X

Department of
AUTOMATIC
CONTROL
and SYSTEMS
ENGINEERING
UNIVERSITY of SHEFFIELD

MAPPING AND SCHEDULING STRATEGIES FOR HETEROGENEOUS ARCHITECTURES

D. N. Ramos-Hernandez ‡ M. O. Tokhi ‡ and J. M. Bass †

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

Tel: + 44 (0)114 222 5617.
Fax: + 44 (0)114 273 1729.
E-mail: o.tokhi@sheffield.ac.uk.

† School of Electronic Engineering and Computer Systems,
University of Wales, Bangor, Dean Street, Gwynedd, LL57 1UT, UK.

Research Report No. 736

December 1998

Abstract

Extensive and computationally complex signal processing and control applications are commonly constructed from small computational blocks where the load decomposition and balance may not be easily achieved. This requires the development of mapping and scheduling strategies based on application to processor matching. In this context several application algorithms are utilised and investigated in this work within the Development Framework (DF) approach. The DF approach supports the specification, design and implementation of real-time control systems. It also contains several mapping and scheduling tools to improve the performance of systems as well as tools for code generation. To improve the performance of an application a new approach, namely the Priority-Based Genetic Algorithm (PBGA) is developed and reported in this paper. The approach is applied to several applications using parallel and distributed heterogeneous architectures and its performance verified in comparison to several previously developed strategies.

Keywords: Distributed systems, genetic algorithms, heuristic algorithms, heterogeneous architectures, mapping, parallel systems, scheduling.

CONTENTS

LIST OF TABLES AND FIGURES

1 Introduction

One of the objectives in heterogeneous computing is to decompose an application into several tasks and match each task to the machine which is best suited for its execution. In this manner, mapping and scheduling are two important issues to be considered in a heterogeneous system (Siegel *et al.*, 1996). The scheduling problem in multiprocessor systems can be stated as a set of partially ordered computational tasks onto the multiprocessor system, so that some criteria will be optimised. The scheduling problem depends heavily on the topology of the task graph representing the precedence relations among the computational tasks, the topology of the multiprocessor system, the number of parallel processors, the uniformity of the task processing time, and the performance criteria chosen (Huo *et al.*, 1994).

The multiprocessor scheduling problem is computationally complex, so that heuristic algorithms have been proposed to obtain optimal and sub-optimal solutions. Most of the existing techniques are based on list scheduling, where each task is assigned a priority either statically or dynamically. Whenever a processor becomes available a task with the highest priority is selected from the list and assigned to the processor (Ahmad and Dhodhi, 1996). Kasahara and Narita proposed a heuristic algorithm (critical path/most immediate successors first) and an optimisation/approximation algorithm (depth first/implicit heuristic search) (Kasahara and Narita, 1984). Chen et al. developed a state-space search algorithm coupled with a heuristic to solve the multiprocessor scheduling problem (Chen *et al.*, 1988). Recently, genetic algorithms (GAs) have been used for solving parallel processing problems such as scheduling, data organisation and partitioning, communication and routing. Genetic algorithms are defined as search algorithms based on the mechanics of natural selection and natural genetics (Goldberg, 1989). GAs have been applied to optimisation problems. However, GAs differ from traditional optimisation methods in the following ways: a GA uses a coding of the parameter set rather than the parameters themselves; a GA searches from a population of search nodes instead of from a single one; a GA uses an objective function instead of auxiliary knowledge. Finally, a GA utilises probabilistic transition rules. Ahmad and Dhodhi developed a technique based on the problem-space GA (PSGA) for the static scheduling of direct cyclic graphs onto homogeneous multiprocessor systems to reduce the response-time (Ahmad and

Dhodhi, 1996). Baxter et al. (1996) proposed several mapping and scheduling algorithms that attempt to minimise the cycle-time of the application algorithms for parallel heterogeneous architectures. These include a heuristic algorithm called the Modified Menasce (MM), a simple genetic algorithm (SGA), a grouping genetic algorithm (GGA) which is an augmented version of the SGA with several specialised operators to improve performance, and a GA with a mechanism to adapt the operator probabilities based on their recent performance (AGA). A priority based genetic algorithm (PBGA) approach is proposed in this paper and its performance verified in comparison to the approaches reported in (Baxter *et al.*, 1996). The objective of this comparison is to minimise the overall execution time of signal processing and control applications. The comparison is done within a parallel heterogeneous architecture. The method is further tested in a distributed heterogeneous architecture.

The organisation of the paper is as follows. Section 2 describes the application algorithms and the hardware used in this investigation. The description of the Development Framework is presented in Section 3. Section 4 presents the Simulink diagrams and the Development Framework representations of the application algorithms. Section 5 describes the proposed mapping and scheduling approach, namely the PBGA and the different approaches of the Development Framework. The results obtained with the different approaches with both parallel and distributed heterogeneous architectures are presented in Section 6, and finally, the paper is concluded in Section 7.

2 Algorithms and hardware architectures

In this section, several signal processing and control applications are described as case studies considered in this work. These include an adaptive filtering algorithm, the simulation algorithm for a cantilever beam, the control law for an autopilot and a benchmark problem. The parallel heterogeneous architecture and the distributed heterogeneous network used are also described in this section. The parallel heterogeneous architecture consists of two different processors: the Texas Instruments TMS320C40 (C40) DSP device and the INMOS T805 (T8) transputer.

The distributed heterogeneous network includes two nodes and several sensors and actuators, which communicate with each other via a CAN (controller area network) bus.

2.1 Application algorithms

The application algorithms considered in this study include the least mean square (LMS) adaptive filter algorithm, a cantilever beam simulation algorithm, the versatile auto-pilot (VAP) algorithm and a continuous benchmark problem (BEN2aSYS). Among these the LMS algorithm is of irregular nature, with uneven inner and outer loops performed and many multiply and accumulate functions. The beam simulation algorithm is of regular nature and is based on matrix multiplication and addition. The VAP algorithm is a multiple-input multiple-output control law. The BEN2aSYS is the autopilot portion of the algorithm developed at Johns Hopkins University Applied Physics Laboratory (JHU/APL) (Hawley and Stevens, 1986; Rimer *et al.*, 1990; The Mathworks, 1992). These algorithms are briefly described below.

2.1.1 The LMS algorithm

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

$$W_{j+1} = W_j + 2e_j \mu X_j \tag{1}$$

where, W_j represents the weight vector, X_j the input signal vector, μ is a constant that controls the stability, and $e_j = y_j - W_j^T X_j$ is the output error. A block diagram representation of the algorithm is shown in Figure 1.

2.1.2 The beam simulation algorithm

Consider a cantilever beam system with a force $U(x,t)$ applied at a distance x from its fixed end at time t (see Figure 2). This produces a deflection $y(x,t)$ of the beam from its stationary position at the point where the force has been applied. The dynamic equation representing this system is given as (Tokhi and Hossain, 1994)

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

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 (2) can be obtained as (Tokhi and Hossain, 1994)

$$Y_{k+1} = -Y_{k-1} - \lambda^2 S Y_k + \frac{(\Delta t)^2}{m} U(x,t) \tag{3}$$

where $\lambda^2 = (\Delta t)^2 (\Delta x)^{-4} \mu^2$, S is a pentadiagonal matrix, entries of which depend on the physical properties and boundary conditions of the beam, Y_i $(i = k+1, k, k-1)$ represents the deflection at sections $1,\ldots,n$ of the beam at time step i, and Δt and Δx are increments along the time and the distance coordinates respectively. Equation (3) represents a discrete formulation of the dynamic behaviour of the beam in transverse motion.

2.1.3 The VAP algorithm

The VAP control law is the most complex of the two laws for approach and landing developed in the theoretical studies at Royal Aerospace Establishment, Bedford (Garcia-Nocetti and Fleming, 1992; Goddard, 1979). The control law has previously been employed on the Civil Avionics Section's BAC 1-11, using a single-processor (M68000) implementation. The VAP is a four inputs, two outputs control algorithm.

Figure 3 shows a block diagram of the VAP algorithm. The inputs are pitch rate (q), barometric height error (h_B), vertical acceleration (d^2h/dt^2) and airspeed error (u_a). The outputs are elevator rate demand (dn_d/dt) and throttle demand (v_d).

The notation in Figure 3 follows the convention that the gain between the elevator position and an aircraft state error input is denoted by **G** with the error variable as a subscript. Similarly, **A** is used to denote the gain associated with throttle position. The throttle position control law consists of a smoothing lag on the airspeed error input together with the **A** gains. As the elevator servo-system is a rate demand servo, the elevator control law demand is of derivative type. The first component of the elevator control law comprises **G** gains together with a 0.1s lag. In the second component of the law, the height error inputs from several sources are mixed in complementary filters with normal acceleration information. The signals are further transformed by the lag terms and **G** gains (Garcia-Nocetti and Fleming, 1992).

2.1.4 The BEN2aSYS

The Autopilot and Airframe system is presented in three parts, corresponding to the three phases in the design of such a system. The initial design considers a low-order and uncoupled system. In the intermediate design, the model is upgraded to a three-channel, coupled and low-frequency model, which sometimes is used for initial performance analysis. In the final phase, the model is extended to include all the high-frequency filtering found in a system in the advanced stages of the design process. For experimental purposes only the simple model is considered in this work. Figure 4 shows the autopilot and airframe pitch channels.

2.2 Hardware resources

This section describes the hardware that comprise the parallel and distributed heterogeneous architectures. These architectures are utilised for investigations throughout this research.

2.2.1 Topology of the parallel heterogeneous system

The topology of the parallel heterogeneous system used to carry out the experimental investigations in this work is shown in Figure 5. This comprises a Transtech TMB08 motherboard with 10 TRAMs possessing two INMOS T805 (T8) transputers. The root T8 incorporates 2 Mbytes local memory and is connected to the Host computer (SUN SPARC). The root T8 is connected, via its link 2 to a sub-network of three Texas Instruments TMS320C40 (C40) DSP devices each with 3 Mbytes DRAM and 1 Mbyte SRAM. Further descriptions of the processors are given in the following paragraphs.

The T8 is a 32 bit CMOS microcomputer with a 64 bit floating point unit. It is a general-purpose medium-grained parallel processor with 25 MHz clock speed, yielding up to 20 MIPS performance and is capable of 4.3 MFLOPS. This has 4 Kbytes on-chip RAM for high speed processing, a configurable memory interface and four standard INMOS communication links. The links operate at speeds of 20 Mbits/sec, achieving data rates of up to 1.7 Mbytes/sec uni-directionally or 2.3 Mbytes bi-directionally. The T805 can directly access a linear address space of 4 Gbytes and a configurable memory controller provides all timing, control and DRAM refresh signals for a wide variety of mixed memory systems (Inmos, 1989).

The C40 is a 32-bit DSP device with 40 MHz clock speed, 8 Kbytes on-chip RAM, and 512 bytes on-chip instruction cache. It is capable of 275 MOPS and 40 MFLOPS. This processor has six communication ports for high speed inter-processor communication (20-Mbytes/sec asynchronous transfer rate at each port for maximum data throughput), six-channel DMA coprocessor for concurrent I/O and CPU operation, two identical external data and address buses supporting shared memory systems and high data rate (single-cycle transfers), and on-chip program cache and dual-access/single-cycle RAM for increased memory access performance and separate internal program data (Texas Instruments, 1991).

2.2.2 Topology of the distributed heterogeneous system

The distributed heterogeneous system utilised in this work consists of two nodes (PC machines, 486 and 386), and several sensor and actuator devices connected via CAN (Controller Area Network) bus (see Figure 6). Each PC contains a CAN application controller 2 (CAN-AC2), which exchanges information with the PC through a dual ported RAM (DPRAM). The CAN-AC2 contains its own processor (NEC V25+) and two CAN connections that can be operated in parallel (Softing, 1996). The CAN controller controls all the mechanisms specified in the CAN protocol. The sensors and actuators are autonomous devices connected to the nodes via CAN bus.

CAN Bus architecture:

The CAN bus has several advantages to develop real-time distributed systems, such as acceptable speed, high noise immunity, low cost per node, relative degree of determinism and high reliability (Rodd *et al.*, 1997). CAN is a serial communication protocol (ISO/IS-11898, 1993). This was developed by Robert Bosch GmBH for use in time-critical applications in the automotive industry. The characteristics of CAN include high transmission rate, flexibility, high data integrity, prioritised access control, data consistency and error detection function (Yun *et al.*, 1997).

The CAN bus has a multi-master priority-based bus access, with non-destructive contention-based arbitration. An identifier (ID) is given to each message in order to identify the message and to control the bus access. Thus, the identifiers of the packets are not associated with the directions of the stations, but to the data itself. A station wishing to transmit sends data and ID together, and other stations receive or ignore the message depending on the ID. Therefore, the ID classifies the message and is used to define the priority of each message. Each station constantly monitors the bus to determine whether any station is using the network. The message with the lowest ID value wins the right to use the network while others stop their transmission immediately. Since this identifier-based bitwise arbitration does not lose any information and time, it is called non-destructive bus access. So, the network is utilised efficiently even under heavy network load. CAN supports data rates ranging from 5 Kbps to 1 Mbps, with either linear bus or star topology.

