X

Department of
AUTOMATIC
CONTROL
and SYSTEMS
ENGINEERING
UNIVERSITY of SHEFFIELD

COMPARATIVE STUDY OF SEQUENTIAL AND PARALLEL IMPLEMENTATIONS OF A DOPPLER SIGNAL SPECTRAL ESTIMATOR

M. M. Madeira *, M. O. Tokhi +, M. Graça Ruano *

* Area Departamental de Engenharia Electronica e Computação, UCEH, Universidade do Algarve, Portugal.

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

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

Research Report No. 733

November 1998

Abstract

Doppler signal spectral estimation has been used to evaluate blood flow parameters in order to diagnose cardiovascular diseases. The Modified Covariance (MC) method has proved to provide accurate estimation of the two spectral parameters employed in clinical diagnosis, namely mean frequency and bandwidth. The aim of the work reported in this paper is to determine an efficient real-time implementation of the MC spectral estimator by investigating several architectures and implementation methods. A comparative performance analysis of the implementation of the MC algorithm on several homogeneous and heterogeneous architectures incorporating transputers, digital signal processing (DSP) devices and a vector processor is reported. The performances of these architectures are evaluated and compared in terms of computational time (execution and communication) and gradient measurements. Analysis of the results reveals that both the DSP-based parallel architectures meet the real-time requirements.

Key words: *Digital signal processing, Doppler signal spectral estimator, heterogeneous architectures, high performance computing, homogeneous architectures, real-time signal processing, sequential processing, parallel processing, spectral estimation.*

CONTENTS

Title i

Abstract ii

Contents iii

List of figures iv

1 Introduction 1

2 The algorithm 2

3 Hardware architectures 3

4 Performance measures 4

5 Implementations and results 5

5.1 Algorithm characterisation 5

5.2 Implementations with the T8s 7

5.3 Implementations with the C40s 11

5.4 Implementations with the i860 12

5.5 Implementations with the heterogeneous architecture 13

5.6 Communication 14

5.7 Comparison of performance of the architectures 15

6 Concluding remarks 15

7 Acknowledgements 16

8 References 16

LIST OF FIGURES

Figure 1: Matlab implementation of the algorithm; (a) CPU time elapsed. (b) Number of floating point operations.

Figure 2: Number of floating-point operations per second using Matlab.

Figure 3: Regularity of the MC algorithm.

Figure 4: Comparative task regularity of the MC algorithm.

Figure 5: The homogeneous architecture of T8s.

Figure 6: Algorithm mapping on a single T8 architecture.

Figure 7: Execution times of a single T8 in implementing the MC algorithm.

Figure 8: Communication times in loading and storing data.

Figure 9: Execution times in implementing each task on a single T8.

Figure 10: Single-task and multi-task model implementations on a single T8.

Figure 11: Execution times of the algorithm considered as two and three data segments on two and three T8s.

Figure 12: Implementation gradient with the T8.

Figure 13: Communication times between two T8s.

Figure 14: Algorithm mapping on a single C40.

Figure 15: Execution times in implementing the MC algorithm on a single C40 as measured on the T8 and the C40.

Figure 16: Algorithm mapping on the homogeneous architecture of three C40s.

Figure 17: Execution times in estimating three data segments on three C40s (measured on the T8).

Figure 18: Communication times between the T8 and the C40.

Figure 19: Communication and conversion times between the T8 and the C40.

Figure 20: The i860 architecture.

Figure 21: Process of building an i860 application (Chambers, 1995).

Figure 22: Execution times in implementing the MC algorithm on the i860.

Figure 23: Effect of compiler optimisation on execution time of the i860.

Figure 24: Effect of compiler optimisation in implementing the algorithm on the I860; (a) Implementation gradients. (b) Speedups.

Figure 25: Communication times between the T8 and the i860.

Figure 26: The heterogeneous architecture.

Figure 27: Execution times in implementing the algorithm on the heterogeneous architecture.

Figure 28: Comparison of communication times between the processing elements.

Figure 29: Gradient of communication between the processors.

Figure 30: Effect of code optimisation on communication time between the T8 and the i860.

Figure 31: Execution times of the different architectures.

1 Introduction

Spectral estimation methods are particularly used in Doppler ultrasound cardiovascular disease detection. Clinical diagnosis procedures generally include analysis of a graphical display and parameter measurements, produced by blood flow spectral evaluation.

Ultrasonic instrumentation typically employ Fourier based methods to obtain the blood flow spectra, and blood flow measurements. These computationally efficient Fourier based methods, when applied to signals such as consecutive blood flow cardiac cycles, suffer from poor spectral resolution, due to the windowing effects caused by the segmentation of the signal into blood flow data segments. Typically, these data segments present short lengths, around 10ms, to preserve stationarity properties of the signal (Fish, 1991). Since blood flow velocity (spectral mean frequency) and blood flow disturbance (spectral bandwidth) (Evans, 1992), are parameters generally employed for clinical diagnosis, more accurate methods of minor stenosis detection have been investigated (Ruano and Fish, 1992; Vaitkus *et al.*, 1988).

A comparison of the statistical performance of several spectral estimators on the evaluation of spectral mean frequency and bandwidth, when a simulated common carotid artery blood flow signal is considered has revealed that the autoregressive modified covariance (ARMC) spectral estimator (Kay, 1988) gives more accurate spectral parameter estimates, taking into account the algorithmic computational burden of each spectral estimator considered (Ruano and Fish, 1993). The cost of using a parametric spectral estimator, in particular the ARMC algorithm, against the traditional FFT based methods, is the increase in computational burden, leading to spectral evaluation computational times greater than those desired for real-time application of the spectral estimator (Ruano *et al.*, 1992b). This time is 10ms.

Research into the real-time implementation of the MC spectral estimator has led to the development of a simplified new version of the algorithm (Ruano *et al.*, 1992a). Parallel implementation on a transputer based platform has revealed that real-time implementation of this accurate blood flow spectral estimator is feasible (Ruano *et al.*, 1993). This has motivated further research to enable utilisation of the algorithm over all clinically admissible ranges of blood flow velocities. Another approach to this problem has been reported in (Ruano *et al.*, 1992b), where instead of implementing a single data segment spectral estimator in parallel, several data segments are spectral estimated concurrently. This has resulted in better execution times, yet without reaching the real-time requirements for all clinical signal cases under test. These results have motivated further study into the real-time implementation of the algorithm. It has been demonstrated that, for an efficient implementation of an algorithm to be achieved in real-time, a close match has to be forged between the capabilities of the architecture and the computing requirements of the algorithm (Tokhi *et al.*, 1995).

Several sequential and parallel architectures are investigated for real-time implementation of the MC blood flow spectral estimator. These incorporate the Inmos T805 (T8) transputers, the Texas Instruments TMS320C40 (C40) DSP devices and the Intel i860 (i860) vector processor. The algorithm is implemented on these architectures and the performance of each architecture assessed. Finally, a comparative performance evaluation of the architectures in the real-time implementation of the algorithm is presented and discussed.

2 The algorithm

The algorithm considered in this work is the ARMC algorithm (Kay, 1988). The MC spectral estimator enables the estimation of the power spectral density, $P_{MC}(f_n)$, of a discrete time signal by evaluating

$$P_{MC}(f_n) = \frac{\sigma^2}{\left|A(f_n)\right|^2} = \frac{\sigma^2}{\left|1 + \sum_{k=1}^{p} a[k] z^{-k}\Big|_{z=e^{j2\pi f_n}}\right|^2} \tag{1}$$

where σ^2 denotes the signal white noise variance estimate and $A(f_n)$ is a 'p' polynomial function of the autoregresive parametric spectral parameter estimates, $a[k]$.

The signal white noise variance estimate is calculated as

$$\sigma^2 = c_{xx}[0,0] + \sum_{k=1}^{p} a[k] c_{xx}[0,k] \tag{2}$$

The $a[k]$, $k = 1,\dots,p$, are estimates obtained by minimising the average powers of the forward and backward prediction errors. This is achieved by solving the MC equations

$$[c_{xx}[i,j]].[a[i]] = [c_{xx}[i,0]] \tag{3}$$

where each element $c_{xx}[i,j]$ of the covariance matrix and the right-hand-side vector in equation (3) are obtained as

$$c_{xx}[i,j] = \frac{1}{2(N-p)}\left(\sum_{n=p}^{N-1} x^*[n-i]x[n-j] + \sum_{n=0}^{N-1-p} x[n+i]x^*[n+j]\right) \tag{4}$$

with x and N representing the signal under consideration and the length of the data segment respectively. The MC algorithm thus considered in this work comprises equations (1)-(4).

3 Hardware architectures

The hardware architectures considered in this study comprise unit-processor and multi-processor architectures incorporating three types of processing elements; namely, the Inmos T805 transputer (T8), the Texas Instruments TMS320C40 DSP device (C40) and the Intel 80i860 vector processor (i860). These are briefly described below.

The T8 is a 32-bit RISC processor with 20-25 MHz clock speed, up to 20 MIPS, 1-4 Kbytes dedicated RAM, a 64-bit floating point unit capable of 4.3 MFLOPS. It has two real-time clocks (a low priority running at 15.625 kHz and a high priority running at 1 MHz), four bit-serial bi-directional links at a speed of 20 Mbit/sec for communication with other processors (Inmos Ltd., 1988).

The C40 is a high-performance 32-bit parallel device. Its main characteristics are 40 MHz clock speed, 8 KB dedicated RAM, rated as 275 MOPS and 40 MFLOPS. It has six high-speed parallel unidirectional links for communication with other processors and an asynchronous transfer rate of 20 Mbytes/sec (Texas Instruments, 1991).

The i860 is a RISC processor with large data (8 Kbytes) and instruction (4 Kbytes) caches. It is a 64-bit floating-point processor with 40 MHz clock speed, up to 40 MIPS and 80 MFLOPS. It is contained in a TTM110 i860 TRAM that includes a T8 running at 25 MHz and 16 Mbytes shared memory (Transtech Parallel Systems Ltd., 1992).

A parallel architecture comprising processing elements of the same nature is referred to as a homogeneous architecture. If the architecture has processing elements of different nature then it is referred to as a heterogeneous architecture. The parallel architectures utilised in this work include a homogeneous architecture of T8s, a homogeneous architecture of C40s and a heterogeneous architecture comprising three C40s and a single i860. As noted later, a T8 is used in the parallel architectures as a root processor for downloading programmes and for communication between the host and the architecture.

4 Performance measures

A parallel system is conventionally evaluated in terms of execution time and speedup. The execution time is especially relevant when real-time is required. Speedup, S, is the ratio of execution time of a single processor, T_{1P}, over the execution time of a parallel architecture, T_{nP}, of n processors in implementing a given task;

$$S = \frac{T_{1P}}{T_{nP}} \tag{5}$$

Since the task size is fixed, the definition in equation (5) is referred to as fixed-load speedup.

An alternative measure of performance of an architecture is gradient (Tokhi *et al.*, 1997a,b). The gradient, G, is a measure of the mean execution time, $T(m)$, per data element, m;

$$G = \frac{T(m)}{m} \tag{6}$$

Gradient can be applied either to execution or communication times, and varies according to the match between algorithm and architecture. In this manner, it provides scope for optimisation.

5 Implementations and results

In this section results of investigations involving characterisation of the MC algorithm and its implementation on the various hardware architectures are presented. At the algorithm characterisation level, Matlab was utilised. Matlab is a computer aided analysis and design package commonly used in engineering and scientific applications of signal processing and control (The Math Works Inc., 1991). At the hardware implementation level, different homogeneous and heterogeneous parallel architectures were devised and investigated using the parallel processing elements introduced earlier. Initially only the T8s were considered. Different task allocations were experimented. The C40s were then tested in sequential and parallel forms and then, the i860 was tested. Finally, a heterogeneous architecture comprising three C40s and one i860 was investigated.

With the multiprocessor architectures considered, the work load was distributed among the available processors with each processor dedicated to the estimation of a data segment. To evaluate the communication time overhead of the processors in implementing the algorithm, investigations on communication times between processors were carried out. The times thus obtained correspond to the forward and backward transfer (round trip) of floating type data segments. Considering the range of typical sampling frequencies in this particular application the data segment lengths considered are 64, 128, 256 or 512 points. These times were measured at different points on the architecture wherever differences could be sensed.

5.1 Algorithm characterisation

The MC algorithm, as described earlier involves manipulation of equations (1)-(4). The selection of a suitable model order is typically considered a drawback of parametric methods. Previous studies on the performance of several methods at estimating normal carotid artery blood flow spectra have resulted parametric model order selection by making use of the cost/benefit criterion (Ruano and Fish, 1993). The signals under analysis, in these studies, typically comprised mean frequency values ranging from 1 kHz to 8 kHz in octaves, each with bandwidths of 5, 10 and 20% of the corresponding mean frequency value. Taking into account all 12 signal cases, the cost/benefit criterion identified a 4th order MC model as the more accurate spectral estimator.

The MC algorithm estimates the spectrum of a signal corresponding to a data segment. Each data segment has a predetermined number of data points (64, 128, 256 or 512). The resulting output data segment, being symmetrical, will have half the length of the input data segment.

The approach used to partition the algorithm considers the block of data-time segments strategy (Ruano *et al*, 1992b). It has been demonstrated that this approach would permit a large range of data segment lengths in homogeneous transputer architectures. Therefore, each processor takes up a data segment of input data, starts processing the MC algorithm and outputs the spectrum.

To establish a reference for comparison of accuracy of results in subsequent investigations, the algorithm was implemented using Matlab. The algorithm computational complexity was evaluated in terms of time and floating point operations ("CPU_time" and Matlab "Flops" functions). Results of this study are shown in Figure 1.

For the algorithm to be easily mapped onto different architectures, it was partitioned into the following modules:

- module RMC - calculation of the elements of the covariance matrix and right-hand side vector in equation (4);
- module Cholesky - resolution of the linear system of equations in equation (3) using the Cholesky method;
- module Wnv - calculation of the white noise variance, equation (2); and
- module Psd - the power spectral density, equation (1).

