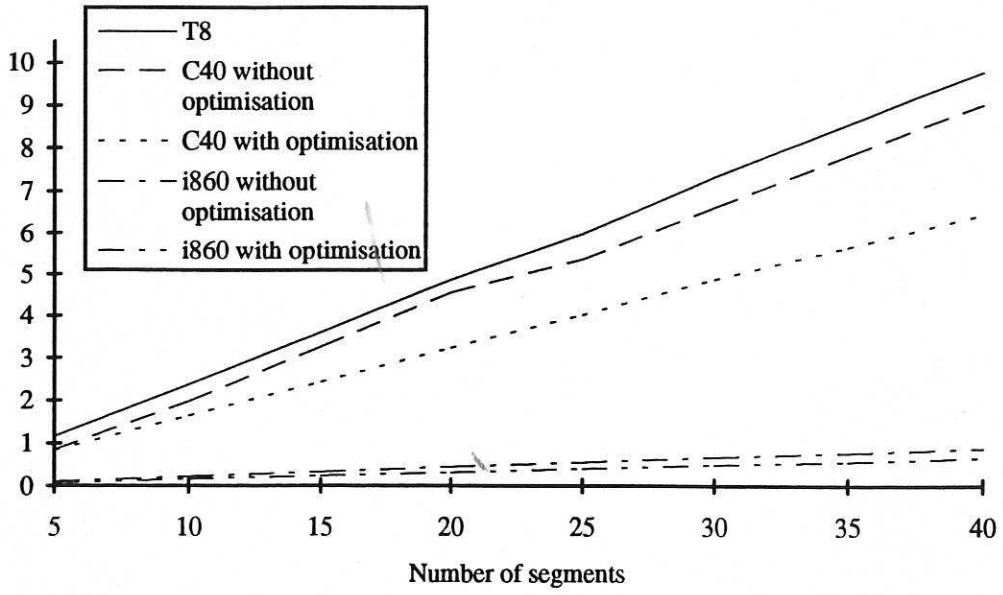


Figure 10: Execution times of the processors in implementing the beam simulation algorithm.

Figure 10 shows the execution times of the processors in implementing the beam simulation algorithm. It is noted that the i860 with optimisation has performed the fastest and the T8 as the slowest of the processors. The performance of both the i860 and the C40 appear not to have enhanced significantly with optimisation as compared to their performances without optimisation. This is due to the nature of the beam simulation algorithm for which the optimisation facility does not offer much in each case. The performance of the C40 without optimisation appear not to be significantly different from that of the T8. The performance of the i860 with and without optimisation is significantly better than the performance of both the C40 and the T8. This is further demonstrated by the average computational speed of the processors shown in Figure 11 in implementing the algorithm. It is evident from Figure 11 that the speedup achieved with the i860 with optimisation is 1.322, 9.693, 14.032 and 14.711 relative to the i860 without optimisation, the C40 with optimisation, the C40 without optimisation and the T8 respectively. The C40 with optimisation, on the other hand, appears to have performed 7.331 times slower than the i860 without optimisation, but only 1.448 and 1.518 times faster than the C40 without optimisation and the T8 respectively. Such a significant performance of the i860 in implementing the beam simulation algorithm is further evidenced in Figure 12 which shows the speedup achieved with the i860 relative to the T8 and the C40 as function of the beam segments.