3 The Development Framework

The Development Framework is an advanced CACSD environment to support the specification, design and implementation of real-time control systems (Bass *et al.*, 1994). Three phases are identified into the Development Framework: the *specification phase*, in which the designer specifies, analyses and simulates the application, the *software design phase*, which enables to analyse and refine the system under development, and the *implementation* *phase*, which allows to generate source code of the system. Figure 7 illustrates these three phases.

In the specification phase, the Simulink toolbox (The Mathworks, 1992) is used to support the modelling and simulation of the application and is also used to provide a well documented mechanism of specification of control systems. Once the control engineering representation is obtained using Simulink, the Development Framework includes tools that automatically translate this representation into a software engineering domain. In this phase, the Development Framework produces a data-flow diagram (DFD) in the file format of the Software through Pictures (StP) CASE environment. Thus, the DFD is equivalent to the Simulink diagram and each functional block (gains, transfer function, for example) is converted to a process symbol within the Development Framework. This complete description of the application system allows the analysis, implementation and documentation of the proposed design. Finally, the Development Framework tools translate the software engineering representation into source code that can be compiled into executable code for a network of processors. To obtain the source code of the application, firstly, it is necessary to apply the mapping tool, which appends an annotation to the DFD to indicate the task to processor mapping and determine the task execution order. From this information the code generator creates the source code for the implementation which is then compiled.

The Development Framework incorporates several code generators which exist for the different forms of hardware architectures, including single processor UNIX workstations and also heterogeneous architectures (consisting of mixed networks of transputers, C40 DSPs, and Intel i860). However, the Development Framework as well as other environments; for example, Ptolemy (Pino *et al.*, 1994), Comdisco's DPC (Powell *et al.*, 1992) and CADIS's Descartes (Ritz *et al.*, 1992) do not incorporate a bus-based architecture. A bus-based architecture has several advantages in the development of real-time distributed systems, such as acceptable speed and high reliability. In order to incorporate an architecture based on CAN bus into the Development Framework, a tool for automatic code generation was developed. Thus, C code is generated automatically and the application can be implemented in a distributed heterogeneous environment based on CAN bus.

4 Simulink diagrams and the Development Framework representations of the applications

In previous investigations the Simulink block diagrams of the VAP and BEN2aSYS were developed (Baxter *et al.*, 1996; Garcia-Nocetti and Fleming, 1992; The Mathworks, 1992). Thus, it was only necessary to develop further the block diagrams for the LMS and the beam simulation algorithms in this work. Figures 8 and 9 show the Simulink diagram of the LMS and the beam simulation algorithms respectively. It is noted in Figures 8 and 9 that a small number of control blocks including unit delays, multiplexes, de-multiplexes, S-Functions, gains, sums and constants are used to construct control and digital signal processing applications, such as the LMS and beam simulation algorithms.

Since the mapping and scheduling problem depends heavily on the topology of the task graph representing the precedence relations, It is necessary to convert the block diagram representations of the applications into task graph form. To do this the Development Framework is used, which produces the DFD for each application. Figure 10 shows the DFD for the LMS algorithm. For this algorithm a DFD with eleven processes was generated. As noted each process corresponds to the block diagram in the Simulink representation. In the DFD of Figure 10 a duplicating de-multiplexer (Demux 1) process is added where lines within the block diagrams split.

Figure 11 shows the DFD generated for the beam simulation algorithm. In this diagram two de-multiplexer processes are added in the translation. Thus, the DFD possesses eight processes with their corresponding data communication flows.

Figure 12 shows the DFD of the VAP control law. This application contains 40 processes and significant cross-coupling terms. Finally, the DFD of the converted BEN2aSYS system is shown in Figure 13. The diagram only shows the top level of the BEN2aSYS translation. The number of processes generated is 47 in all its levels. Thus, this is the most complex of the applications considered in this work.

5 Mapping and scheduling approaches

The mapping and scheduling problems are simple in concept. However, several factors need to be considered. For example, a single task must be mapped to a single node with the aim of minimising its execution time, but the upper and lower bounds of the execution time are difficult to obtain. In heterogeneous architectures execution time of a task will vary depending upon which processor type it is mapped on. Thus, each task should be matched to the processor that achieves minimum execution time. Inter-processor communication overhead, which often dominates the computing time, is important. Furthermore, it is important to note that a DFD has precedence, so that, tasks cannot be executed until their predecessors have been completed. Finally, it is important to consider the interference cost that occurs when the combination of several tasks on a processor affect their expected execution times (Baxter *et al.*, 1996). The remainder of this section focuses on first describing the MM and SGA approaches previously proposed (Baxter *et al.*, 1996) and then presenting the PBGA approach. The MM approach is considered for reasons of comparison of the GA algorithms with a heuristic technique.

5.1 Modified Menasce approach

The MM algorithm is a simple heuristic that constructs a mapping in a single pass (Baxter *et al.*, 1996). This algorithm is based on task execution times for each processor, communication costs and the precedence of tasks. The algorithm determines the mapping of the tasks and the order in which they should be executed.

5.2 Simple GA

The SGA is a basic three operator approach (Baxter *et al.*, 1996). This algorithm uses a simple encoding strategy, where the chromosome length is equal to the number of tasks and the value of each element of the chromosome corresponds to the processor to which that task is mapped to. The number of individuals in the population is currently set to twice the number of tasks in the application. Its objective function is based on the precedence tasks graph of the application. The fitness values are derived from the objective values by linear ranking, with a selection pressure of 2, prior to selection. The selection strategy is stochastic universal sampling with a generation gap of 1. The crossover is reduced to surrogate shuffle crossover with a survival rate of 0.3. Finally, the mutation survival rate is set to 0.02.

5.3 Priority-based GA mapping algorithm

The PBGA is also a basic three operator approach. Each individual (chromosome) of the population, as in the SGA approach, has a length equal to the number of tasks and the value of each element represents the processor to which it is mapped, as shown in Figure 14.

The fitness value of each chromosome is calculated according to the fitness function (objective function). The objective function of the PBGA is based on a priority assigned to each task. In a real-time system this will ensure minimum execution time with the allocation of processes. The priorities are obtained according to the sequence of the tasks in the data-flow diagram. A priority equal to one (highest priority) is assigned to all these tasks with zero precedence tasks (input tasks). Then, the priority is increased by one for each successor. This procedure is repeated until either the successor is an output or its priority has been assigned. If the task's priority has been assigned already, this means that a loop has been found and the task needs to keep its priority. After this procedure a *temporal priority list* is created. Thus, an input task can have several temporal priority lists, from among which the one with the highest priority for each task, is chosen as the *priority list*. Similarly, the application can have several priority lists, from which the *final priority list*, is constructed and passed as parameter to the objective function. Figure 15 shows a simple example illustrating the process of obtaining the final priority list.

The execution times for each task in different processors, the inter-processor communication time, the precedence of the tasks in the DFD and the data communication are also passed as parameters to the objective function. In the objective function, the tasks with highest priority (one) are scheduled first on the processor where they were mapped. This process is repeated until all tasks are scheduled. In the PBGA approach the cost of inter-processor communication is also considered.

The fitness values are derived from the objective values by linear ranking. A stochastic universal sampling selection strategy is used with a generation gap of 1. A simple one-point crossover operator is used with a crossover rate of $P_{xover} = 0.6$. Finally, a mutation operator is used with a mutation rate of $Pm = 0.7\,/\,(\text{number of tasks})$ to indicate the percentage of the total number of genes in the population which are mutated in each generation.

6 Experimentation and results

This section is divided into two parts; experiments and results using the parallel heterogeneous architecture (T8 and three C40s) and experiments and results carried out with the distributed heterogeneous network based on the CAN bus. In order to obtain the mapping and scheduling of the different applications, it is required to know a-priori the execution times for each task on all the processors. Thus, the C code generated for each application using the Development Framework is compiled and executed on each processor in both architectures. In this manner, the computing time for each task is obtained. Moreover, it is necessary to know a-priori the precedence relations of the tasks and their priorities. To obtain this information two C programs were developed. The programs were implemented in ANSI C using several functions of the Framework Information Interchange (FII). The FII is a data storage, creation and access mechanism, which is used for the Development Framework's tools (Browne *et al.*, 1997). The procedure to obtain the priorities was explained in Section 5.3. The C program developed to obtain the precedence relations of each task consists of the following two steps:

1) input and output tasks connected to external devices are eliminated, and a task list is created,
2) from the task list, the precedence relations of each task are obtained. A precedence relation is an input task.

It is also required to know a-priori the communication times between processors. In case of the parallel heterogeneous architecture inter-processor communication is point-to point message passing and hence the communication times are straightforward to obtain.

In the distributed heterogeneous system based on the CAN bus, the end-to-end communication delay for a message is not simple to obtain, considering the time taken to access and transmit data on a communication link, the time taken to deliver the message to the destination processor, and the time taken to assemble and queue the message at the source processor (Tindell *et al.*, 1994). The times involved in the overall communication process are: (1) real to integer conversion (t_{RI}), (2) CAN bus time (t_{CAN}) and (3) integer to real conversion (t_{IR}). Figure 16 illustrates these three times. The time taken to deliver or access a message in the buffer and to queue the message are not considered due to the poor precision of the PC nodes. To obtain the times t_{RI} and t_{IR}, the applications were implemented on the distributed architecture and an average of these times was computed ($t_{RI} + t_{IR} = 307\ \mu sec$). For the bus time t_{CAN} a frame of 130 bits was considered. This resulted in $t_{CAN} = 130\ \mu\,\mathrm{sec}$.

The MM, SGA and PBGA approaches were executed on a Sun workstation and for the GA routines the GA Toolbox of MATLAB 4.2b was used.

6.1 Experiments and results with the parallel heterogeneous architecture

The execution times for the VAP algorithm using the MM and SGA approaches were previously obtained for the case of four processors (Baxter *et al.*, 1996). In this paper, the execution times for two and three processors are also obtained and presented for reasons of comparison with the execution times of the PBGA approach. Thus, the VAP was mapped using the MM, SGA and PBGA accordingly. The population size used was 68 (34 tasks x 2, input and output tasks are not considered) and the number of generations considered was 20. The results obtained with the PBGA are shown in Table 1. The execution times for the MM and SGA are also shown in this table.

Table 1 confirms that the GA based approaches outperform the MM approach. For the case of the GA approaches, it can be seen that a significant reduction in execution time is obtained with the PBGA approach than with the SGA. This indicates that improving the objective function results in a reduction in the execution time. Table 2 shows the performance in terms of execution time speedup achieved with the PBGA over the MM and SGA approaches. It is noted, that the PBGA outperforms the MM approach by 2.559 and the SGA approach by 2.004 for the case of two processors.

Figure 17 shows the distribution of the tasks on two, three and four processors using the PBGA and considering inter-processor communication (pctt) for each case. The distribution of the tasks in time is shown along the horizontal axis, as the task starts and finishes. The number of tasks is indicated along the vertical axis. It can be observed that for the case of two and three processors the inter-processor communication time is acceptable, as it takes less than half of the total execution time of the application. However, in the case of four processors this time increases, due to heavy communication between processors when the number of processors increases.

Figure 18 shows the objective values reached in each generation after executing the PBGA approach for the case of two, three and four processors. It is noted that an acceptable objective value is obtained with a small number of generations.

For the case of the LMS, beam simulation and BEN2aSYS applications, the MM and SGA approaches were applied to minimise their overall execution time. However, using these approaches some problems arose. The applications include feedback loops in their control representations. Thus, when the MM and SGA approaches were executed for these applications no solution was reached, since these approaches do not consider cyclic data-flow diagrams. In contrast, using the PBGA approach, the mapping and scheduling of these applications was found. The PBGA solves the recurrence by assigning a priority to each task. Therefore, the PBGA results in a better method to solve the mapping and scheduling problem than the MM and SGA approaches. Figures 19 and 20 show the results obtained using the PBGA approach for all the applications with two processors (T8&C40). The scheduling and inter-processor communication times are shown in Figure 19 and the objective values are shown in Figure 20. This representation later allows comparing results with the distributed system.

It is observed in Figures 19 and 20 that communication overheads for the beam simulation and LMS are zero, meanwhile for the VAP this time takes less than half of the total execution time and for the BEN2aSYS it takes more than half the total execution time. It is also observed that for the LMS algorithm a minimum execution time is reached using the faster processor (C40).

6.2 Experiments and results with the distributed heterogeneous architecture

Since the PBGA approach can be considered as an excellent mapping and scheduling strategy. This approach was used to minimise the execution time of the applications with the distributed heterogeneous architecture.

The mapping and scheduling solution and inter-processor communications for each application are shown in Figure 21 and the objective values over 20 generations are shown in Figure 22. For the case of the beam simulation and LMS algorithms no communication overhead was noted in the overall execution time. However, in the case of the VAP and BEN2aSYS, this time is more than half of the total execution time. For the LMS algorithm, the PBGA approach resulted in a minimum execution time, allocating all the tasks to the faster processor (486).

Comparing both architectures, the execution times of the LMS and beam simulation algorithms are reduced with the parallel heterogeneous architecture. In contrast, the execution times for the large applications such as the BEN2aSYS and VAP are reduced with the distributed architecture. A similar distribution of tasks is observed in both architectures. Moreover, it is observed that the PBGA approach allocated more tasks to the faster processor.