To evaluate the possibility of implementing these modules on different processors, the number of flops involved in each module was evaluated. This is shown in Figure 2. Since the model order was fixed, module Cholesky, as expected, spent the same number of flops irrespective of the data segment length. Modules RMC and Wnv depend on data segment length, showing a linearly increasing computational load rate with increasing data segment length. The Psd module depends on the data segment length, as it incorporates implementation of a fast Fourier transform (FFT) algorithm. These modules were coded in high-level languages, and the code was kept as simple as possible with a transcription of equations (1)-(4). It is noted that the calculation of the covariance matrix constitutes the dominant part of computation.

The regularity of the algorithm is based on the response of the algorithm in terms of increasing load (in this particular case, data segment length). A reference value, the ratio between the data segment size and its minimum value (64) was used for comparison purposes with the results in Figure 1(b). This reference is, therefore, a proportional coefficient. The corresponding regularity of the algorithm is shown in Figure 3. It may be concluded from this result that the algorithm is regular, showing a linear relation between data segment size and computation load, expressed here in flops.

Task regularity of the algorithm was similarly obtained using the results in Figure 2. This is shown in Figure 4. As with Figure 3, a reference was included, representing the proportional coefficient. As the number of flops of the Cholesky module does not vary with data segment size, this module was excluded from the diagram. It can be assumed that the tasks that implement the MC, Wnv and Psd modules are regular.

5.2 Implementations with the T8s

Three transputer modules were installed on the TMB08 motherboard, enabling the use of one to three transputers. As described previously, each transputer has four bi-directional links to communicate with other transputers. The topology of the architecture utilised is shown in Figure 5. The cabling of the interprocessor links and confidence test is ascertained with the *check* command. This will activate a test program that will verify all the links and identify which ones are used and with which processor including the root processor.

For the architecture shown in Figure 5, the *check* command produces the following description:

```
total of 3 processors found
processor ROOT type=T800 20.0MHz, 2.0 penalties, 0 hops 1024K
links to HOST[0],-------,P001[1],P002[1]
processor P001 type=T800 20.0MHz, 2.0 penalties, 1 hop 1024K
links to -------,ROOT[2],-------,-------
processor P002 type=T800 25.0MHz, 2.0 penalties, 1 hop 4096K
links to -------,ROOT[3],-------,-------
```

A function similar to the *check* command is performed by the *worm* command. This provides a more detailed description of the architecture;

```
# Part rate Mb Bt      [ Link0        Link1   Link2   Link3  ]
  0 T805b-20 1.75 0   [  HOST       1:1     3:1     4:1     ]
  1 T2   -17 1.74   1  [  ...        0:1     ...     C004    ]
  3 T805b-20 1.74 1   [  ...        0:2     ...     ...     ]
  4 T805d-25 1.32 1   [  5:0        0:3     ...     ...     ]
  5 T2   -20 1.75   0  [  4:0        ...     ...     C004    ]
```

where the T2's are T200 transputers. These are responsible for communication with other processors and can not be used for computation.

The process of building an application consists of

- compiling the different modules of code (t8c);
- linking the compiled modules into tasks (t8ctask, t8cstask);
- configuring according to a configuration file (config).

A typical configuration file includes

- the hardware declarations, i.e. processor types and wiring;
- task declarations, including number of input and output ports and memory requirements;
- software task assignment to physical processors;

- connections between tasks.

To implement the algorithm on a single transputer, the different modules of code (tasks) comprising the algorithm were mapped onto the architecture according to Figure 6, where *afserver* and *filter* are system files. Note that *Stdio* is not a task. This is represented in the configuration since the use of input or output is only possible from the task on the Root transputer and requires counting an extra channel for that purpose. *AS1T* is the main task of the algorithm implementation on the transputer. *AS_RMC*, *Cholesky*, *WNV* and *PSD* are the computation tasks corresponding to the modules RMC, Cholesky, Wnv and Psd respectively, as described earlier.

The execution times achieved in implementing the algorithm on the architecture are shown in Figure 7. The algorithm, as expected, exhibits a linear execution-time to load relation. Data flow in this implementation includes only the load of the data segments and the storage of the spectra. No other communication is involved in this implementation because all tasks are implemented on the same processor and communication between tasks consists of passing data addresses. The resultant estimated spectra correspond to only half the size of the input data segment. However, as can be seen in Figure 8, the loading and storing of data takes a considerable amount of time. It is noted that the output time shows a smaller growth rate with data segment size than the input time. This is probably due to memory allocation during the input process. This issue will be addressed in future investigations.

The use of more than one processor usually implies the partitioning of the algorithm into sub tasks and the allocation of these tasks to processors. As stated earlier, the algorithm has been decomposed into several modules in order to be easily mapped onto different architectures. The execution time of each task would allow the determination of the computing requirements of each. The execution times in implementing each task on a single T8 are shown in Figure 9. As noted, Psd is the most time consuming task. This task implements an FFT the execution time of which depends on the particular algorithm used. As FFT algorithms have been thoroughly studied and optimised, it was thus convenient later to adopt data parallelism rather than task parallelism for multiple processor implementations. An alternative to task partitioning by treating the whole algorithm as one task was also investigated. As stated earlier, previous studies using a different setting indicate that this approach would lead to better execution times (Ruano *et al.*, 1992b).

Figure 10 shows the execution times of the T8 in implementing the algorithm considered as a single-task and a multi-task model. It is noted that the execution times for the multi-task model are more than the execution times of the single-task model. The difference in execution times is probably due to context switching (Galletly, 1990). Time slices of processor time are attributed to each task. Although this is partially done in hardware, a certain amount of time is spent in the process of saving and restoring of task environment during the switch. Note also that the increase in total execution time with the multi-task model is due to communication time between the tasks. When implementing the whole algorithm as a single task, a significant decrease of execution time is achieved, mainly with higher data segment lengths.

Implementations of the algorithm on two and three transputers were investigated with the calculation of two and three data segments allocated to two and three T8s respectively. The major advantage of this single task approach is a scaleable solution, not a significant increase of execution time when doubling/tripling the processing capacity for the double/triple data processing load. The execution times of these implementations are shown in Figure 11. These values include communication times.

To assess the performance of the T8s in implementing the algorithm, the gradient is evaluated using the results of Figure 7 and Figure 11. This is shown in Figure 12. It is noted that, using two transputers, each element is, roughly, calculated in half the time; using three transputers, the time is reduced to a third, in comparison to that with a single T8.

Samples of communication times between two T8 were also obtained. These were measured on the first transputer of the architecture using a different application program. The results thus obtained are shown in Figure 13. These will be compared with the results obtained later from posterior implementations.

5.3 Implementations with the C40s

The process of building an application for implementation on the C40 consists of compiling, linking and configuring, in a similar manner as considered for the T8. This is shown in Figure 14. The modules that will run on the C40 are compiled and linked using *c40c* and *c40cstask*. TIO represents the input and output task. AL1C represents the estimation of the spectra. The T8s in Figure 14 are not involved in the actual spectral estimation. These are used for sending data to and receiving data from the C40. The homogeneous architecture of C40's is also shown in Figure 14. The different implementations investigated maintain the same wiring but differ in the task allocation and mapping.

The execution times of a single C40 in implementing the algorithm are shown in Figure 15. As previously stated, the T8 is not involved in the actual spectral estimation. It is noted that the measured times taken from the module running on the T8 and on the C40 differ substantially. The additional time incurred is due to input/output data communication between the T8 and the C40.

An implementation using three C40s for the simultaneous estimation of three data segments was achieved with the architecture shown in Figure 16. The same approach, as with the T8s, of distributing the work load and dedicating each processor to the calculation of a data segment was investigated and the links between the C40s were not used. In this process, three data segments are distributed by the T8 to the three C40s and, upon completion returned to the host. The corresponding execution times for the estimation of three data segments, including communication times, are shown in Figure 17. As compared with the results in Figure 15, the execution time, measured on the T8, has increased slightly due to communication. To investigate this further, experiments with a different application program were conducted to obtain the communication times between the T8 and the C40 with different data segment sizes. Results of these experiments are shown in Figure 18. These values correspond to the communication of raw data (to/from), between the T8 and the C40. The difference in time measurements in the T8 and the C40 (Figure 15) can also be due to the data conversion and communication set-up time. Note that the C40 has a proprietary Texas Instruments (TI) data format. This means that it is necessary for the C40 to convert the IEEE to TI data format when receiving data and vice versa before sending. It is also relevant that the communication between the T8 and the C40 has to go through a serial to parallel adapter and the T8 is only capable of synchronous communication.

Figure 19 shows the times necessary for the communication and conversion of data segments of different sizes between the T8 and the C40. These differences strongly suggest further investigations with homogeneous architectures of C40s without the use of a T8 root processor.

The gradients with implementations of the algorithm on one and three C40s, obtained using the results in Figures 15 and 17, are 0.32 ms and 0.11 ms respectively. This indicates that the task allocation adopted is suitable for this architecture, since the average computation time with three C40s is a third of that achieved with a single C40.

5.4 Implementations with the i860

The i860, as stated earlier, is contained in a motherboard with a T8. The architecture is shown in Figure 20. The procedure for building an i860 application is more elaborate than those presented for the T8 and the C40. The compiler used is the Portland Group C Compiler (*pgcc*) and runs in DOS. The object code is transferred to the host via binary ftp. Figure 21 shows a summary of the stages involved. The blocks represent the commands issued on original files and the respective results (Chambers, 1995).

As noted, three different configuration files are required: one for the i860 and its T8, another for the connection of the transputers on different boards and a third for the main board. In this manner, two applications are running simultaneously; one booting and loading the i860 application on the TMB16 board and the other booting and loading the Host transputer on the TMB08 board (see Figure 20). The algorithm was implemented on the i860 only. The execution times, as measured on the i860, are shown in Figure 22.

Previous work on the i860 has indicated a clear difference in execution times with and without the use of the option of Optimisation Level of the compiler (Chambers, 1995). Figure 23 shows the execution times (in ms) and file size (FS) of the same source code with optimisation levels 0 to 4. It is noted that substantial improvement in execution time is achieved with higher levels of code optimisation.

The execution times obtained in implementing the algorithm on the i860 are much better than those with the T8 and the C40. It is noted in Figure 23 that the compiler optimisation has resulted in a reduction of the execution time. Among the various levels, optimisation levels 3 and 4 produce the best results; around 34% of the execution time with no optimisation.

Figure 24 shows the gradients and corresponding speedups achieved with the various levels of code optimisation in implementing the algorithm on the i860. It is noted that higher data segment sizes take more advantage of the compiler optimisation than lower data segment lengths. The i860 is capable of very fast processing especially at higher data segment lengths (best results around 10 microseconds per data segment were achieved).

To investigate the communication time between the i860 and the T8, the various levels of optimisation were used. Figure 25 shows the communication times, in ms, thus obtained. These indicate that inter-processor communication is not significantly affected by code optimisation as it can be noted that for the data segment lengths considered the variation is less than one microsecond.

5.5 Implementations with the heterogeneous architecture

The heterogeneous architecture considered is shown in Figure 26. This incorporates the three different parallel processing elements. Only the C40s and the i860 are used to perform the actual computation, the T8s are used as routers. This was achieved by connecting together the three C40s and the i860 with the two executable files, considered as "black boxes". Figure 27 shows the results thus achieved in implementing the algorithm on the architecture. These results correspond to the parallel computation of four data segments, distributed among the C40s and the i860. In this implementation the use of the second transputer in the architecture as a relay between the root transputer and the C40s reduces the overall execution time. The first three data segments, as they were received by the root transputer were relayed to the second transputer that would, then, distribute them by the C40s and receive the output values. The fourth data segment is sent to the i860 and its output values received by the root transputer. Therefore, the execution times presented in Figure 27 are measured on the root transputer for the i860 and on the relay transputer for the total of C40 execution. The execution times corresponding to the T8 were measured from the beginning of data transfer to the last data received. Comparing with the results previously obtained on the homogeneous architecture of C40s it can be noted that the explicit use of both transputers for communication purposes actually leads to better results. It can be concluded that this data parallel approach achieves an acceptable data throughput within the required time range for a clinical real time application.

5.6 Communication

Communication times are relevant in the choice of the most adequate parallel architecture for a particular algorithm. The communication times considered in this work correspond to the round trip (forward and backward) of data segments of floating type and variable dimensions (64, 128, 256 and 512). It must be noted that the T8 and the C40 are 32-bit processors and the i860 is a 64-bit processor. A T8 has been utilised throughout as the root processor, connected to the host computer and providing communication with the different processors. A comparison of the communication times of raw data between the T8 and the different processing elements, using the results in Figures 13, 18 and 25 (Opt Level 0), is shown in Figure 28. Data format conversion is necessary only when floating or double data format types are used. This is implemented in software (library functions) and depend on the data structure and compiler adopted. Thus, Figure 28 shows the transfer time of the data segments.

Figure 29 shows the gradients of communication between the processors. It is noted that due to serial to parallel communication in the T8-C40 there is a significant increase in communication times between the T8 and the C40, if conversion of data is also included. The shared memory communication between the i860 and the T8 gives the fastest communication between the two processors. The difference in communication time per data element between T8-C40 and T8-i860 is small (Figure 29). This difference will slightly increase with the utilisation of compiler optimisation, as is evident from Figure 30 representing the percentage of communication time spent for each data element when making use of compiler optimisation. The reference was level 0 (no optimisation). It is also noted in Figure 30 that the compiler optimisation results in a different pattern of communication times gradient from those observed with the execution times.

5.7 Comparison of performance of the architectures

Previous studies have shown that the i860 performs better in implementing algorithms of regular and matrix based nature. Similarly, the C40 is expected to achieve better execution times if the algorithm is irregular. It is also expected that the performance of the T8+C40 architecture will be relatively slower due to the serial to parallel communication links in the system (Tokhi *et al.*, 1995).

It was noted in the investigations carried out that best results were achieve when making simultaneous calculation of more than one data segment. Figure 31 shows the average execution times of the different architectures in implementing the algorithm. The legends 3T and 3C represent the average execution time corresponding to calculating each data segment on three T8s and three C40s respectively. The 3C+I is the average value that corresponds to the architecture using three C40s and one i860 for the calculation of four data segments. It is noted that best execution times are achieved with the 3C+I architecture.