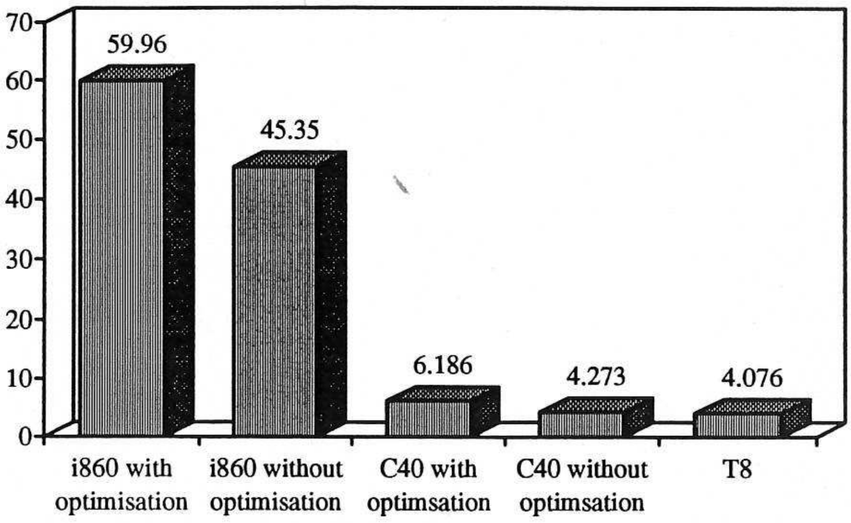


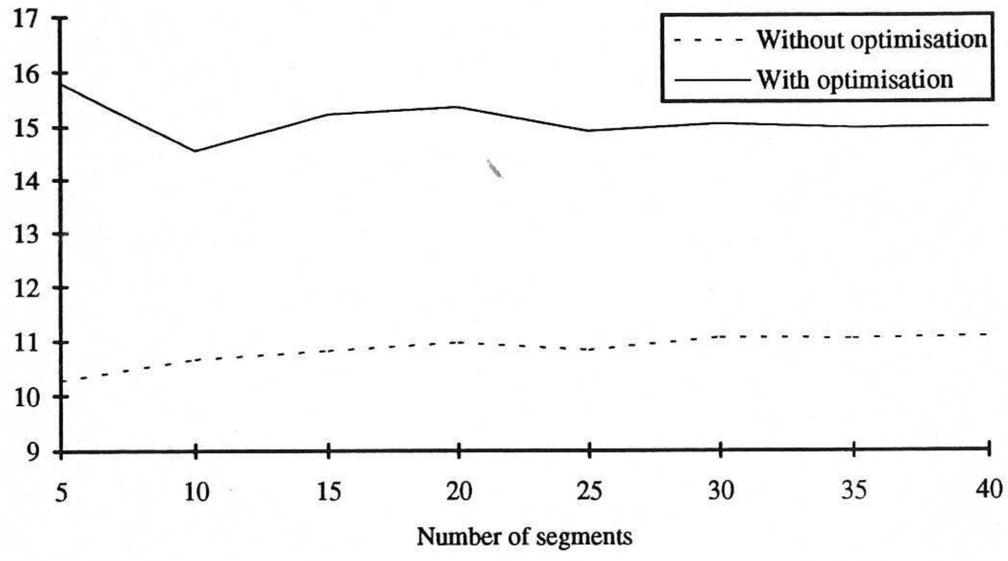
Figure 11: Processor average speed in implementing the beam simulation algorithm.

7 CONCLUSION

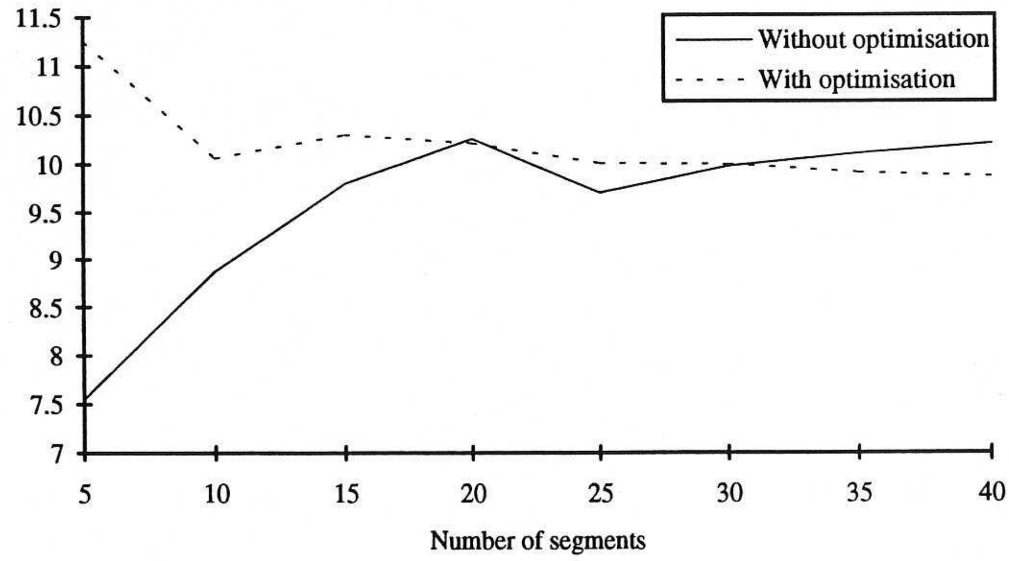
An investigation into the comparative performance evaluation of several high-performance DSP and RISC processors within the framework of real-time applications has be presented. Compiler efficiency and code optimisation of the processors as important issues, affecting the performance of the processors, in real-time applications have been investigated.

Several areas of importance when considering optimising algorithms to suit an architecture or vice versa have been noted. In configuration files, amounts of memory may be allocated for the use of stack, heap or data storage functions. Careful choice of these can determine whether an algorithm can be run in on-chip cache memory or in local memory. This, in turn, can affect the speed of execution. In cases where the algorithm is irregular, involving many branches or subroutines and uneven loops, such as digital filters and similar signal processing algorithms, the C40 has been found to be well suited.

However, with an efficient compiler optimiser the more powerful number crunchers, such as the i860 vector processor, can out perform the C40 at its own game. One reason for this is that the compiler optimiser restructures the code so that it is in a more regular form.



(a)



(b)

Figure 12: Execution time speedup with the i860 in implementing the beam simulation algorithm;
(a) Relative to the T8.
(b) Relative to the C40.

A quantitative measure of performance of processors 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 lead to the introduction of several performance metrics, namely the average speed, average gradient and generalised speedup, for a more comprehensive performance evaluation of a processor under various processing conditions and relative to other processors within an application. 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 processor in a comprehensive manner.

8 REFERENCES

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

Bader, G. and Gehrke, E. (1991). On the performance of transputer networks for solving linear systems of equation. *Parallel Computing*, **17**, pp 1397-1407.

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

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

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

Leitch, R. R. and Tokhi, M. O. (1986). The implementation of active noise control systems using digital signal processing techniques. *Proceedings of the Institute of Acoustics*, **8**, (Part 1), pp 149-157.

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 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). *TMS320C4x User's Guide*, Texas Instruments, USA.

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

Tokhi, M. O. (1992). Implementation of adaptive active noise control systems using digital signal processing devices. *Proceedings of the Institute of Acoustics*, **14**, (Part 4), pp 543-550.

Tokhi, M. O., Chambers, C. and Hossain, M. A. (1996). Performance evolution with DSP and transputer based systems in real-time signal procesing 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**, (I4), 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). Heterogeneous and homogeneous parallel architectures for real-time active vibration control. *IEE Proceedings-D: Control Theory and Applications*, **142**, (6), pp 1-8.

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.