In order to obtain an optimal value of the total execution time and to reduce the inter-processor communication, the PBGA approach was modified. This modification involves running the approach until the best optimal value for each application is found. The experiments were carried out for both the parallel and the distributed heterogeneous architectures. In the case of the parallel architecture, no reduction in the overall execution time was obtained for the beam simulation and the LMS algorithms. In contrast, for the

BEN2aSYS the total execution time was minimised from 1.226 secs to 0.0138 secs and for the VAP from 1.254 secs to 0.67 secs. Figure 23 shows the reduction in the execution time for the BEN2aSYS and for the VAP with the modified version of the PBGA approach. It is observed that for the BEN2aSYS and VAP applications, no inter-processor communication time is involved in the total execution time (pctt=0). These results were reached in the case of the BEN2aSYS after 1200 secs with 800 generations and for the VAP after 200 secs with 900 generations of running the PBGA approach.

Similarly, in case of the parallel architecture, no reduction in execution time was found for the beam simulation and LMS algorithms with the distributed heterogeneous architecture. For the case of the BEN2aSYS, the total execution time decreased from 0.2922 secs to 0.0325 secs after executing the PBGA approach for 250 secs with 320 generations. In the case of the VAP, the reduction of the total execution time was from 0.0652 to 0.0210 secs after 180 secs with 800 generations. Also, the inter-processor communication in both applications was reduced to zero. Figure 24 shows the optimal values obtained for the BEN2aSYS and VAP applications.

7 Conclusion

An investigation into different mapping and scheduling approaches for heterogeneous architectures has been presented in this paper. The MM approach is a simple strategy to map and schedule tasks. However, its performance is poor compared with the GA approaches. Thus, the MM is taking the maximum cycle-time expected for the application. The two GA approaches have the same complexity and they only differ in the objective function. Nevertheless, the PBGA outperforms the SGA approach. This means, that assigning a priority to each task of the application is a better strategy than only considering the precedence of the tasks. Moreover, with the PBGA, acceptable results can be obtained with only running this approach for few generations. Meanwhile for the SGA, it is necessary to run the algorithm for several hours to reach an optimal value. It is important to note that the inter-processor communication time has a heavy influence on the total execution time of the application and to obtain an efficient mapping and scheduling, this time must be reduced.

The results with the modified PBGA, incorporating inter-processor communication have demonstrated that for the beam simulation and the LMS algorithms no reduction in the total execution time with either the parallel or the distributed architecture was obtained. However, a significant decrease in the overall execution time for the BEN2aSYS and the VAP applications was achieved with both architectures.

The new approach has been verified with two different architectures and it has been observed that in the case of large applications the overall execution time is reduced with the distributed heterogeneous architecture, whereas for small applications the execution time is minimised with a parallel heterogeneous architecture.

8 Acknowledgements

The authors acknowledge the financial support of CONACYT-MEXICO and the UK EPSRC (Grant No. GR/K 64310).

9 References

AHMAD, I. and DHODHI, M. K. (1996). Multiprocessor scheduling in a genetic paradigm, *Parallel Computing*, **22**, (3), pp. 395-406.

BASS, J. M., BROWNE, A. R., HAJJI, M. S., MARRIOTT, D. G., CROLL, P. R. and FLEMING, P. J. (1994). Automating the development of distributed control software, *IEEE Parallel & Distributed Technology*, **2**, (4), pp. 9-19.

BAXTER, M. J., TOKHI, M. O. and FLEMING, P. J. (1996). An investigation of the heterogeneous mapping problem using genetic algorithms, *UKACC International Conference on Control'96,* Exeter, 2-5 September 1996, pp. 448-453.

BROWNE, A. R., BASS, J. M. and FLEMING, P. J. (1997). A building block approach to the temporal modeling of control software, *IFAC 4th Workshop on Algorithms and Architectures for Real-Time Control*, Algarve, April 1997, pp. 433-438.

CHEN, C. L., LEE, C. S. G. and HOU, E. S. H. (1988). Efficient scheduling algorithms for robot inverse dynamics computation on a multiprocessor system, *IEEE Transactions on Systems, Man, Cybernetics*, **18**, pp. 729-743.

[GARCIA-NOCETTI, D. F. and FLEMING, J. P. (1992). *Parallel processing in digital control,* Springer-Verlag, London.

GODDARD, K. F. (1979). *Theoretical studies of automatic control laws for a BAC 1-11 aircraft utilising the wing spoilers for direct lift control*, Technical Report 79034, Royal Aircraft Establishment.

GOLDBERG, D. E. (1989). *Genetic algorithms in search, optimization, and machine learning*, Addison-Wesley.

HAWLEY, P. A. and STEVENS, T. R. (1986). Two sets of benchmark problems for CACSD packages, *Proceedings of the Third IEEE Symposium on Computer-Aided Control System Design*, Arlington, September 1986.

HOU, E. S. H., ANSARI, N. and REN, H. (1994). A genetic algorithm for multiprocessor scheduling", *IEEE Trans. on Parallel and Distributed Systems*, **5**, (2), pp. 113-120.

INMOS. (1989). *Transputer databook,* Redwood Burn Ltd, Trowbridge.

ISO/IS 11898. (1993). Road vehicles-interchange of digital information-controller area network (CAN) for high speed communication, *ISO International Standard 11898.*

KASAHARA, H. and NARITA, S. (1984). Practical multi-processing scheduling algorithms for efficient parallel processing, *IEEE Transactions on Computers*, **33**, (11), pp. 1023-1029.

PINO, J. L., PARKS, T. M. and LEE, E. A. (1994). Automatic code generation for heterogeneous multiprocessors, *Proceedings of IEEE International Conference on Acoustics Speech and Signal Processing*, **2**, (Part II), pp. 445-448.

POWELL, D. G., LEE, E. A. and NEWMAN, W. C. (1992). Direct synthesis of optimized DSP assembly code from signal flow block diagrams, *Proceedings of IEEE ICASSP*, San Francisco, 1992, **5**, pp. 553-556.

RIMER, M., FREDERICK, D. K. and HUANG, C. Y. (1990). Solutions of the second benchmark control problem, *IEEE Control Systems Magazine*, August.

RITZ, S., PANKERT, M. and MEYR, H. (1992). High level software synthesis for signal processing systems, *Proceedings of IEEE International Conference on Application Specific Array Processors*, Berkeley, 1992, pp. 679-693.

RODD, M. G., DIMYATI, K. and MOTUS, L. (1997). The design and analysis of low-cost real-time fieldbus systems, *IFAC Distributed Computer Control Systems*, Seoul, 1997, pp. 1-9.

SIEGEL, H. J., ANTONIO, J. K., METZGER, R. C., TAN, M. and LI, Y. A. (1996). Heterogeneous computing. In Zomaya, A. Y., editor, *Parallel & Distributed Computing Handbook*, McGraw-Hill.

THE MATHWORKS. (1992). *SIMULINK: Dynamic system simulation software, user's guide*, The MathWorks Inc., pp.2.66-2.90.

SOFTING. (1996). *Documentation for the CAN application controller* 2, Version 2.00, Rev. 03, Softing GmbH, Munchen.

TEXAS INSTRUMENTS. (1991). *TMS320C4x user's guide*, Texas Instruments.

TINDELL, K., BURNS, A. and WELLINGS, A. (1994). *Analysis of hard real-time communications*, Technical report (27/06/94), Department of Computer Science, University of York, England.

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**, (14), pp. 263 277.

WIDROW, B., GLOVER, J. R., McCOOL, J. M., KAUNITZ, J., WILLIAMS, C. S., HEARN, R. H., ZEIDLER, J. R., DONG, E. and GOODLIN, R. C. (1975). Adaptive noise cancelling: principles and applications, *Proceedings of IEEE*, (63), pp. 1692-1696.

YUN, J-A., NAM, S-W. and LEE, S. (1997). Evaluation of network protocol for automotive data communication, *IFAC Distributed Computer Control Systems*, Seoul, 1997, pp. 73-78.

Table 1: VAP execution times;
G: Number of generations, P: Population size, GGPA: Generation gap, Pm: Mutation survival rate.

Number of processors	**MM** Time (sec)	**SGA** G=20, P=68 GGPA=1, Pm=0.02 Time (sec)	**PBGA** G=20, P=68 GGPA=1, Pm=0.02 Time (sec)
2	3.2084	2.5122	1.2537
3	4.1574	2.7973	1.7156
4	4.6145	2.8819	1.8084

Table 2: Execution time speedup with the PBGA over MM and SGA.

Number of processors	**MM**	**SGA**
2	2.559	2.004
3	2.423	1.631
4	2.552	1.594

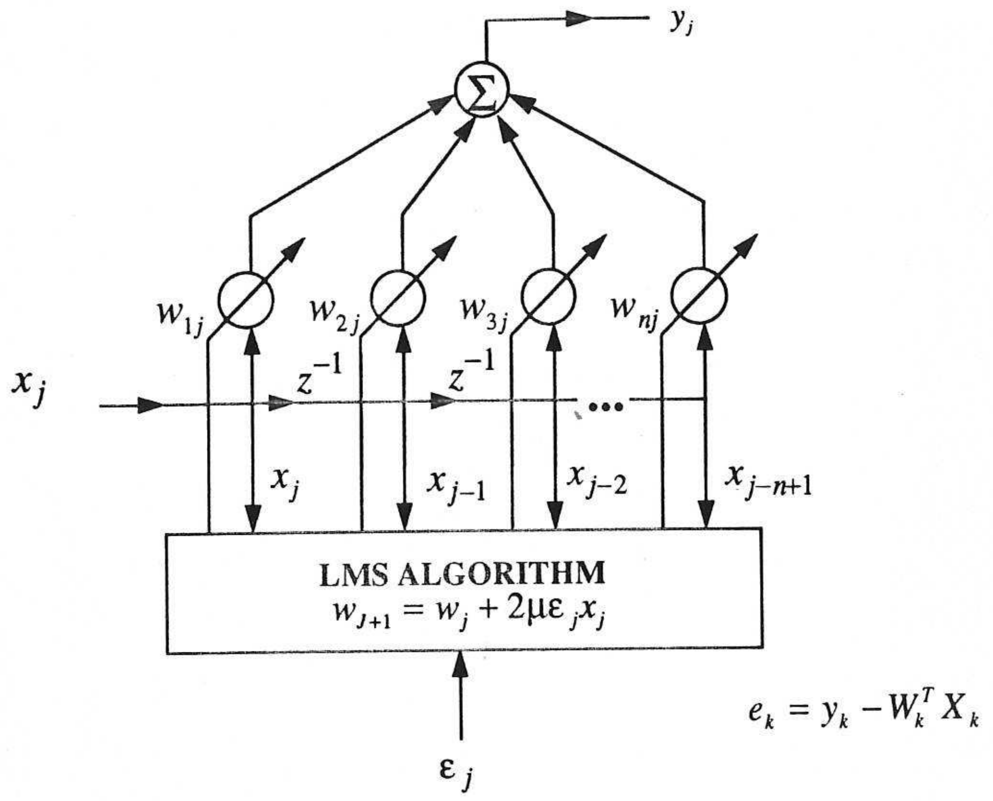


Figure 1: Block diagram of the LMS adaptive filter algorithm.

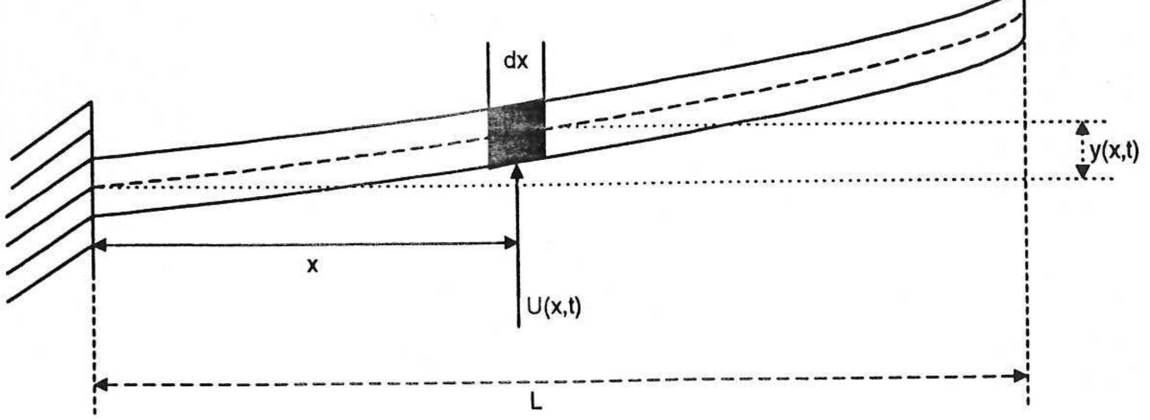


Figure 2: The cantilever beam in flexure.