6 Concluding remarks

A comparative study of different implementations of a Doppler Signal Spectral Estimator on uni-processor and multi-processor (homogeneous and heterogeneous parallel) architectures has been presented. The algorithm itself exhibits a very regular nature and comprises matrix-based operations. A T8 implementation does not meet the real-time constraints that apply to the spectral estimation of the signals. Therefore, such an implementation is not recommended.

As expected, it has been verified that the i860 and the C40 are suited for fast implementation of the algorithm and would successfully implement the algorithm in real-time.

The T8 constitutes a bottleneck in the C40 architectures. The execution times of the algorithm on the C40 are much better than it seems when measured on the T8. A detailed study of homogeneous architecture of C40s is thus recommended in future investigations.

For this particular application the i860 has achieved the best performance in respect of both the execution and the communication times. It has also been shown that the compiler optimisation results in an enhancement in both the execution and communication times. However, the resulting trends in the execution and communication times differ. In the latter case the best option depends on data segment length.

7 Acknowledgements

The authors gratefully acknowledge the financial support of Treaty of Windsor (B-115/96) and JNICT (PBIC/C/CEG/2414/95).

8 References

Chambers, C. (1995). *A study of performance issues in heterogeneous and homogeneous architectures*, The University of Sheffield, UK, MSc thesis.

Evans D.H. (1992). *Doppler ultrasound physics, instrumentation, and clinical application*, New York, John Wiley and Sons.

Fish P.J. (1991). *Nonstationary broadening in pulsed Doppler spectrum measurements*, Ultrasound Medicine & Biology, **17**, pp. 145-155.

Galletly, J. (1990). *Occam 2*, Pitman Publishing, London.

Inmos Limited (1988), *Transputer Architecture and Overview*, Manual of *Transputer Education Kit*, Inmos Limited.

Kay S.M. (1988). *Modern Spectral Estimation - Theory & Application*, Prentice-Hall, Englewood Cliffs.

Ruano M.G., Fish P.J. (1992). *Cost/benefit Selection of Spectral Estimators for use in a Doppler Blood Flow Instrument*, Proceedings of the IEEE 1992 International Conference on Acoustics, Speech, and Signal Processing, **V**, pp 513-516.

Ruano M. G, Garcia Nocetti D. F., Fish P., Fleming P.J. (1992a). *A Spectral Estimator using Parallel Processing for use in a Doppler Blood Flow Instrument*, Parallel Computing: from Theory to Sound Practice, ed.: W. Joosen and E. Milgrom, IOS Press, pp 397-400.

Ruano M.G., Garcia Nocetti D.F., Fish P., Fleming P.J. (1992b). *Parallel Implementation of an AR Estimator for Real-Time Ultrasonic Blood Flow Instrumentation*, International Conference on Parallel Computing and Transputers Applications PACTA 92, Barcelona, Spain, (Eds. Valero, M., Onate, E. et al), IOS Press/CIMNE, pp 337-345.

Ruano M.G., Fish P.J. (1993). *Cost/benefit Criterion for Selection of Pulsed Doppler Ultrasound Spectral Mean Frequency and Bandwidth Estimators*, IEEE Trans. Biomedical Engineering, **40**, (12), pp 1338-1341.

Ruano M.G., Garcia Nocetti D.F., Fish P., Fleming P.J. (1993). *Alternative Parallel Implementations of an AR Modified Covariance Spectral Estimator for Diagnostic Ultrasonic Blood Flow Studies*, Parallel Computing, **19**, pp 463-476.

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

The Math Works Inc. (1991). *Matlab users' guide*, The Math Works Inc., South Natick, MA01760, USA.

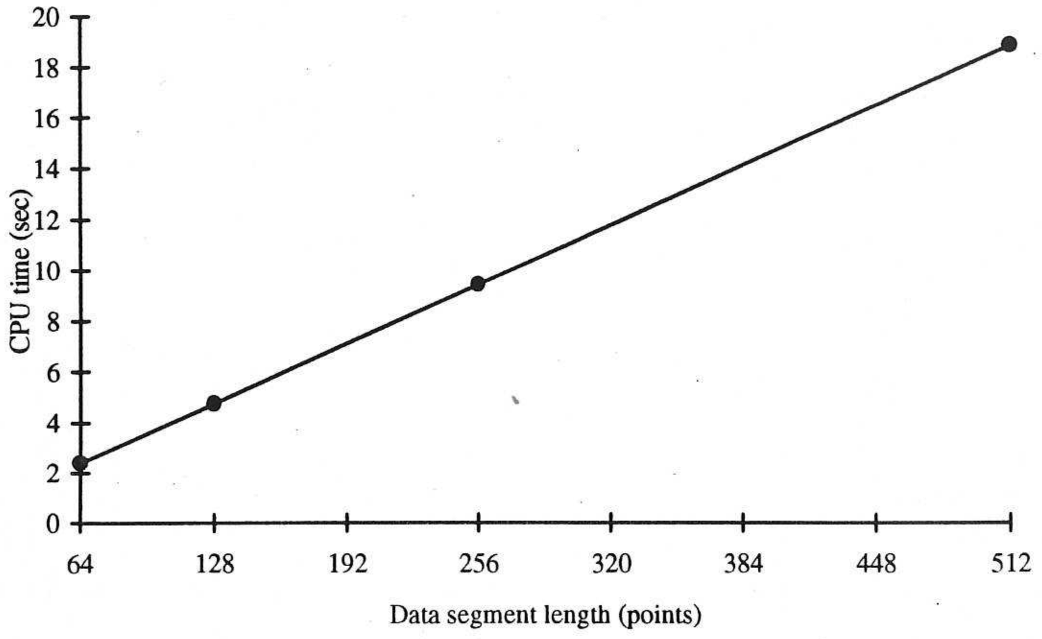
Tokhi M. O., Hossain M. A., Baxter M. J. and Fleming P. J. (1995). *Performance evaluation of homogeneous and heterogeneous architectures in real-time signal processing and control*, Preprints of IFAC Workshop on Algorithms and Architectures for Real-time Control, Ostend, 13 May - 02 June 1995, pp 575-578.

Tokhi, M. O., Hossain, M. A. and Chambers, C. (1997a). Performance evaluation of DSP and transputer based systems in sequential real-time applications. *Microprocessors and Microsystems*, **21**, pp. 237-248.

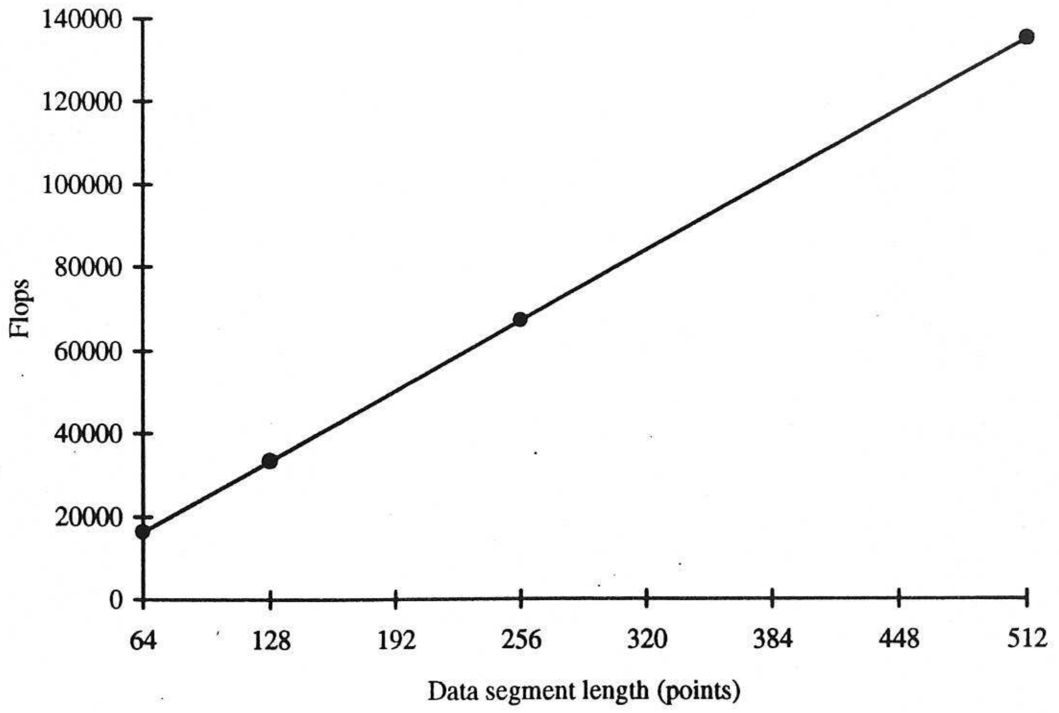
Tokhi, M. O., Ramos-Hernandez, D. N., Chambers, C. and Hossain, M. A. (1997b). Performance evaluation of DSP, RISC and transputer based systems in real-time implementation of signal processing and control algorithms. In Ruano, A. E. (ed.), *Proceedings of AARTC-97: 4th IFAC Workshop on Algorithms and Architectures*, Algarve (Portugal), 09-11 April 1997, pp. 287-292.

Transtech Parallel Systems Ltd. (1992). *The i860 Toolset,* Transtech i860 Parallel Programming System, Transtech Parallel Systems Limited.

Vaitkus P., Cobbold R., Johnston W. (1998). *A comparative study and assessment of Doppler ultrasound spectral estimation techniques. Part II: methods and results*, Ultrasound in Medicine & Biology, **14**, (8), pp. 673-688.



(a) CPU time elapsed.



(b) Number of floating point operations.

Figure 1: Matlab implementation of the algorithm.

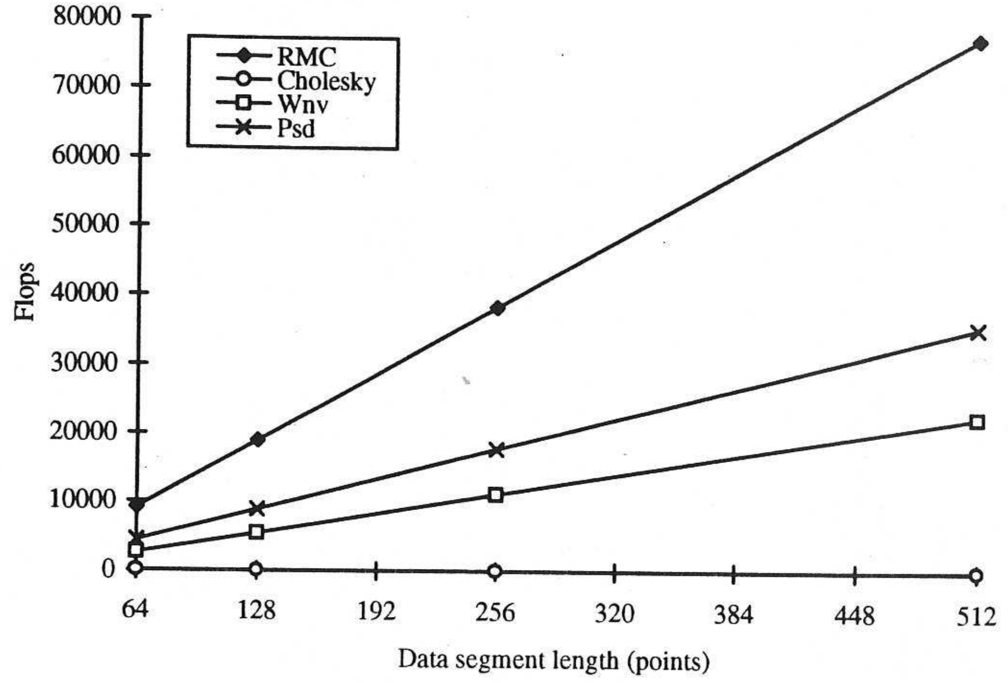


Figure 2: Number of floating-point operations per second using Matlab.

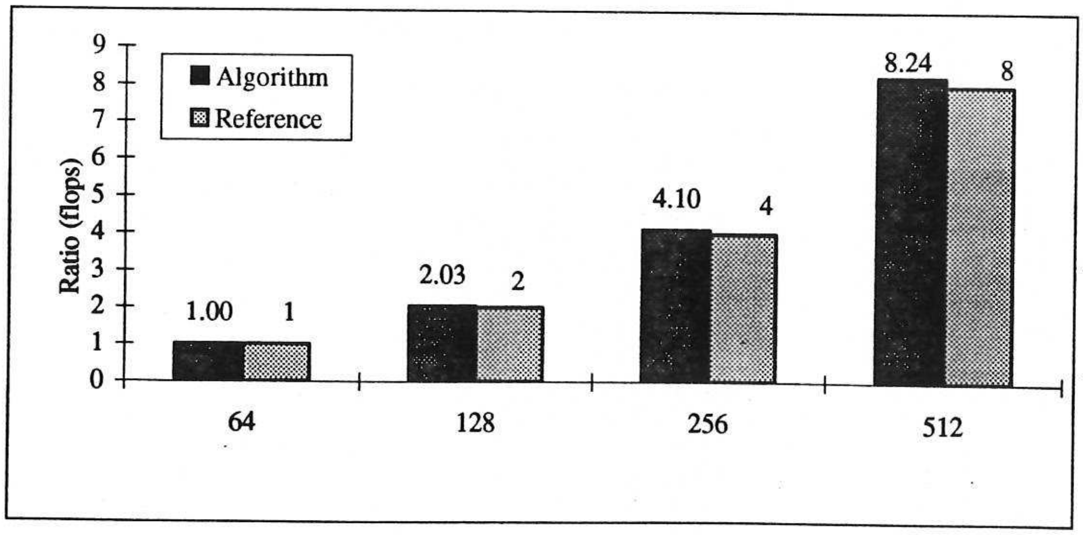


Figure 3: Regularity of the MC algorithm.