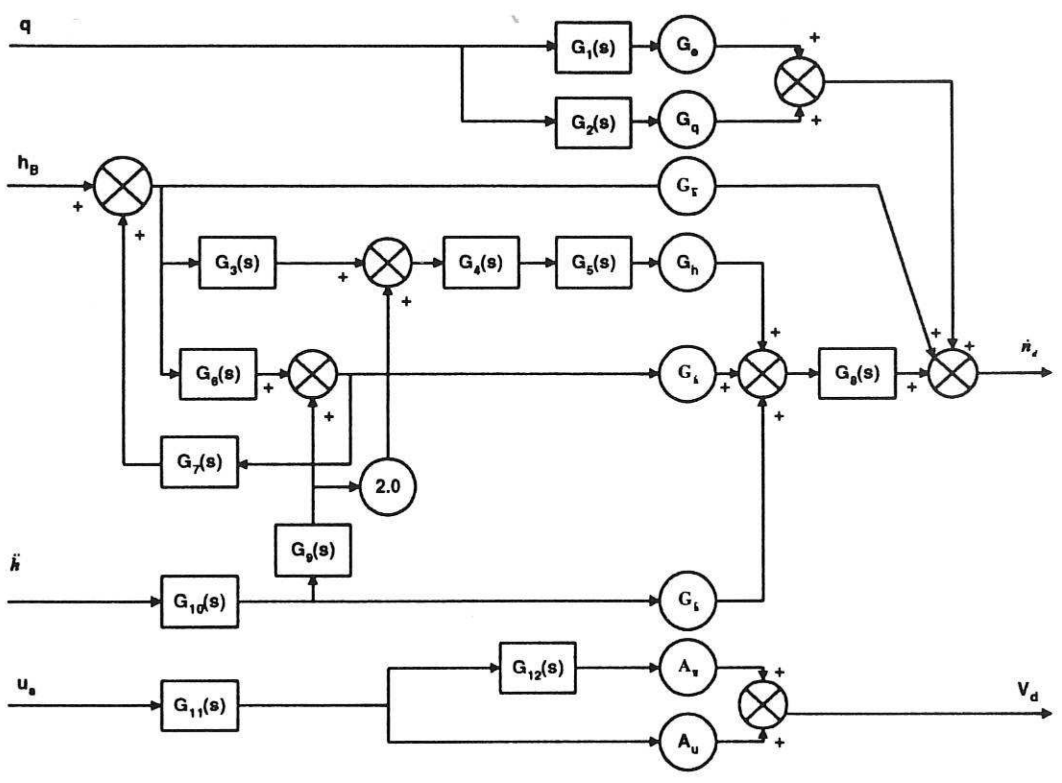
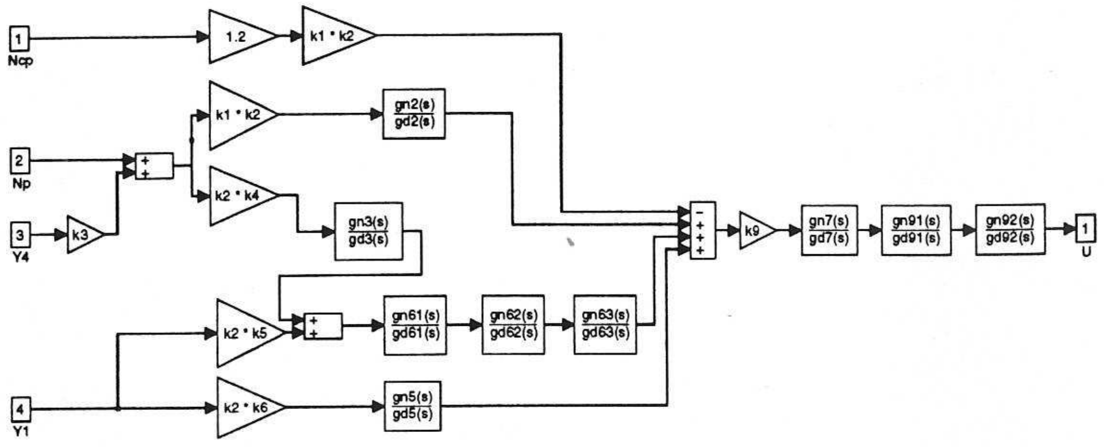
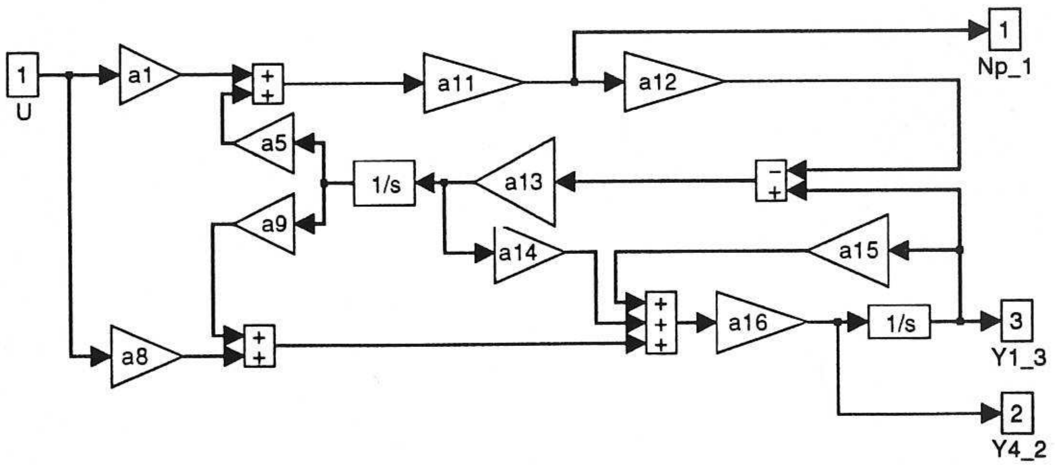


Figure 3: Block diagram of the VAP algorithm.



(a) The pitch autopilot channel.



(b) The pitch aircraft channel.

Figure 4: Block diagram of the BEN2aSYS.

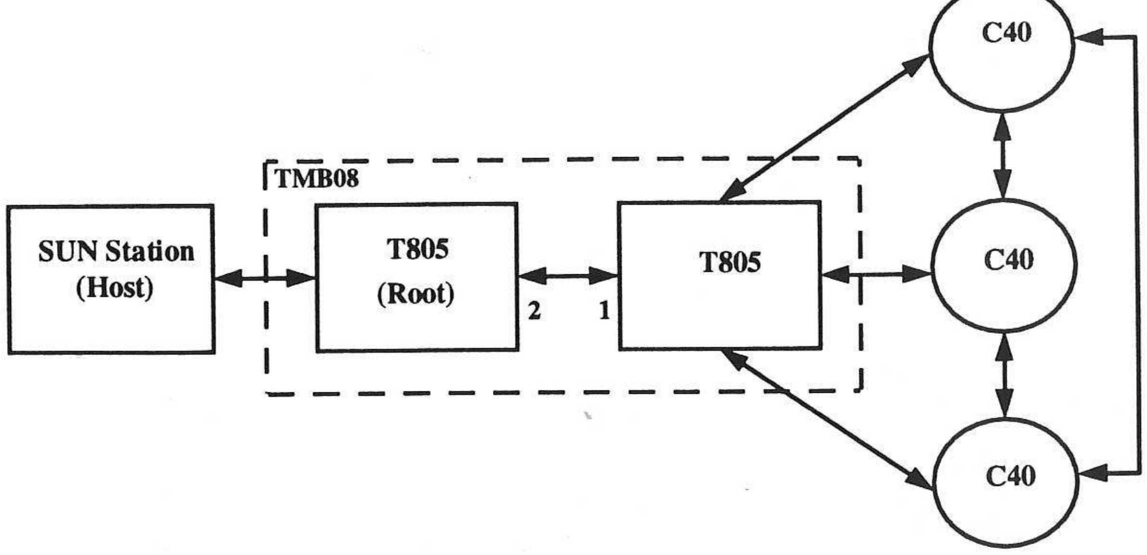


Figure 5: Topology of the parallel heterogeneous system.

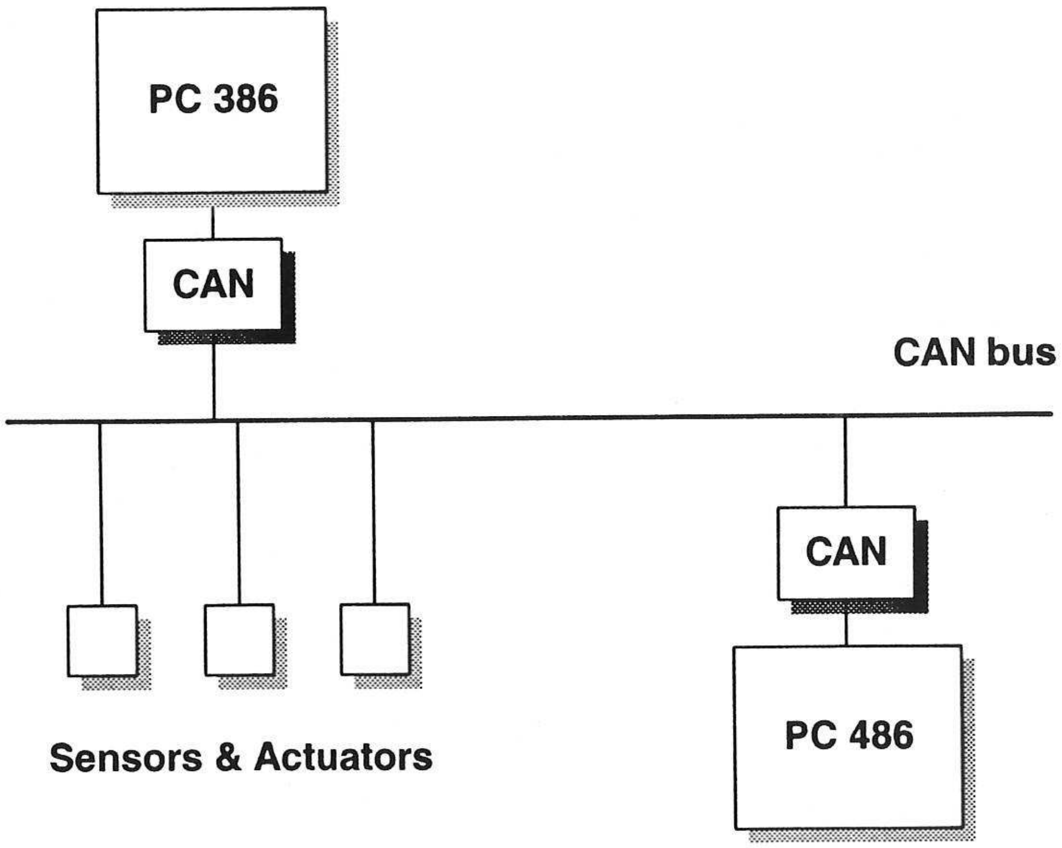


Figure 6: Architecture of the distributed heterogeneous system.

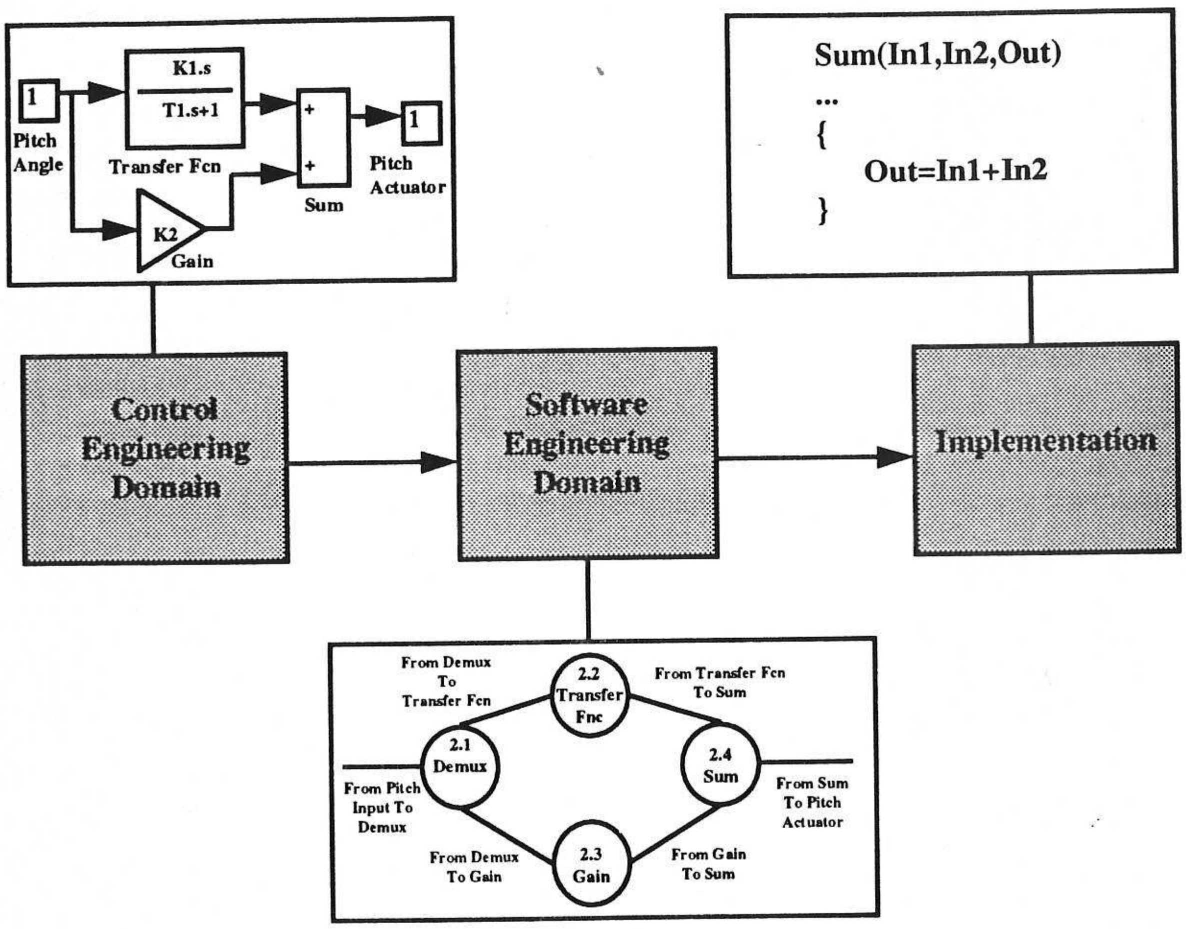


Figure 7: Overview of the Development Framework.

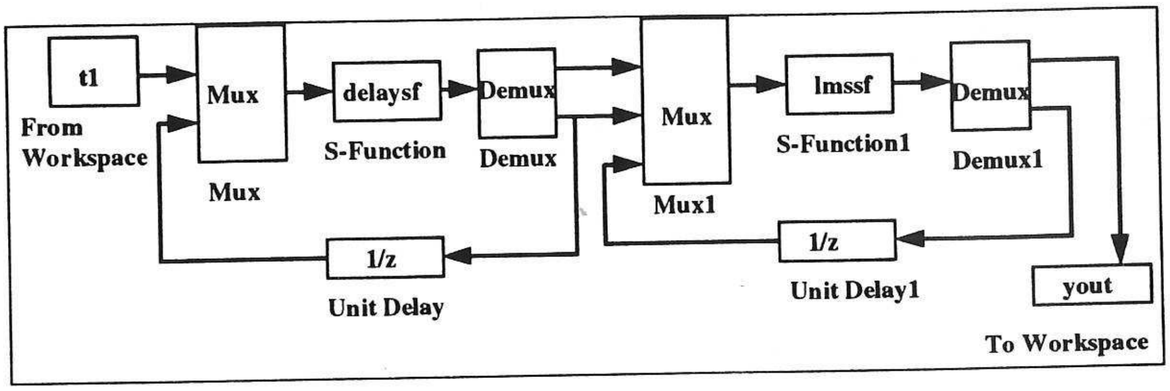


Figure 8: Simulink diagram of the LMS algorithm.

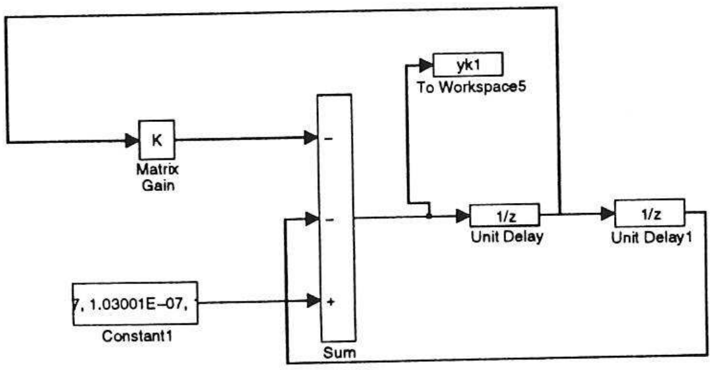


Figure 9: Simulink diagram of the beam simulation algorithm.

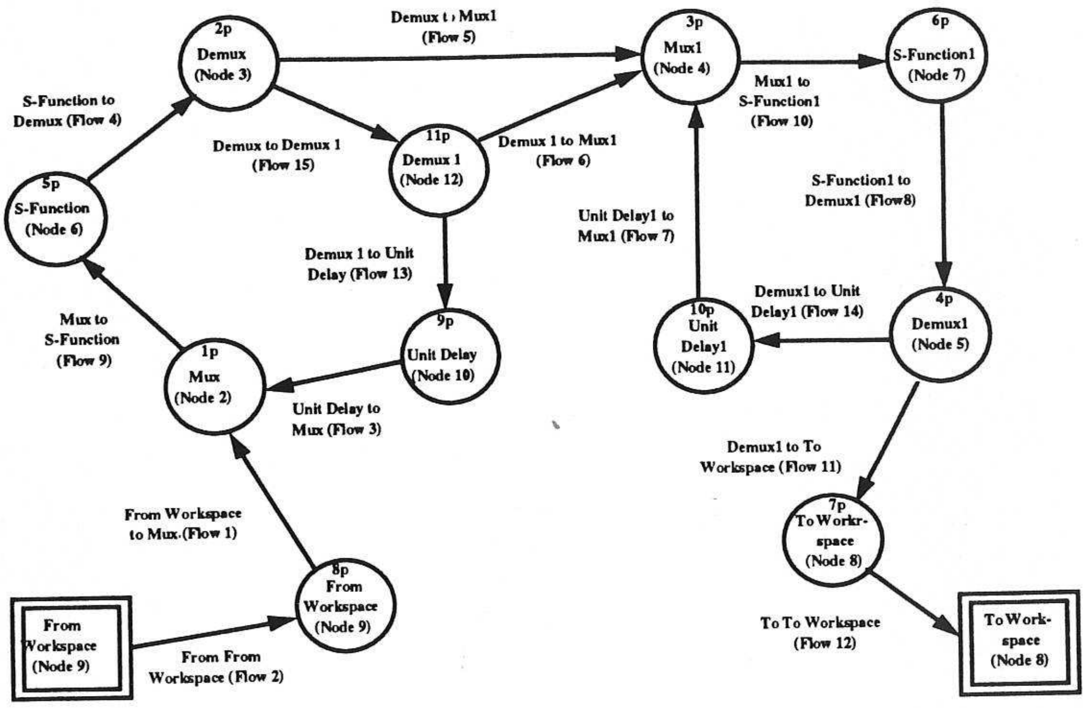

2p Demux (Node 3)
Demux t Mux1 (Flow 5)
3p Mux1 (Node 4)
6p S-Function1 (Node 7)
S-Function to Demux (Flow 4)
Demux to Demux 1 (Flow 15)
11p Demux 1 (Node 12)
Demux 1 to Mux1 (Flow 6)
Mux1 to S-Function1 (Flow 10)
5p S-Function (Node 6)
S-Function1 to Demux1 (Flow8)
Unit Delay1 to Mux1 (Flow 7)
Demux 1 to Unit Delay (Flow 13)
Mux to S-Function (Flow 9)
1p Mux (Node 2)
9p Unit Delay (Node 10)
Unit Delay to Mux (Flow 3)
10p Unit Delay1 (Node 11)
Demux1 to Unit Delay1 (Flow 14)
4p Demux1 (Node 5)
Demux1 to To Workspace (Flow 11)
From Workspace to Mux (Flow 1)
7p To Workr-space (Node 8)
8p From Workspace (Node 9)
From Workspace (Node 9)
From From Workspace (Flow 2)
To To Workspace (Flow 12)
To Work-space (Node 8)

Figure 10: Data flow diagram of the converted LMS algorithm.

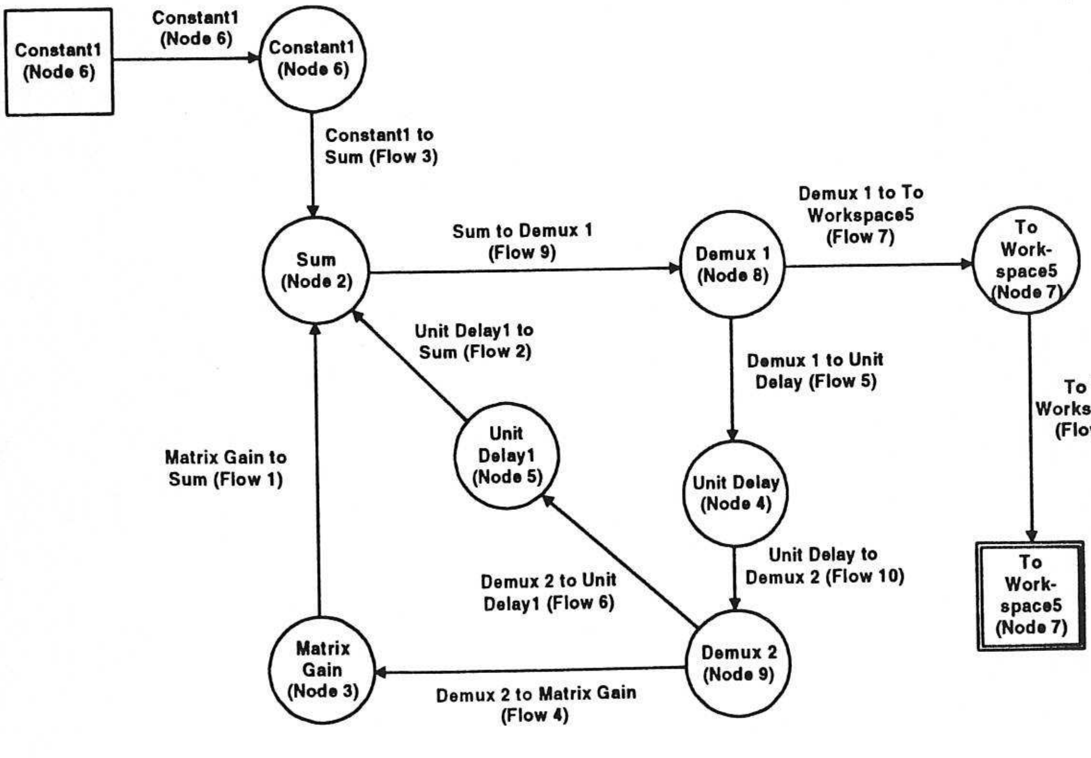

Constant1 (Node 6)
Constant1 (Node 6)
Constant1 (Node 6)
Constant1 to Sum (Flow 3)
Sum (Node 2)
Sum to Demux 1 (Flow 9)
Demux 1 (Node 8)
Demux 1 to To Workspace5 (Flow 7)
To Work-space5 (Node 7)
Unit Delay1 to Sum (Flow 2)
Demux 1 to Unit Delay (Flow 5)
To To Workspace5 (Flow 8)
Matrix Gain to Sum (Flow 1)
Unit Delay1 (Node 5)
Unit Delay (Node 4)
Unit Delay to Demux 2 (Flow 10)
Demux 2 to Unit Delay1 (Flow 6)
To Work-space5 (Node 7)
Matrix Gain (Node 3)
Demux 2 (Node 9)
Demux 2 to Matrix Gain (Flow 4)

Figure 11: Data flow diagram of the converted beam simulation algorithm.

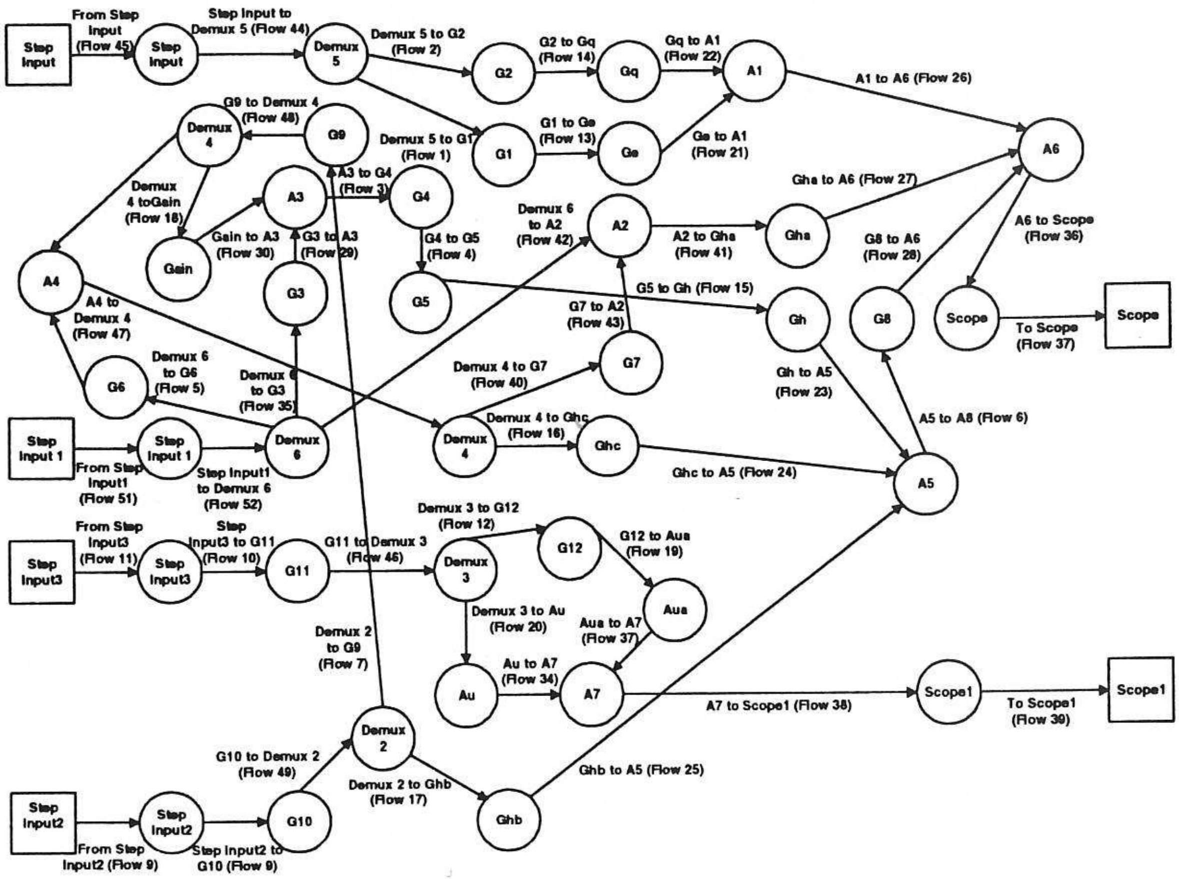


Figure 12: Data flow diagram of the converted VAP.