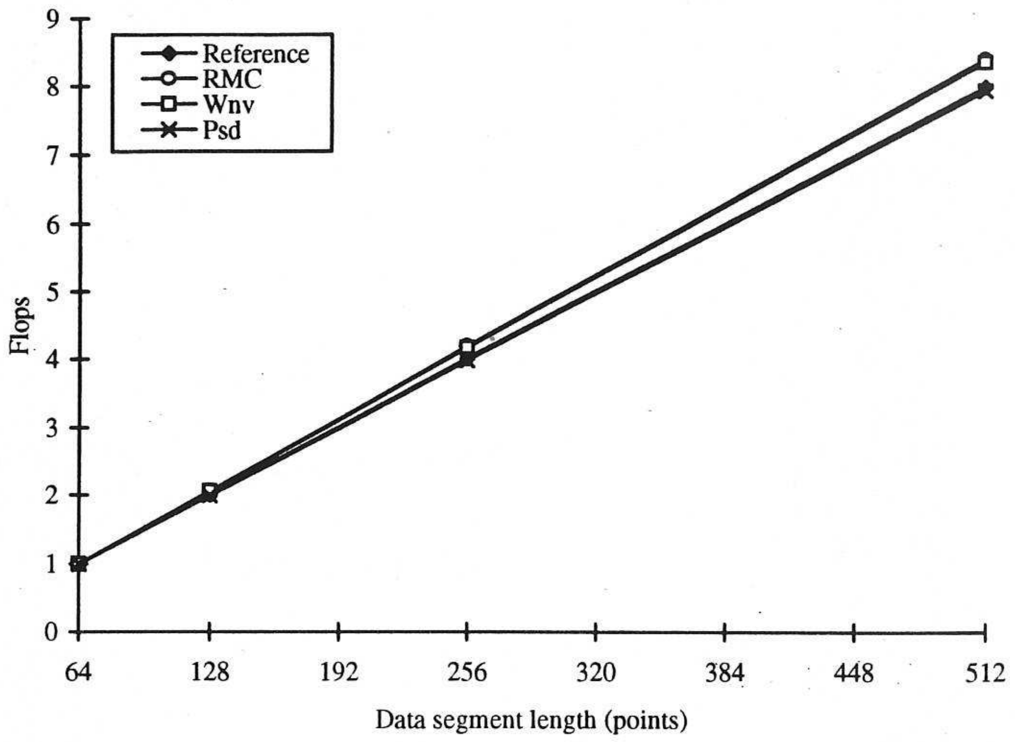


Figure 4: Comparative task regularity of the MC algorithm.

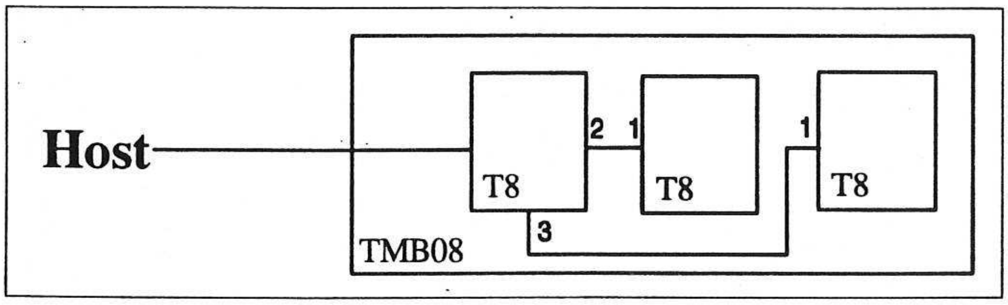


Figure 5: The homogeneous architecture of T8s.

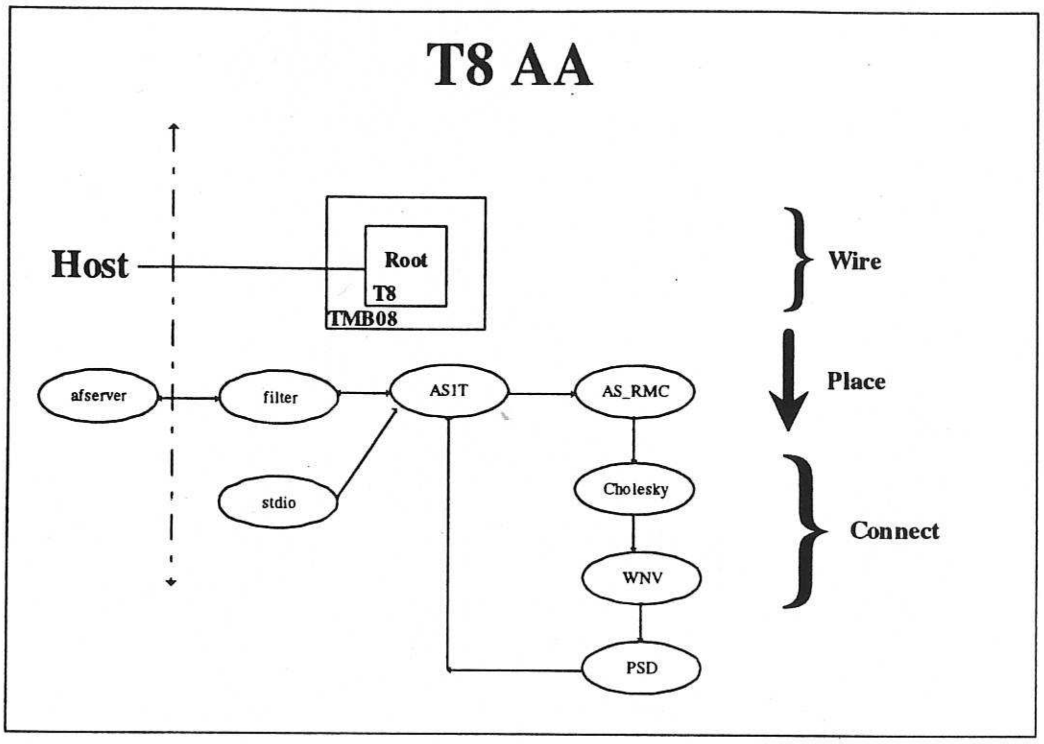


Figure 6: Algorithm mapping on a single T8 architecture.

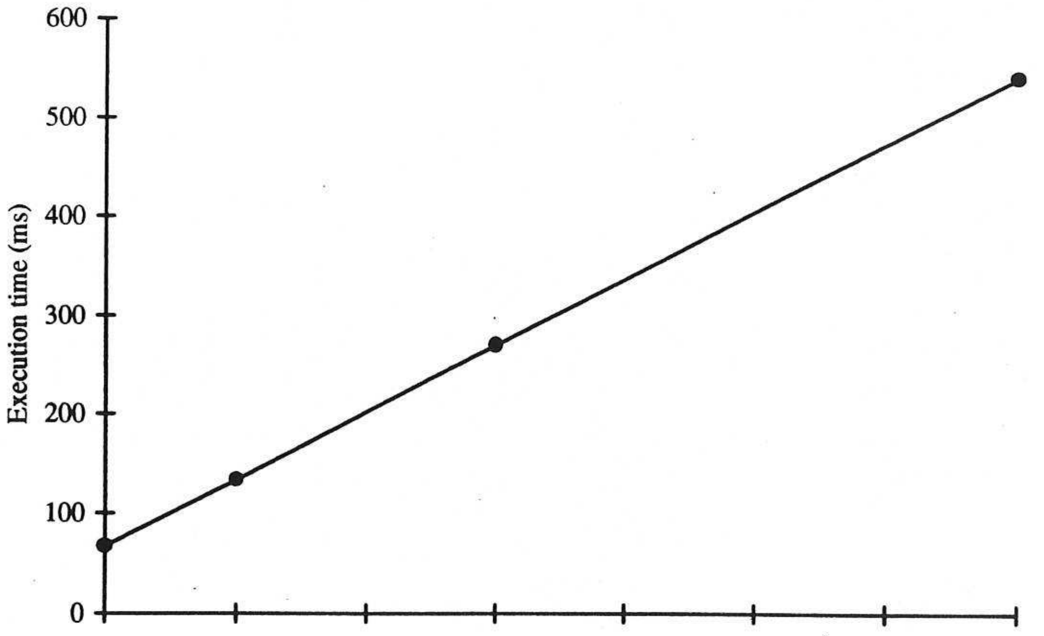


Figure 7: Execution times of a single T8 in implementing the MC algorithm.

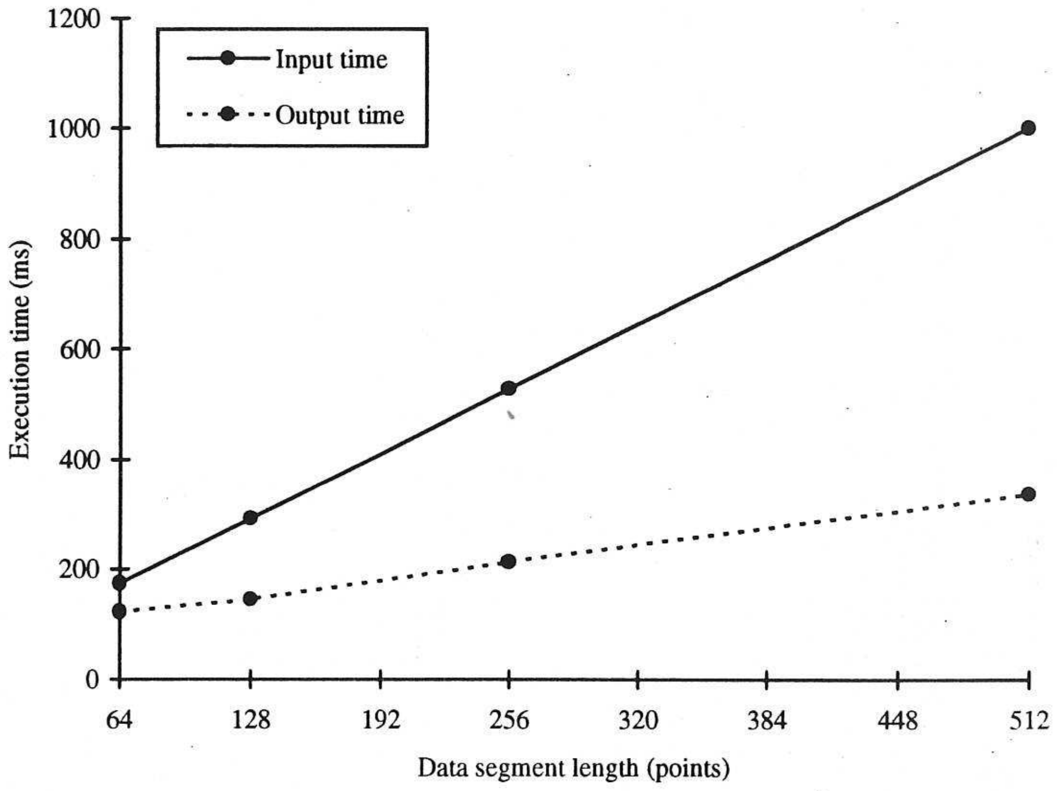


Figure 8: Communication times in loading and storing data.

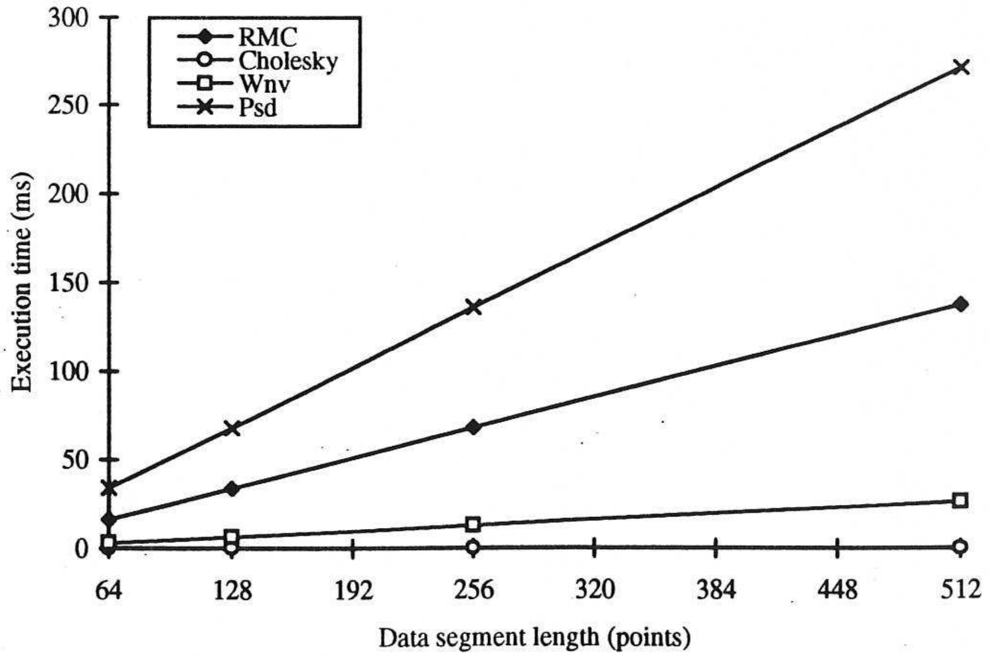


Figure 9: Execution times in implementing each task on a single T8.