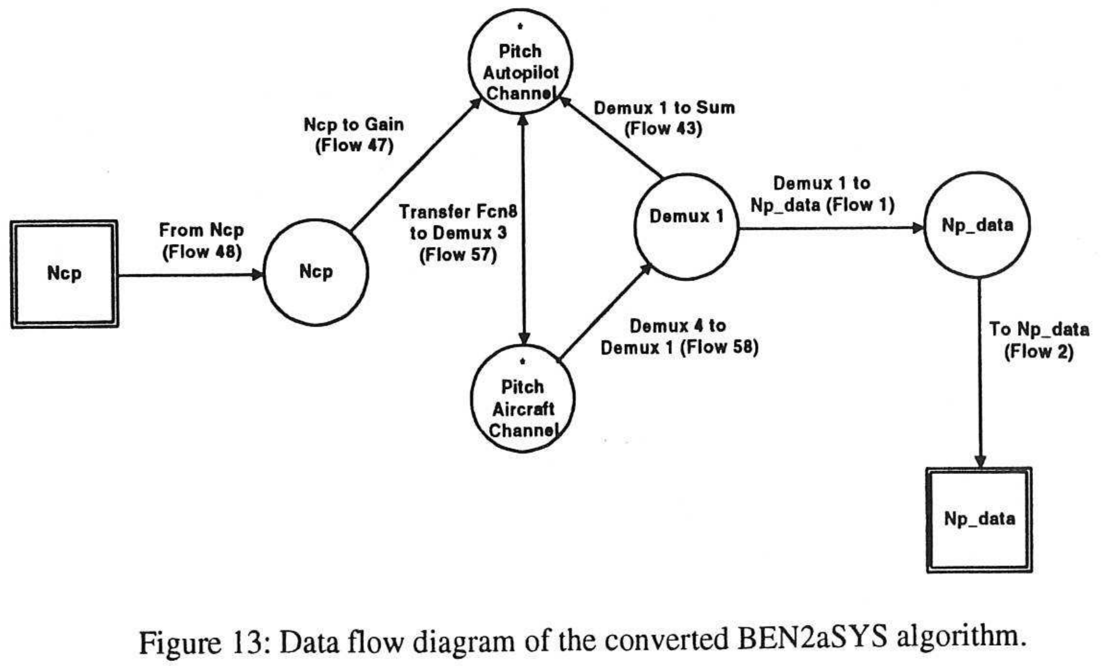


Figure 13: Data flow diagram of the converted BEN2aSYS algorithm.

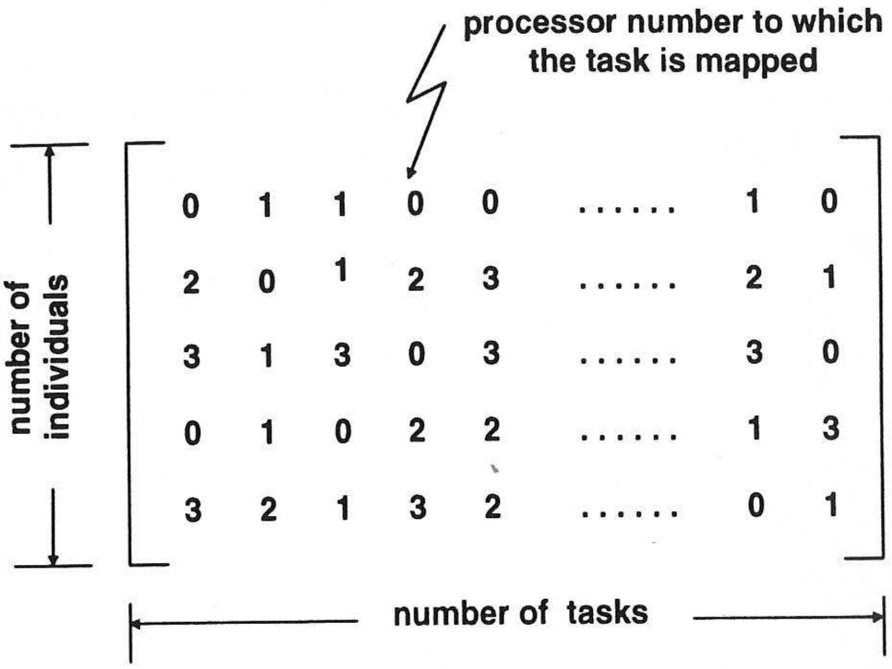


Figure 14: Example of a small population.

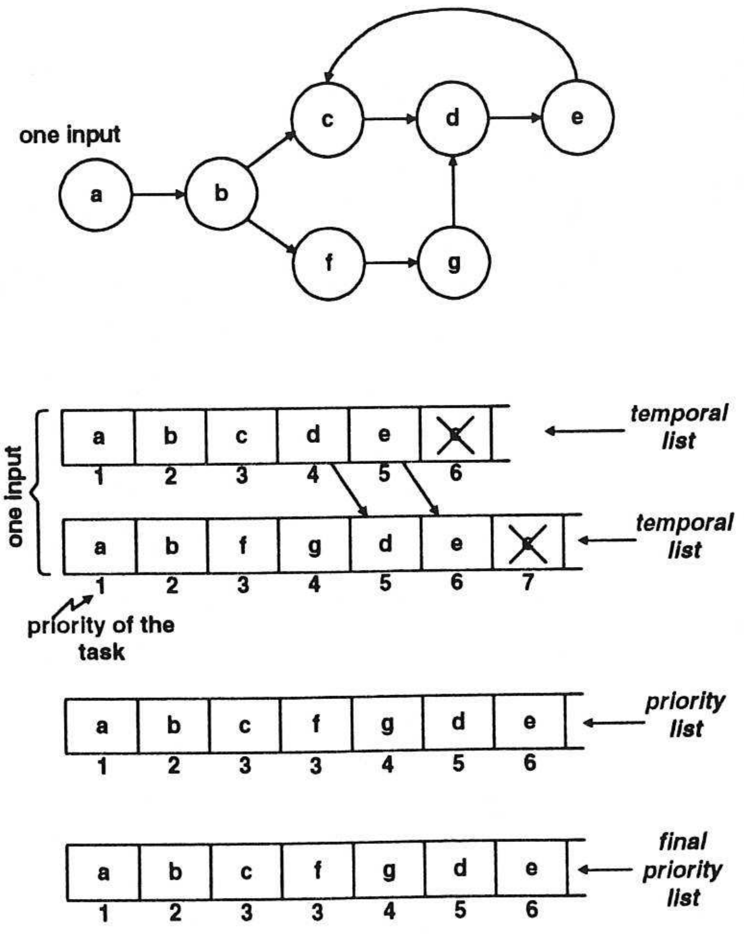


Figure 15: Procedure of obtaining the final priority list.

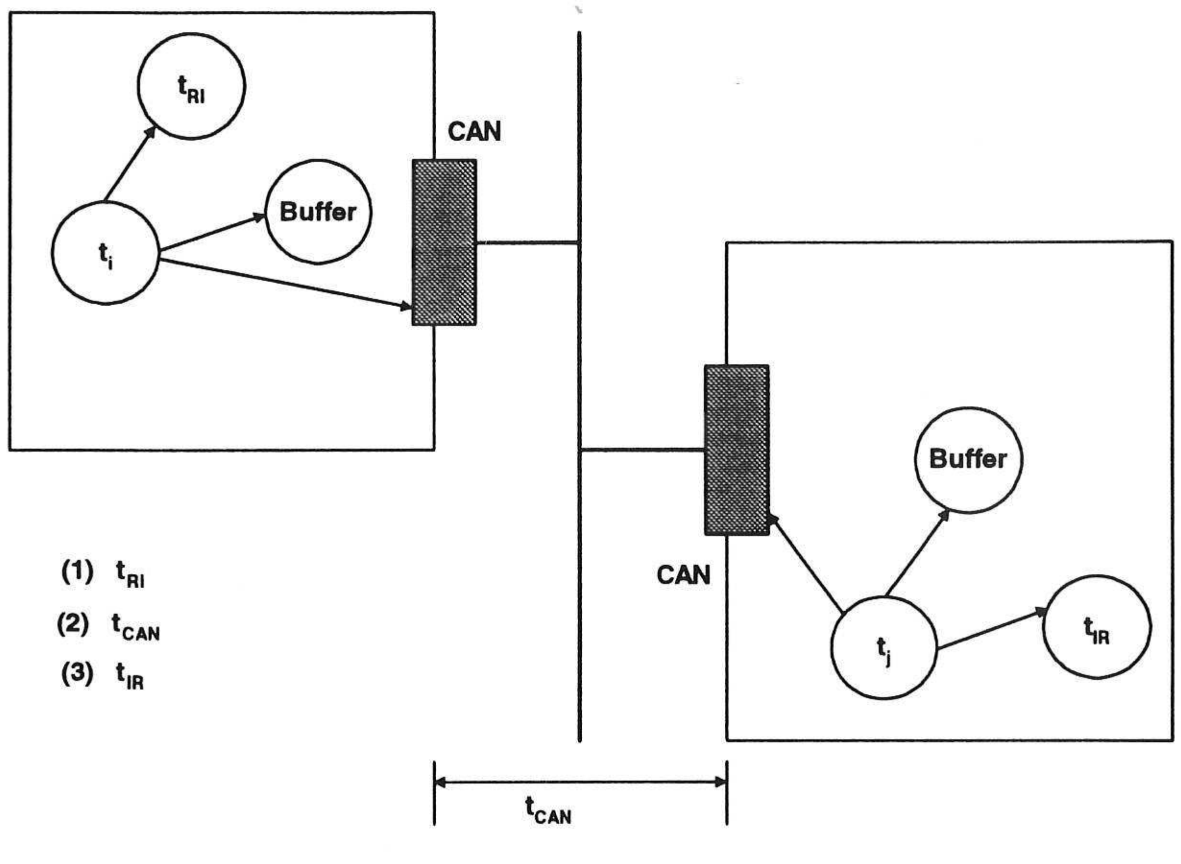
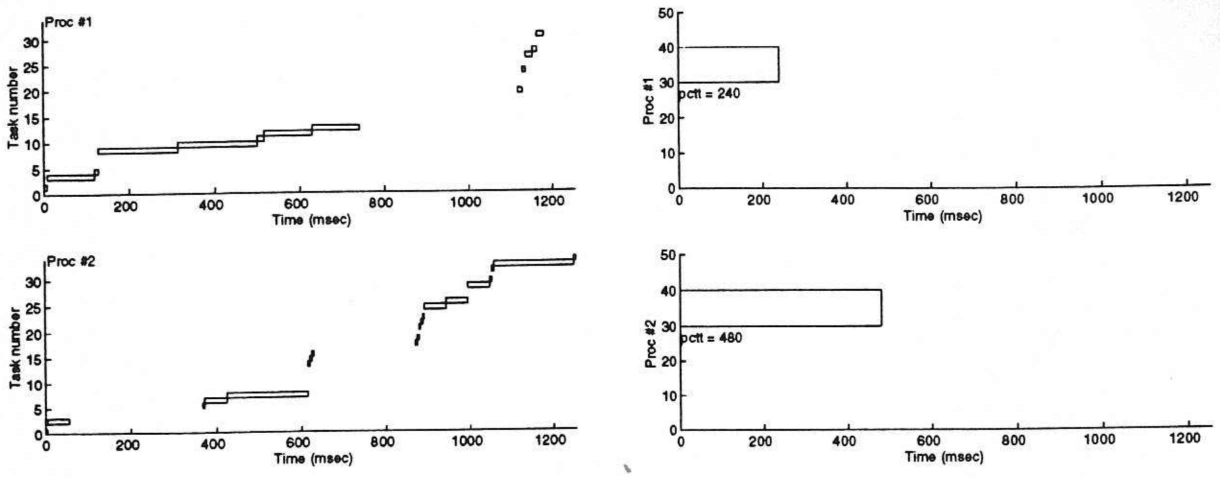
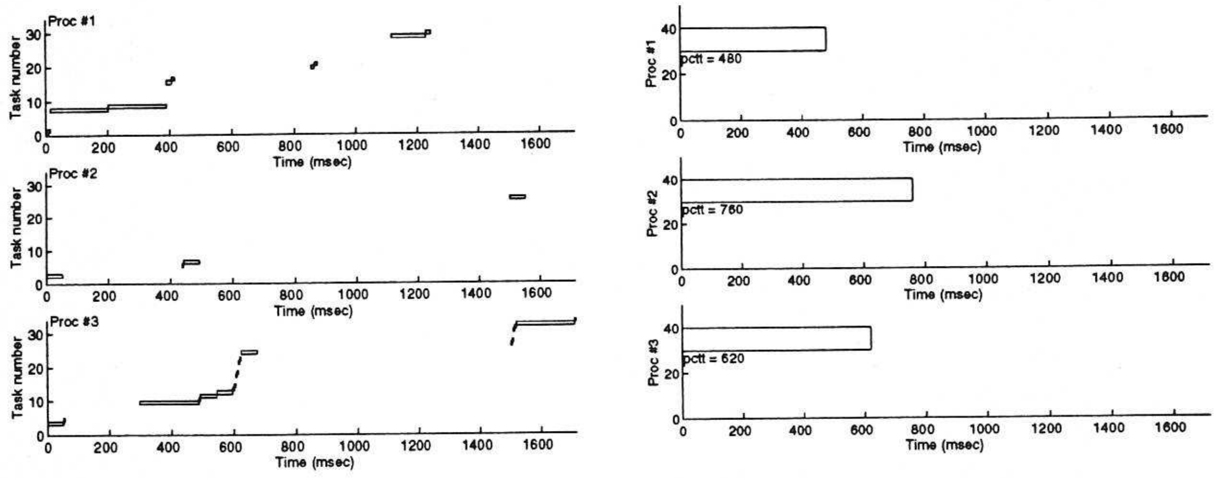


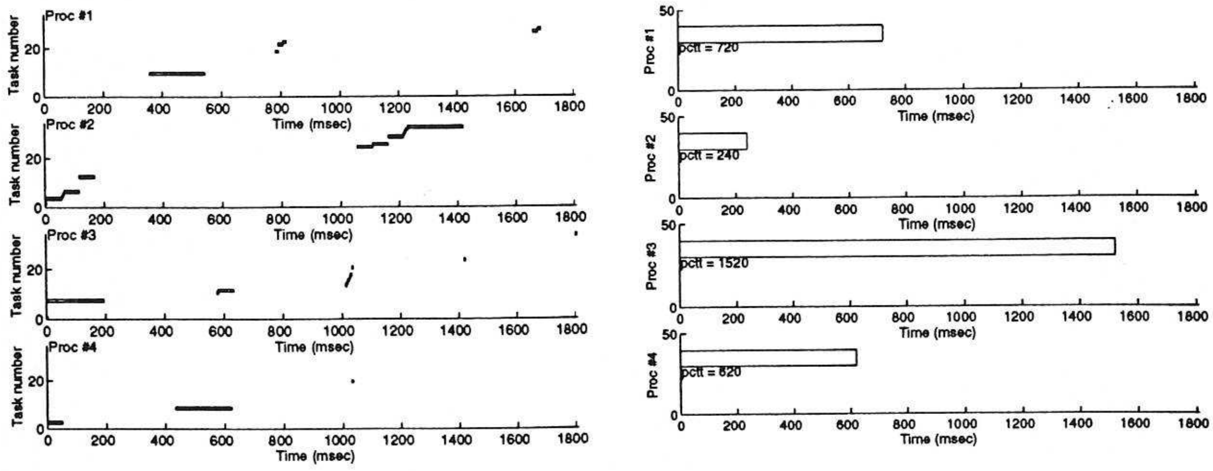
Figure 16: Communication times in the distributed system based on CAN bus.



(a) Two processor implementation.



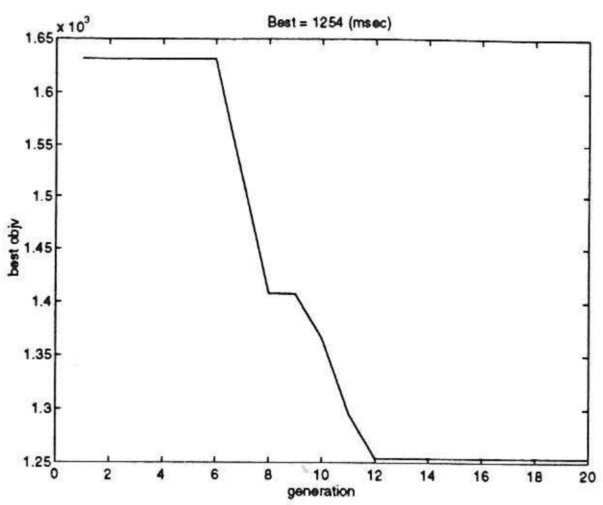
(b) Three processor implementation.



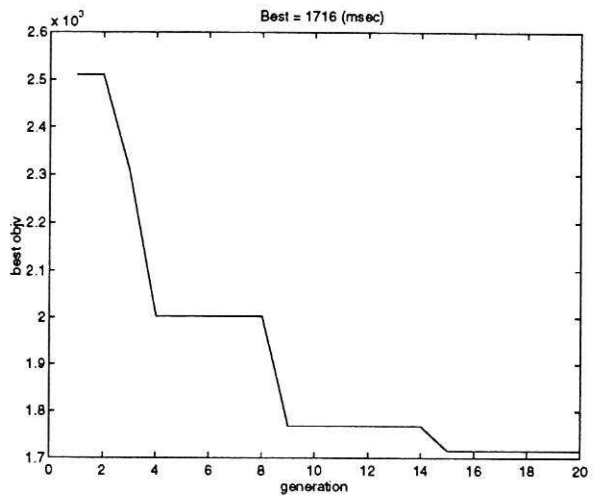
(c) Four processor implementation.

Proc #1: T8, Proc #2: C40, Proc #3: C40 and Proc #4: C40.
pctt : inter-processor communication.

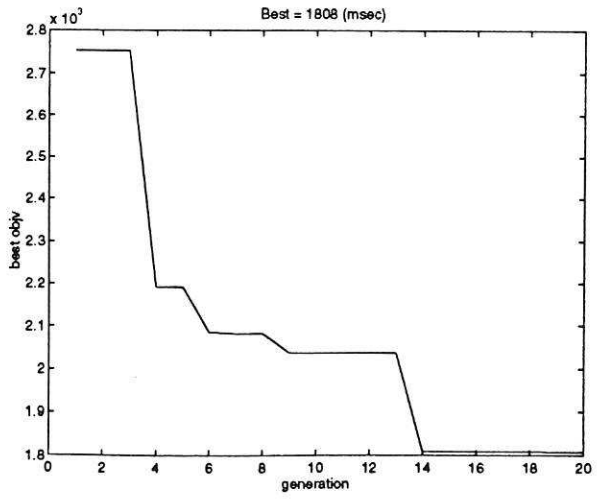
Figure 17: Scheduling and inter-processor communication of the VAP for two, three and four processors.



(a) Two processors.



(b) Three processors.

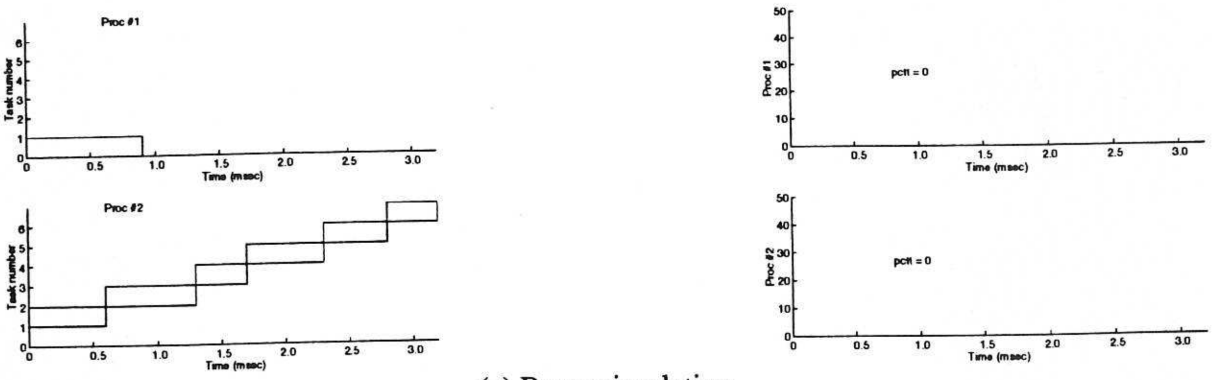


(c) Four processors.

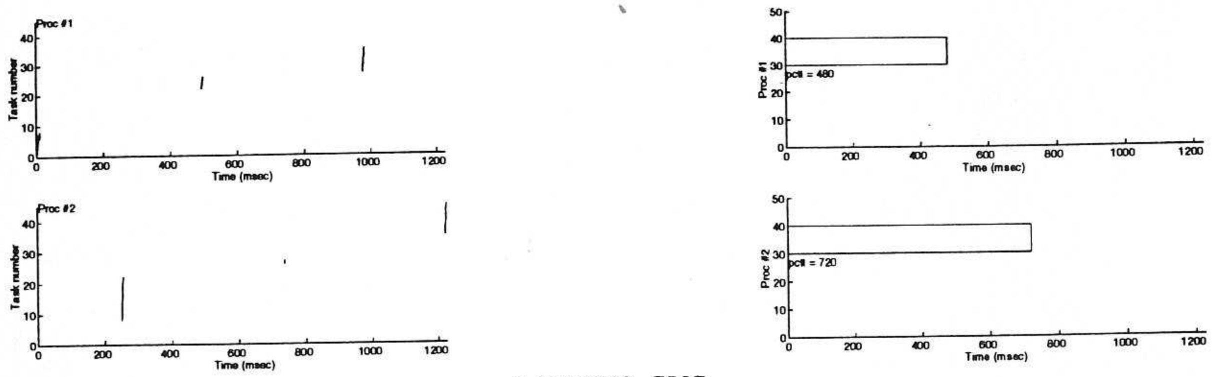
Figure 18: Objective values reached for the VAP with two, three and four processors.

Scheduling

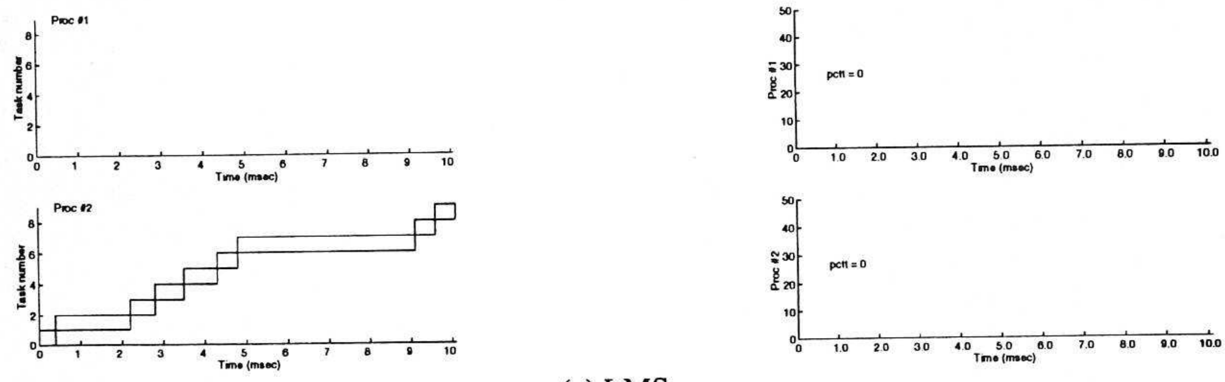
Inter-processor communication



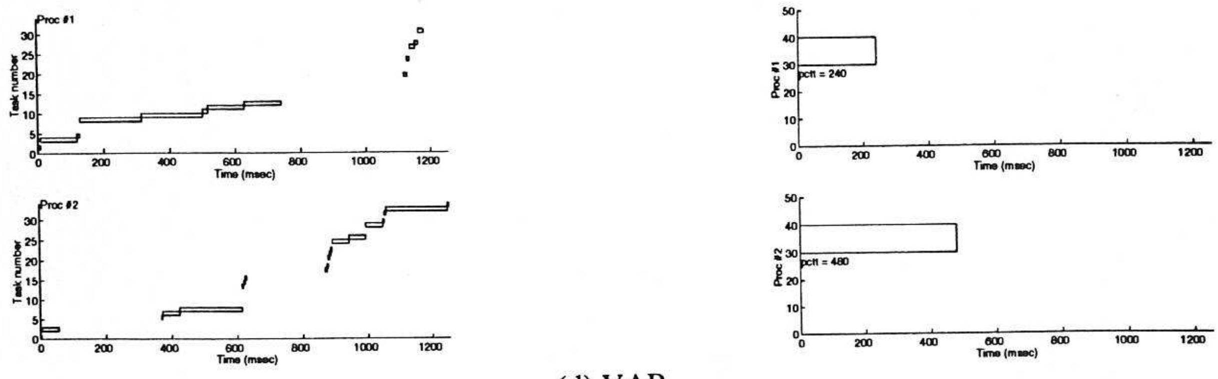
(a) Beam simulation.



(b) BEN2aSYS.



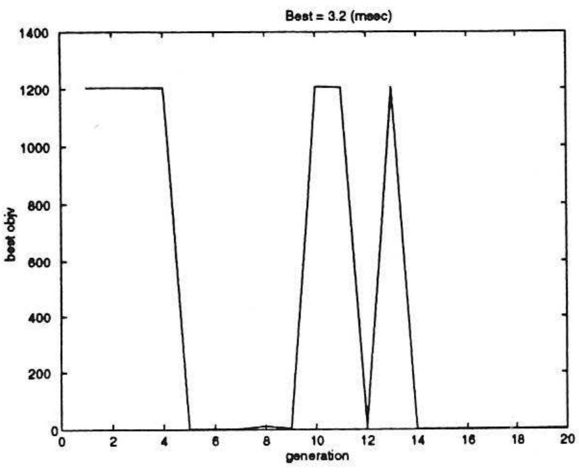
(c) LMS.