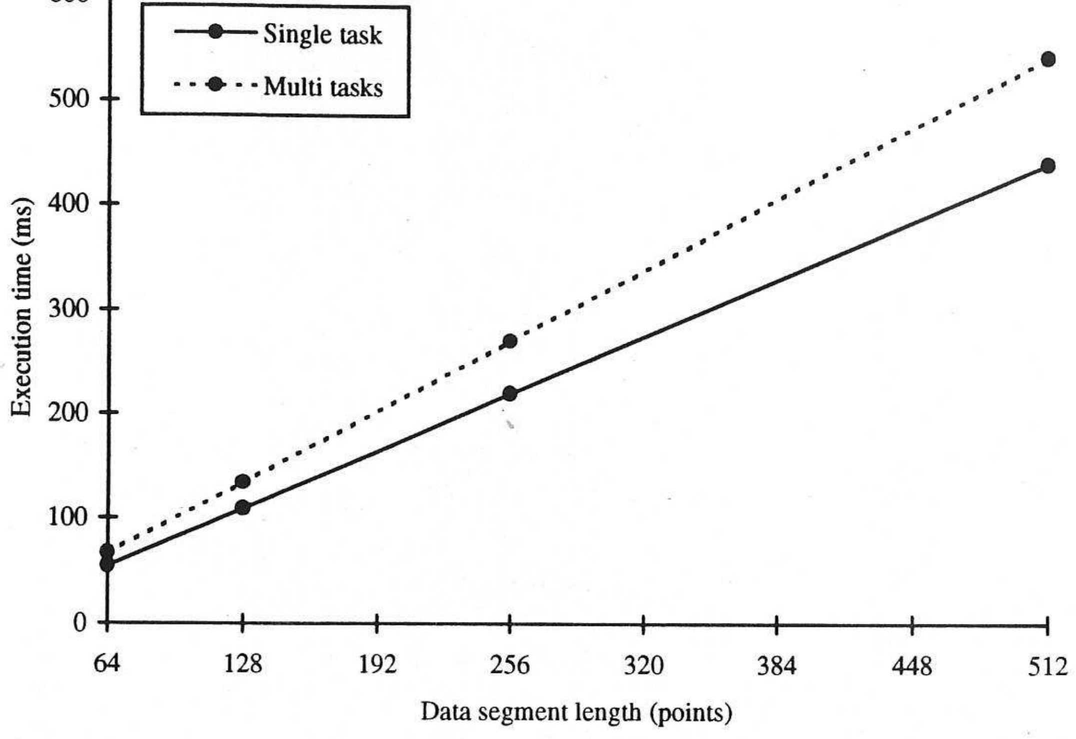


Figure 10: Single-task and multi-task model implementations on a single T8.

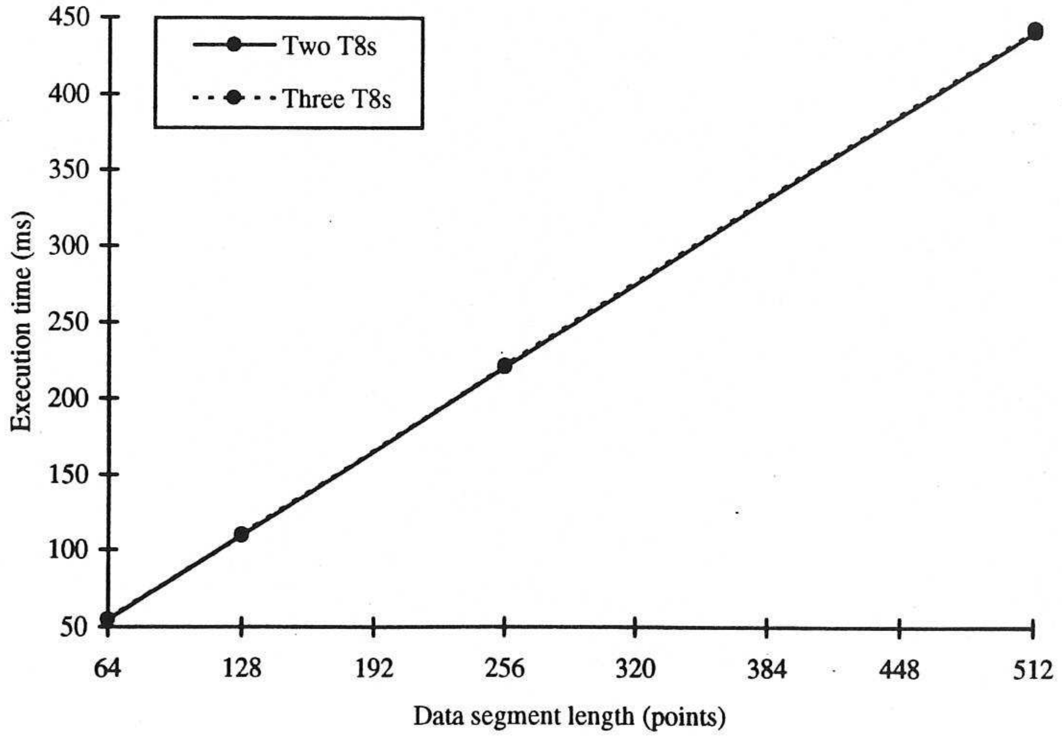


Figure 11: Execution times of the algorithm considered as two and three data segments on two and three T8s.

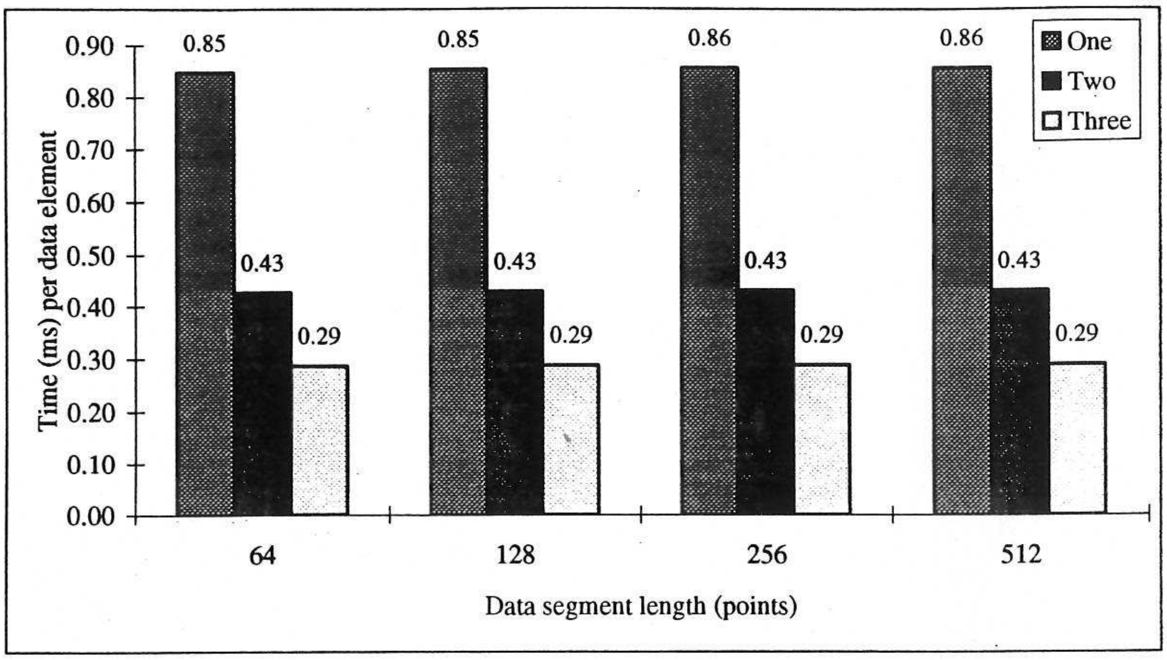


Figure 12: Implementation gradient with the T8.

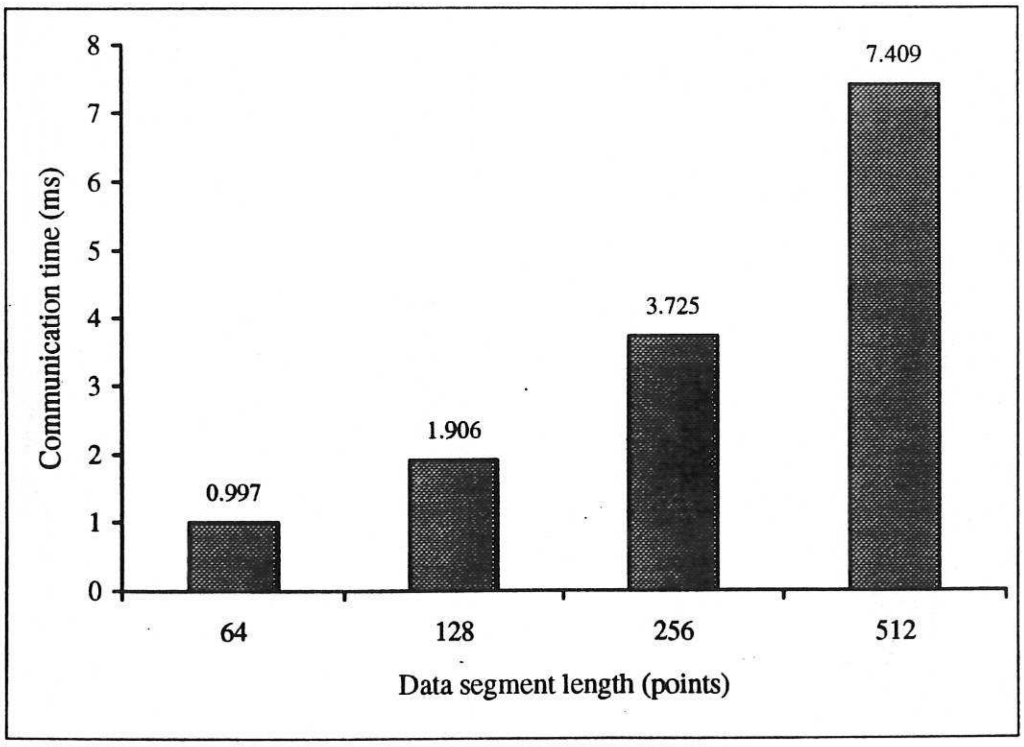


Figure 13: Communication times between two T8s.

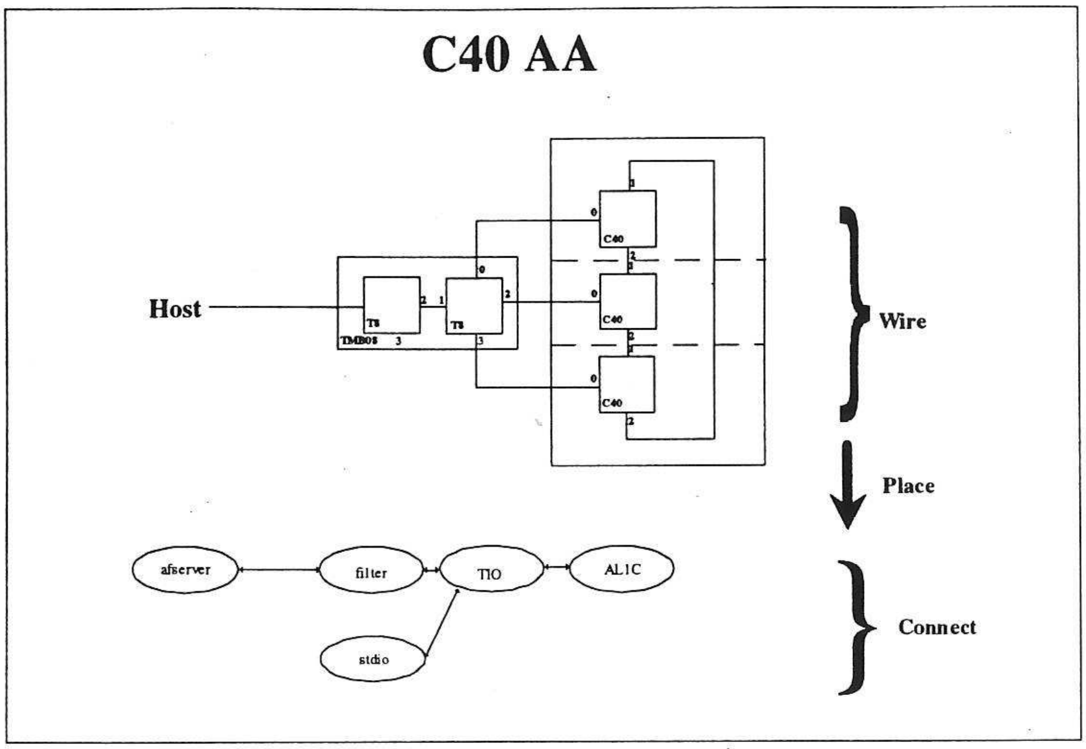


Figure 14: Algorithm mapping on a single C40.

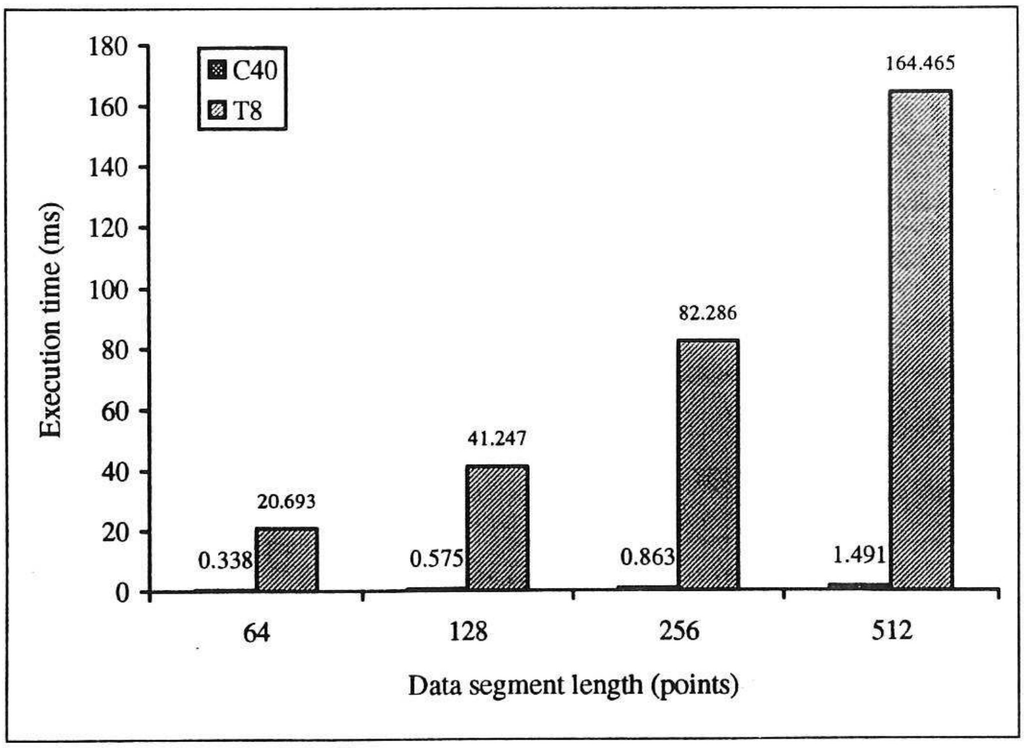


Figure 15: Execution times in implementing the MC algorithm on a single C40 as measured on the T8 and the C40.

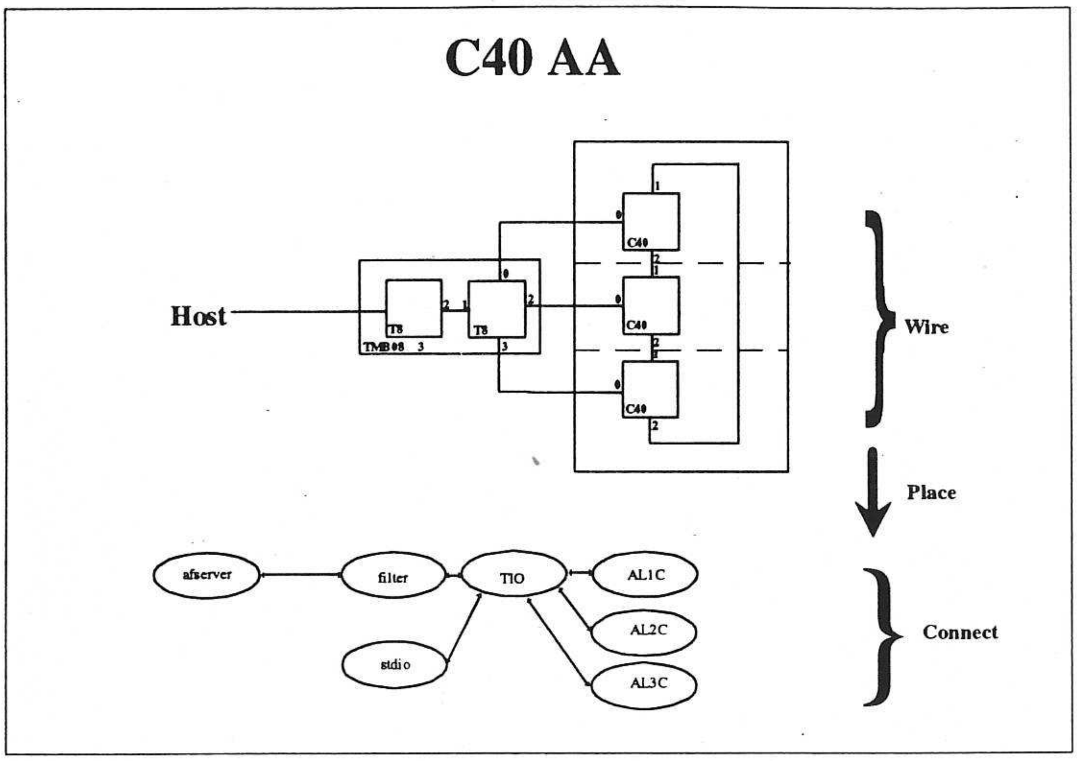


Figure 16: Algorithm mapping on the homogeneous architecture of three C40s.

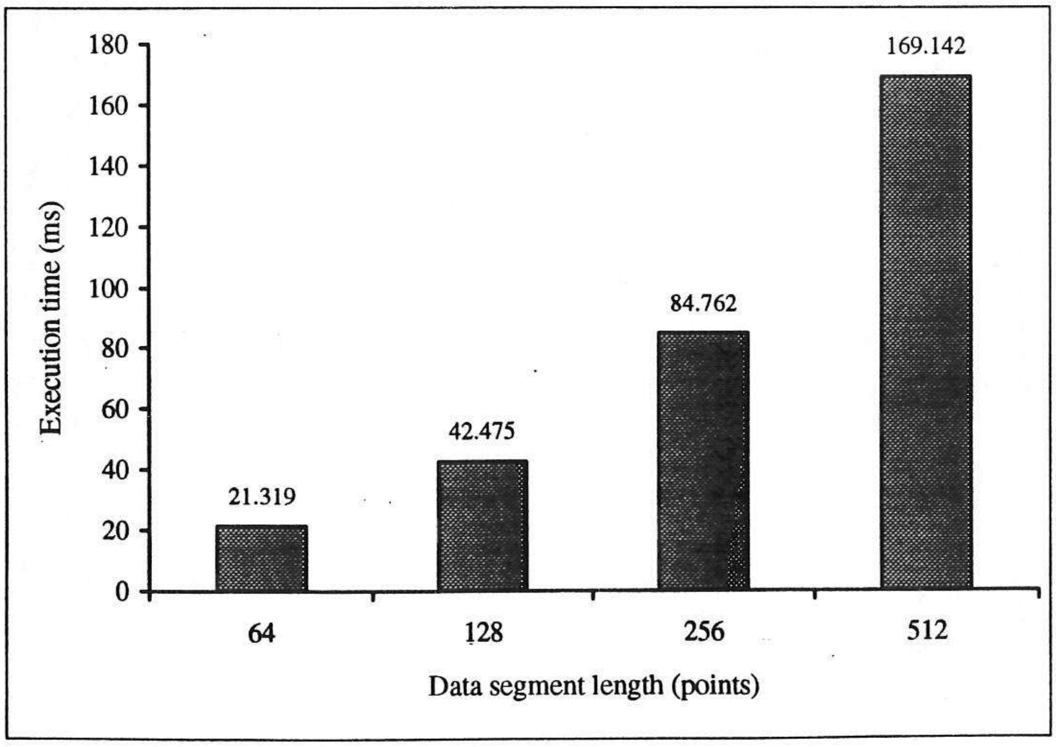


Figure 17: Execution times in estimating three data segments on three C40s (measured on the T8).

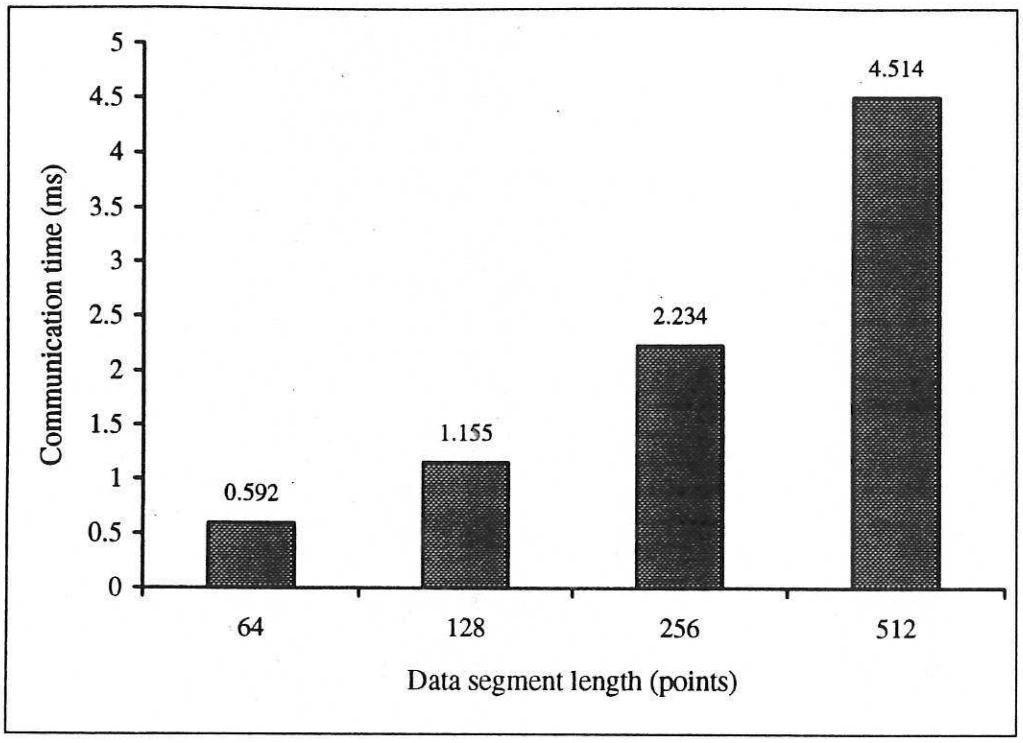


Figure 18: Communication times between the T8 and the C40.

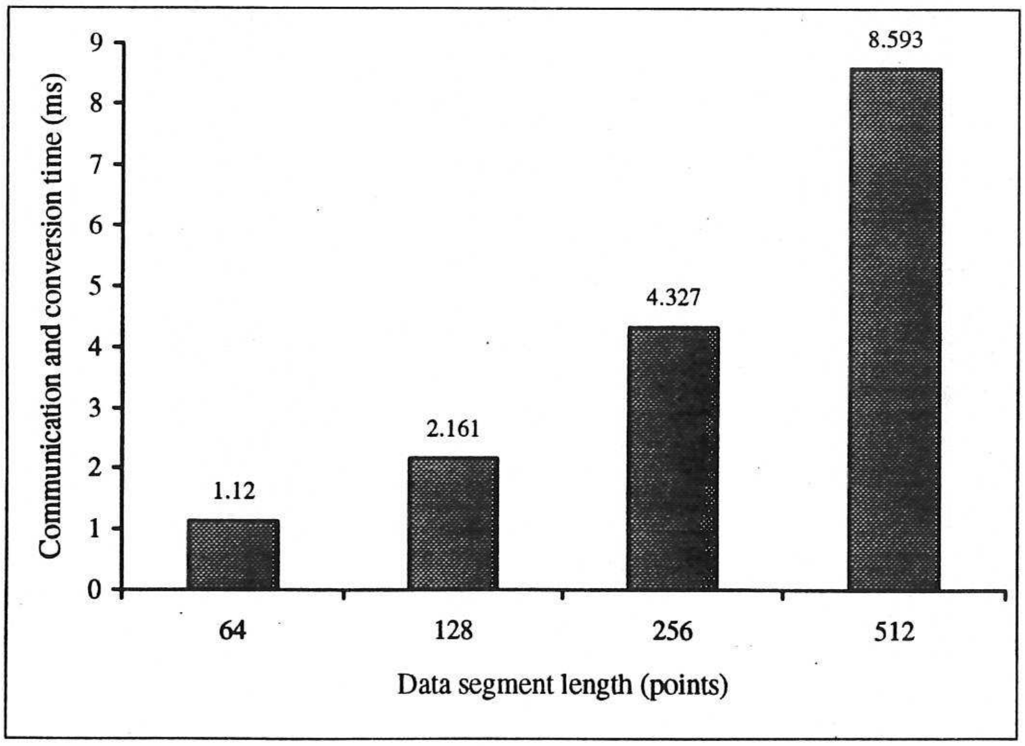


Figure 19: Communication and conversion times between the T8 and the C40.

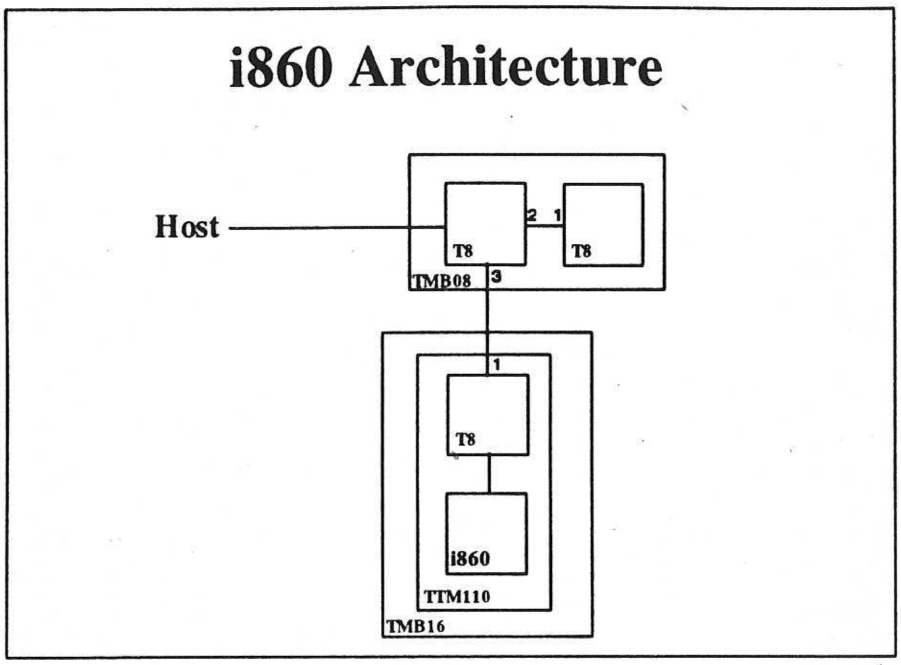


Figure 20: The i860 architecture.

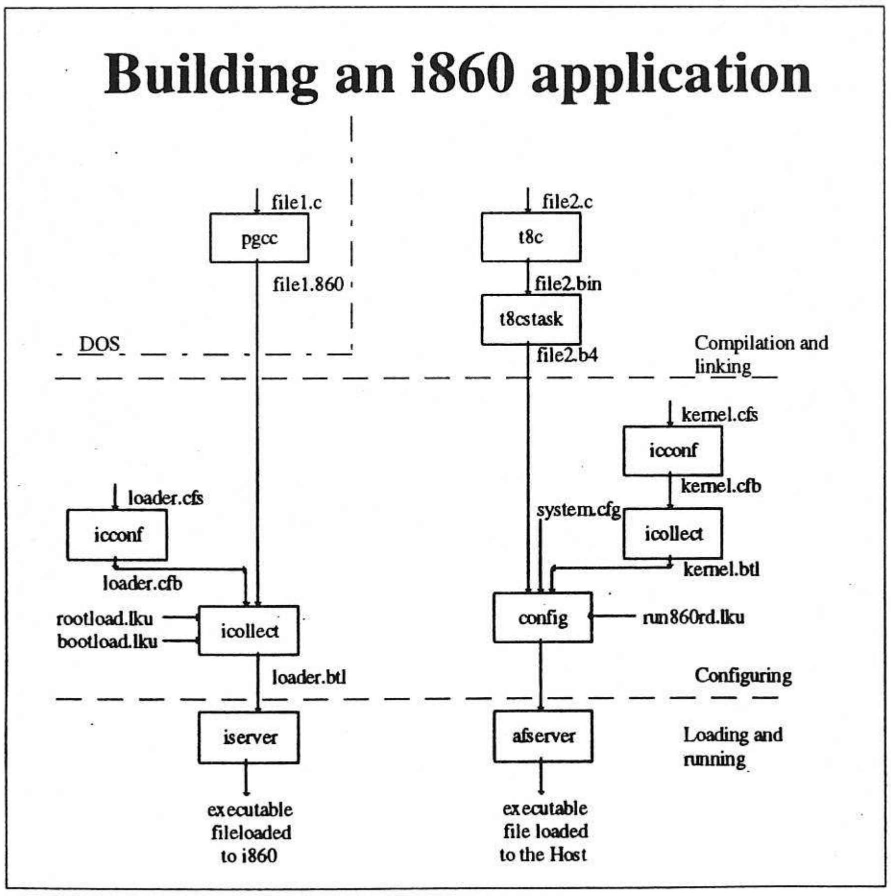


Figure 21: Process of building an i860 application (Chambers, 1995).

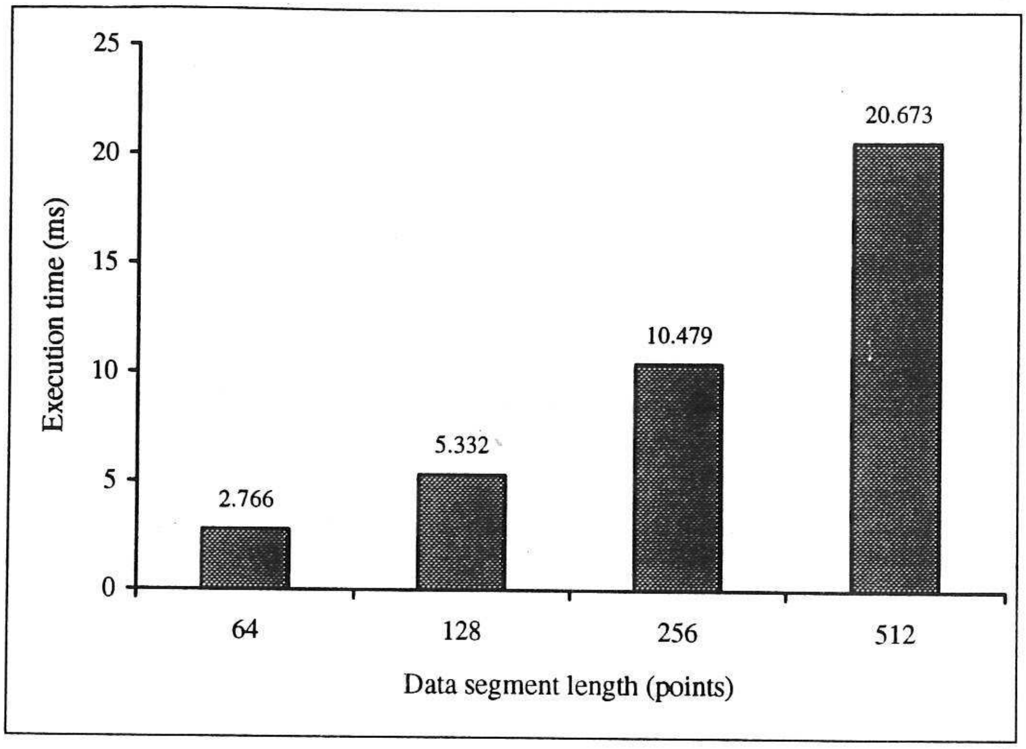


Figure 22: Execution times in implementing the MC algorithm on the i860.

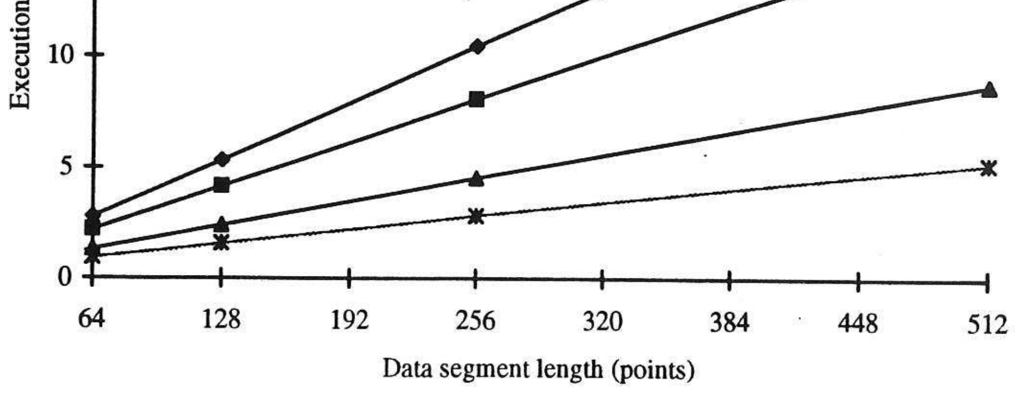
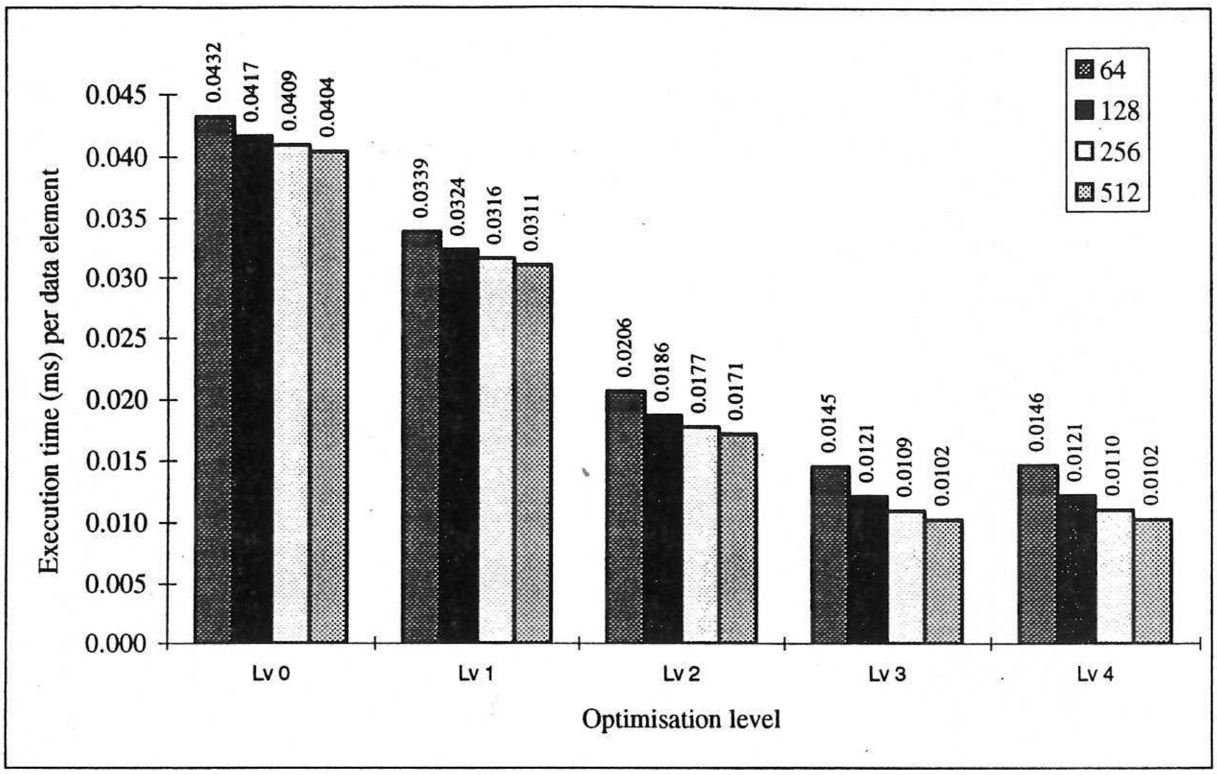
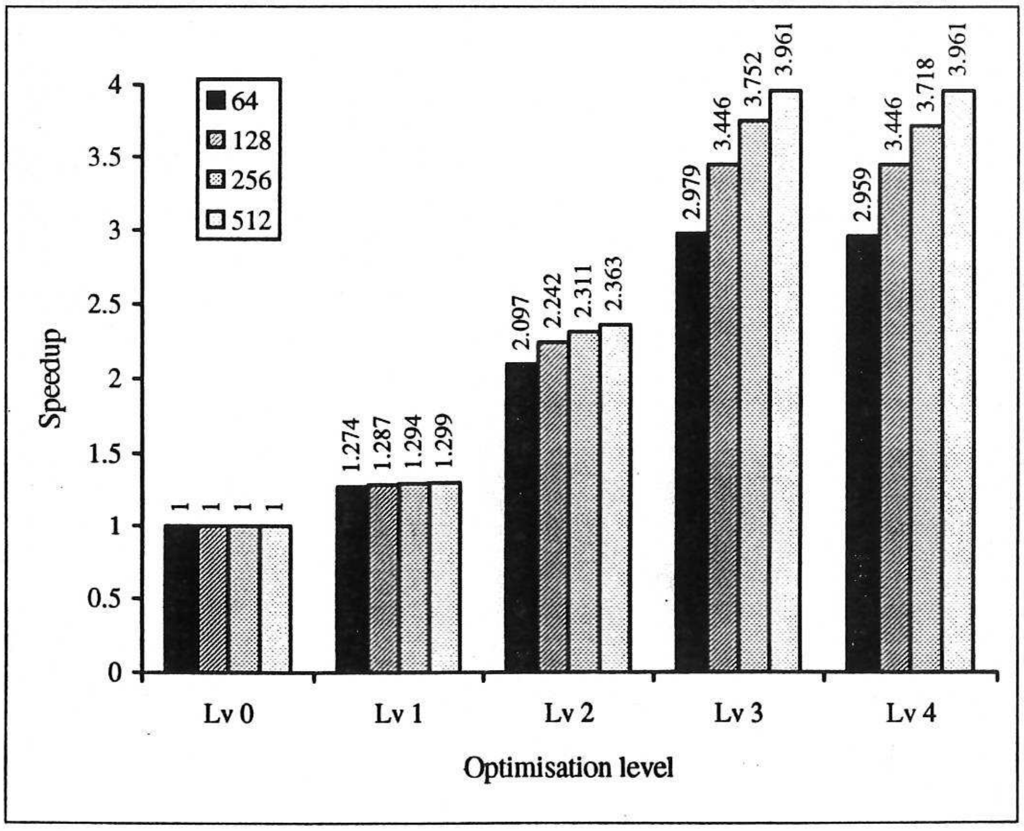


Figure 23: Effect of compiler optimisation on execution time of the i860.



(a) Implementation gradients.



(b) Speedups.

Figure 24: Effect of compiler optimisation in implementing the algorithm on the i860.

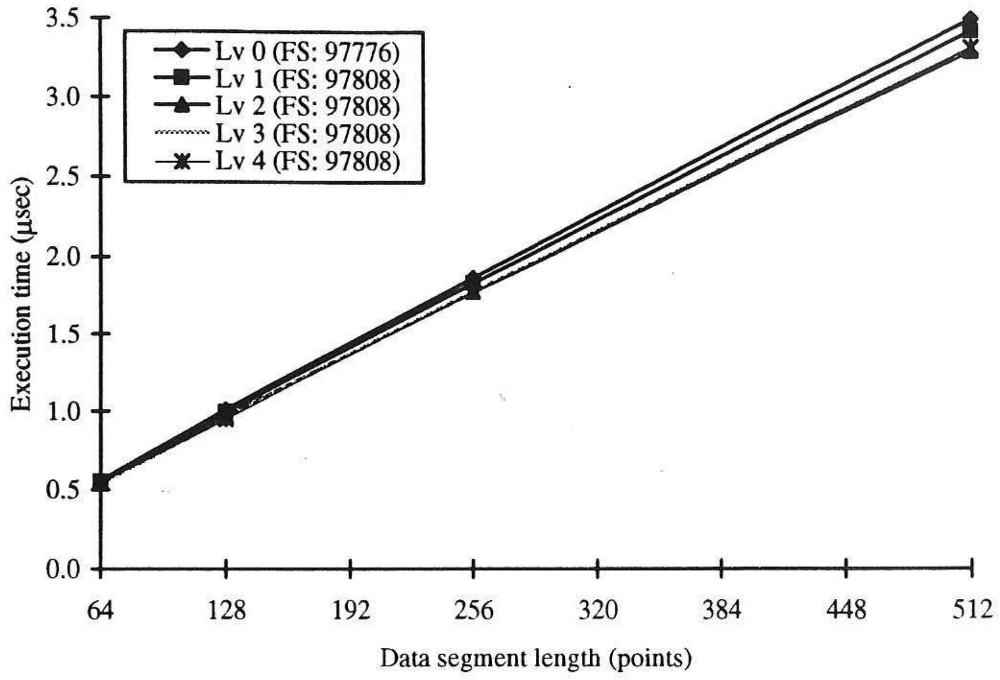


Figure 25: Communication times between the T8 and the i860.

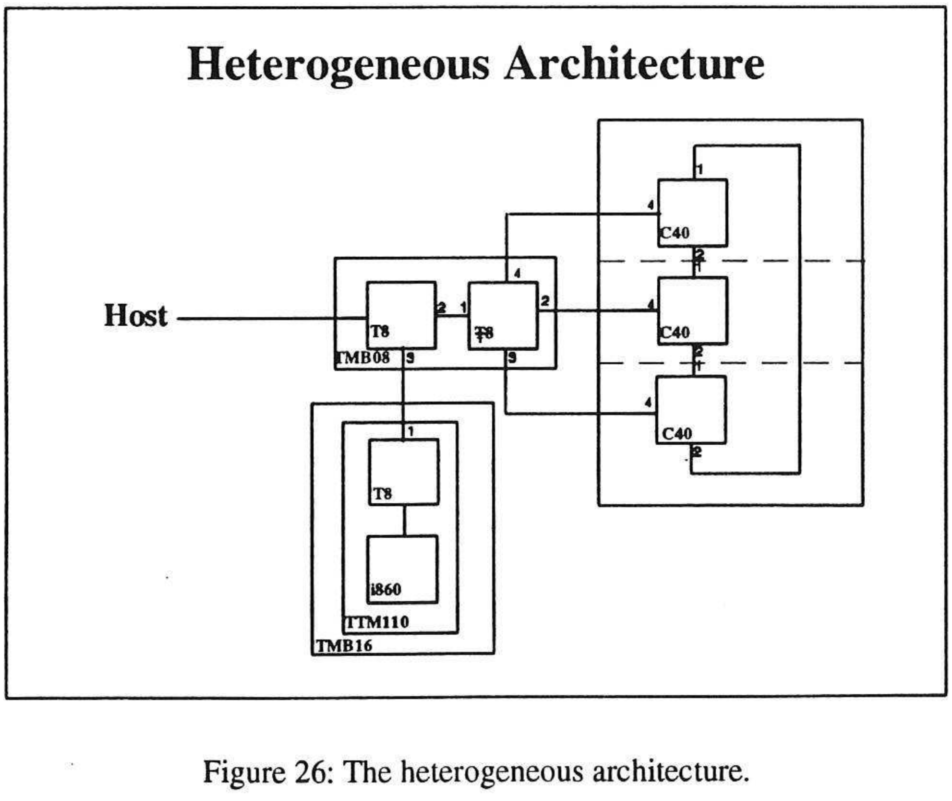


Figure 26: The heterogeneous architecture.

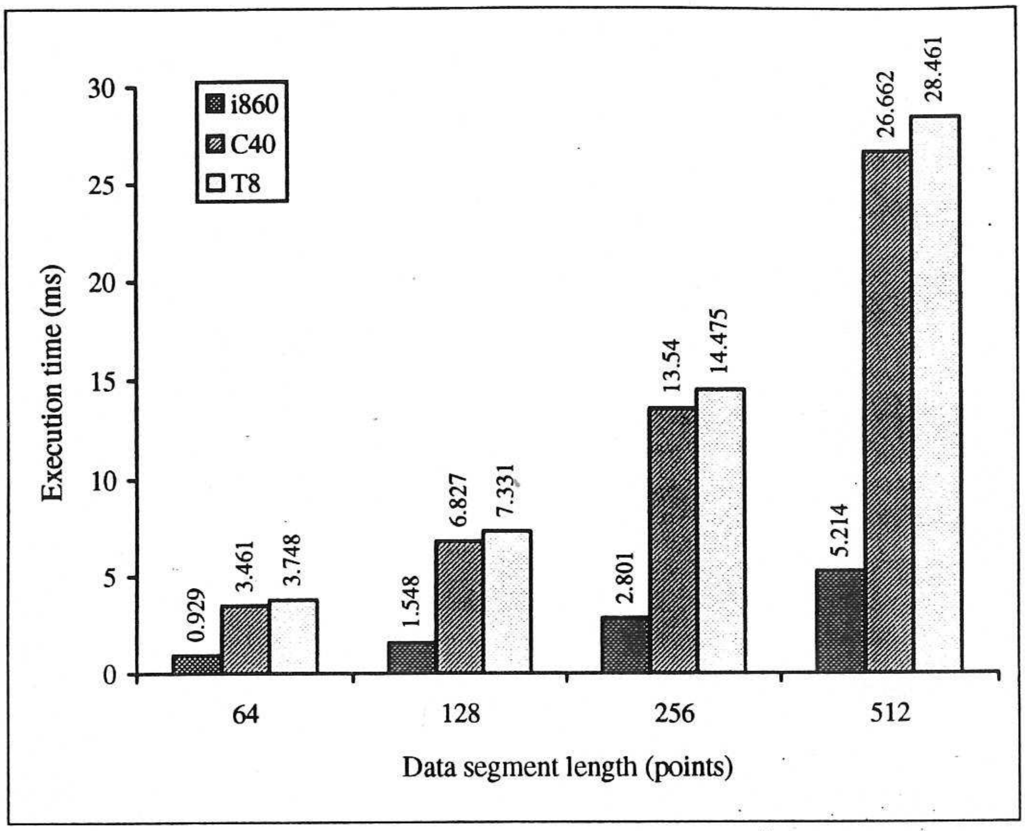


Figure 27: Execution times in implementing the algorithm on the heterogeneous architecture.

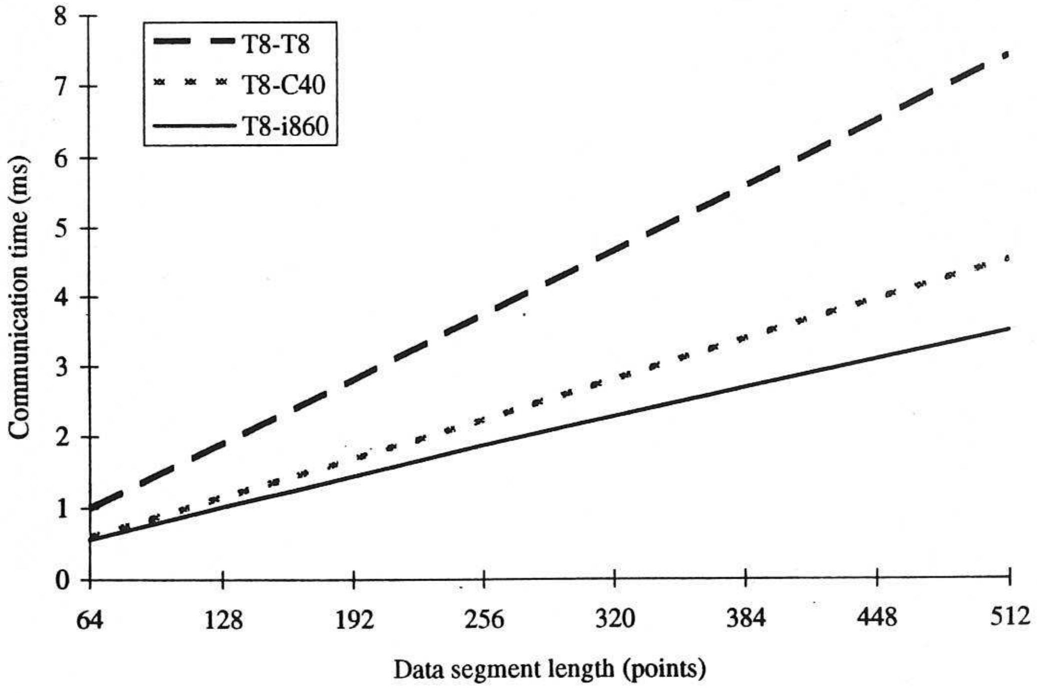


Figure 28: Comparison of communication times between the processing elements.

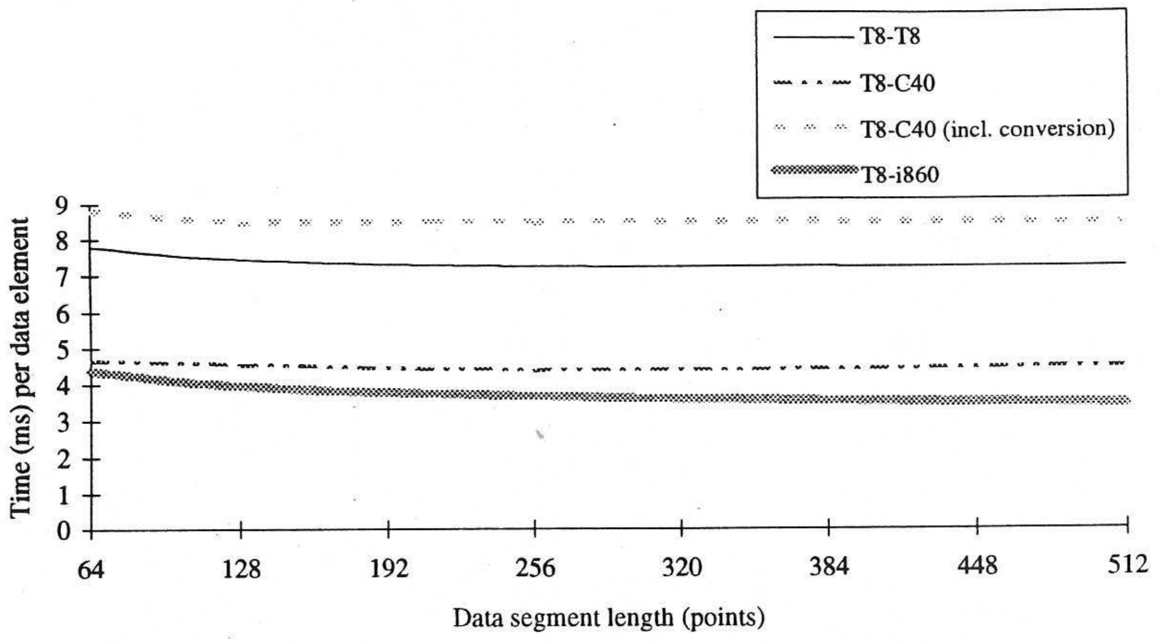


Figure 29: Gradient of communication between the processors.

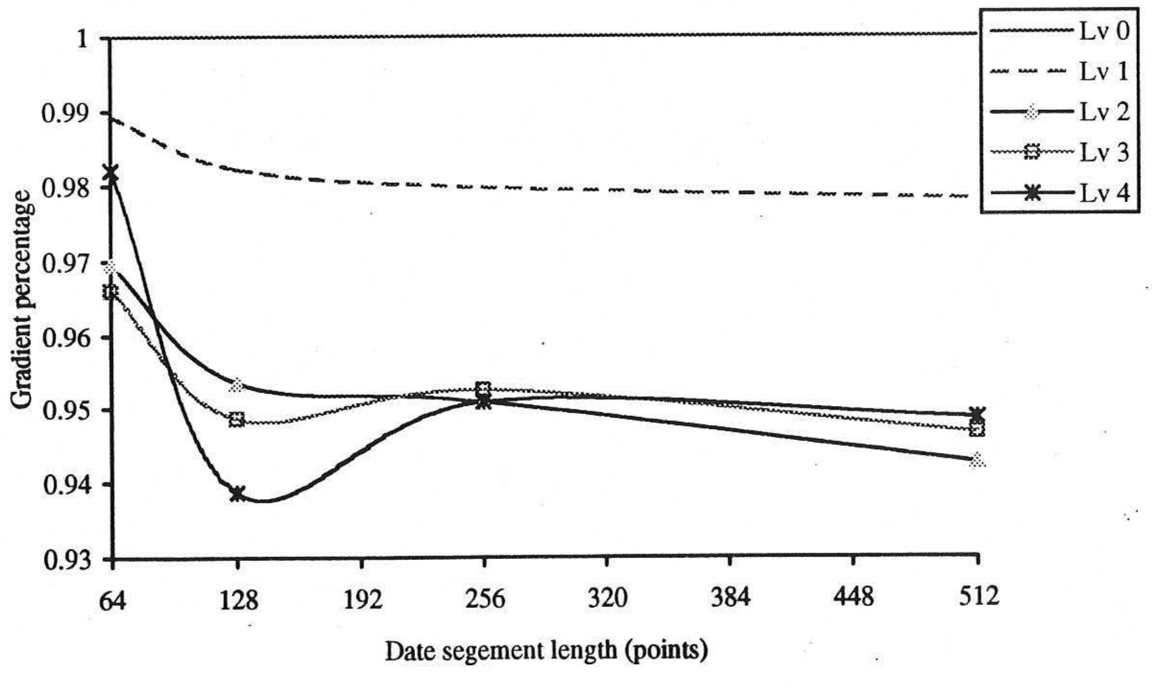


Figure 30: Effect of code optimisation on communication time between the T8 and the i860.

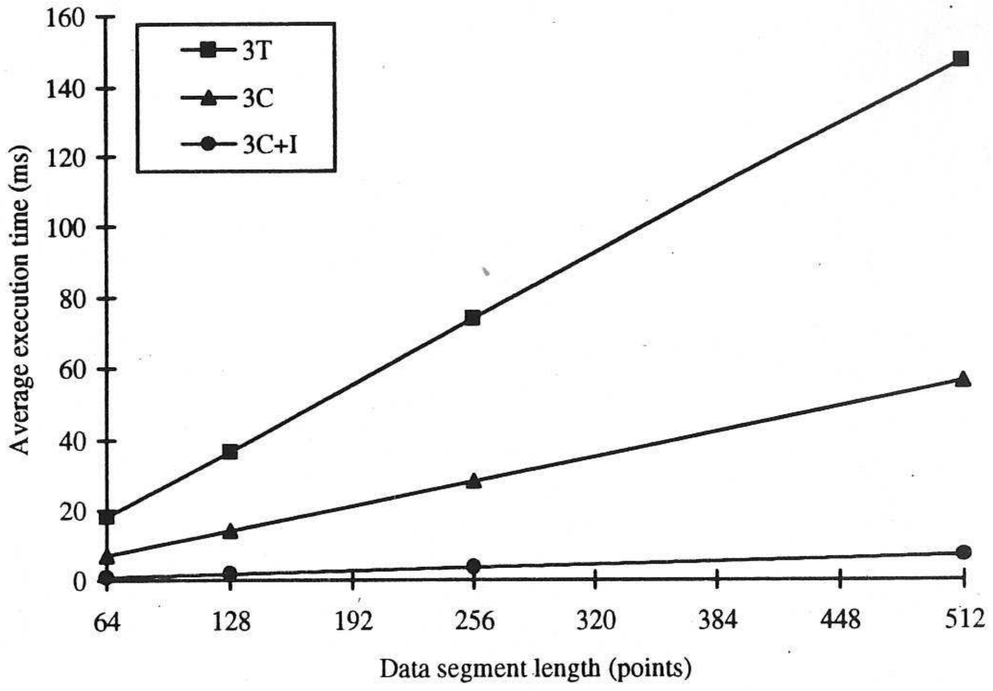


Figure 31: Execution times of the different architectures.