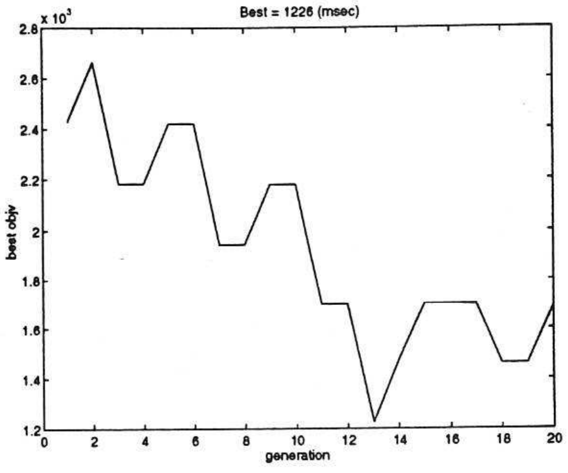
(d) VAP.

[Proc #1: T8 and Proc #2: C40. pctt : inter-processor communication.]

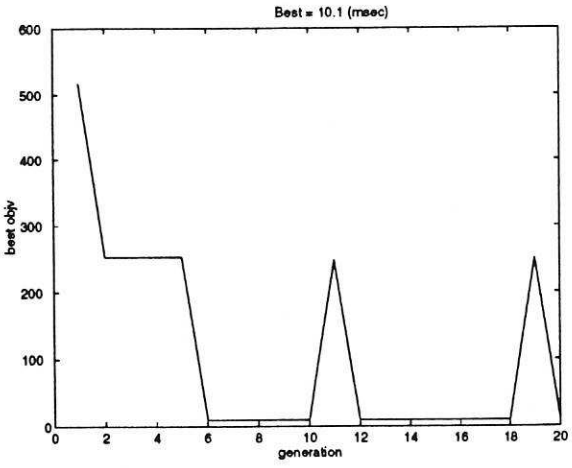
Figure 19: Mapping and scheduling of the LMS, beam simulation, VAP, BEN2aSYS using the PBGA approach (parallel heterogeneous architecture).



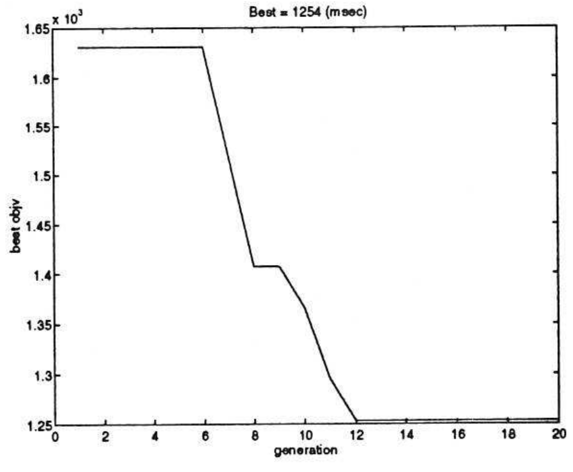
Beam simulation



BEN2aSYS

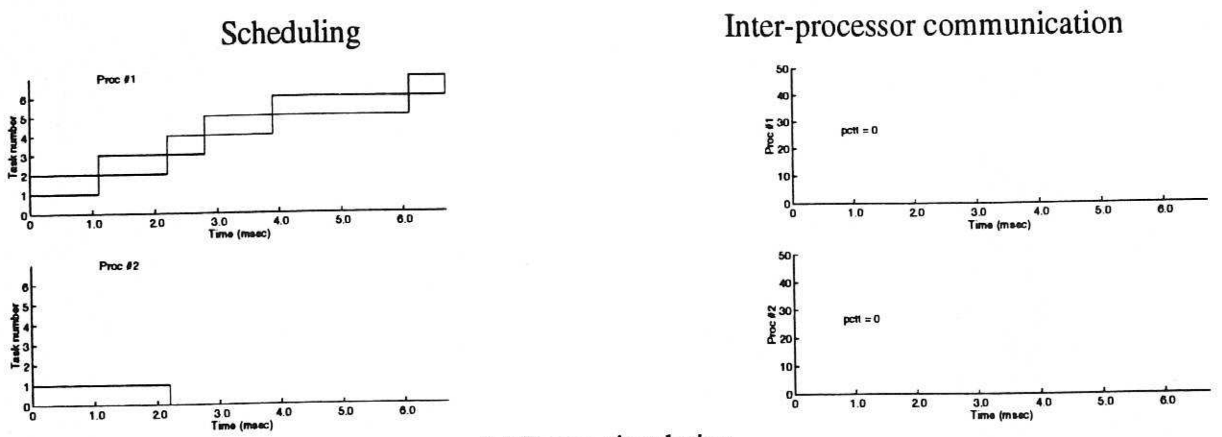


LMS

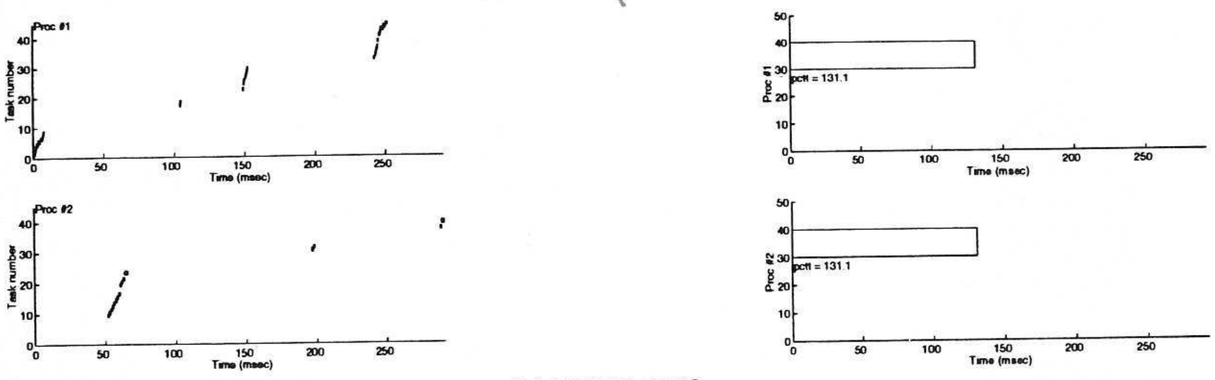


VAP

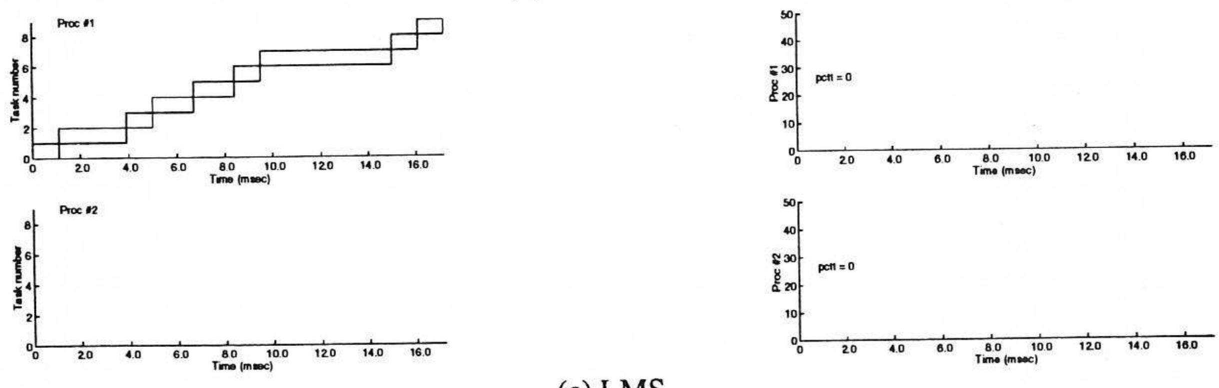
Figure 20: Objective values with mapping and scheduling of the LMS, Beam simulation, VAP, BEN2aSYS using the PBGA approach (parallel heterogeneous architecture).



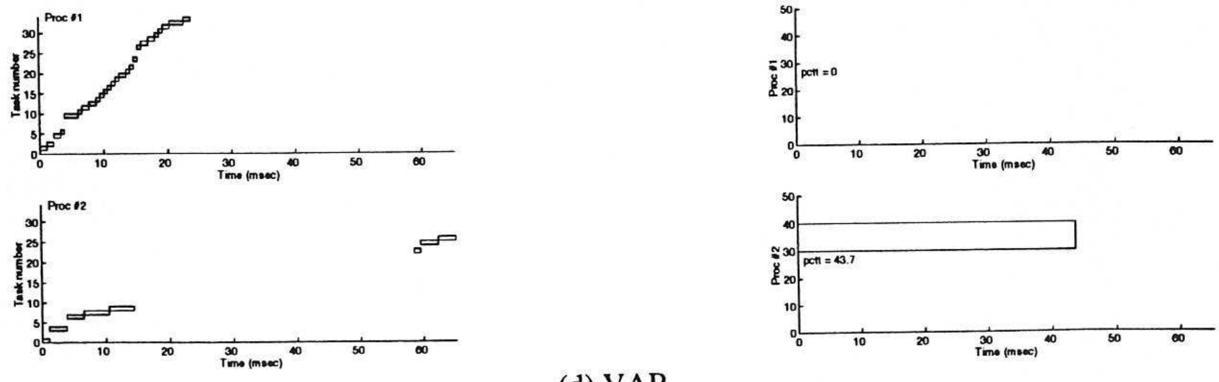
(a) Beam simulation.



(b) BEN2aSYS.



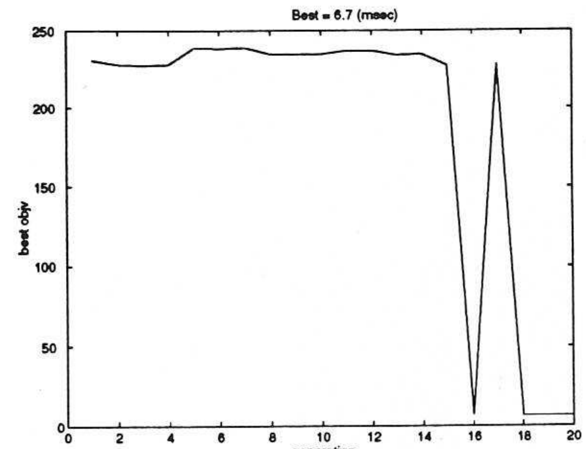
(c) LMS.



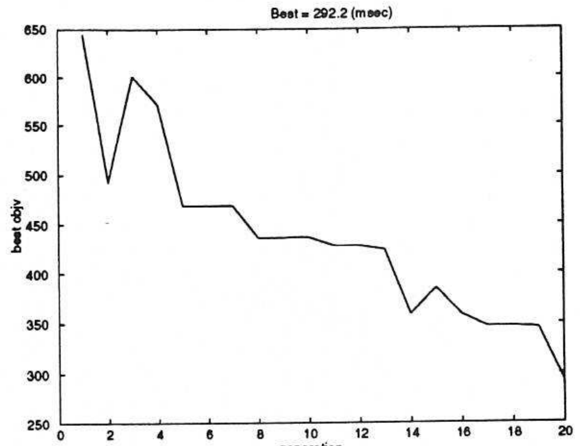
(d) VAP.

[Proc #1: 486 and Proc #2: 386. pctt : inter-processor communication.]

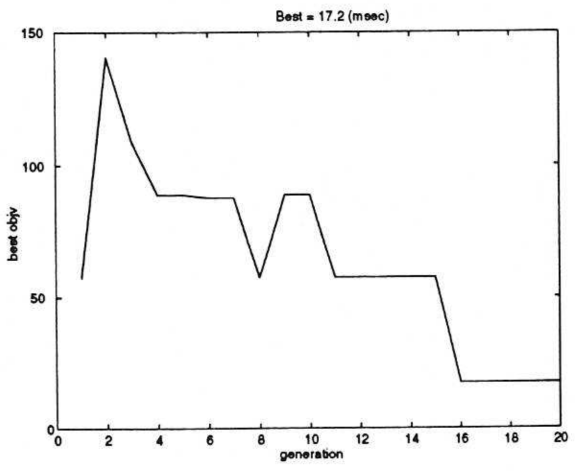
Figure 21: Mapping and scheduling of the LMS, beam simulation, VAP, BEN2aSYS using the PBGA approach (distributed heterogeneous system).



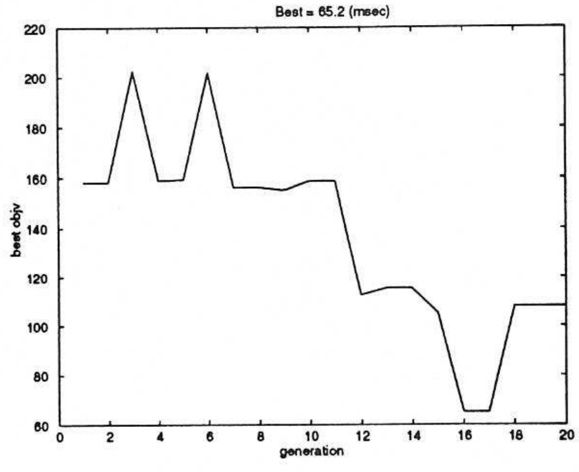
Beam simulation



BEN2aSYS

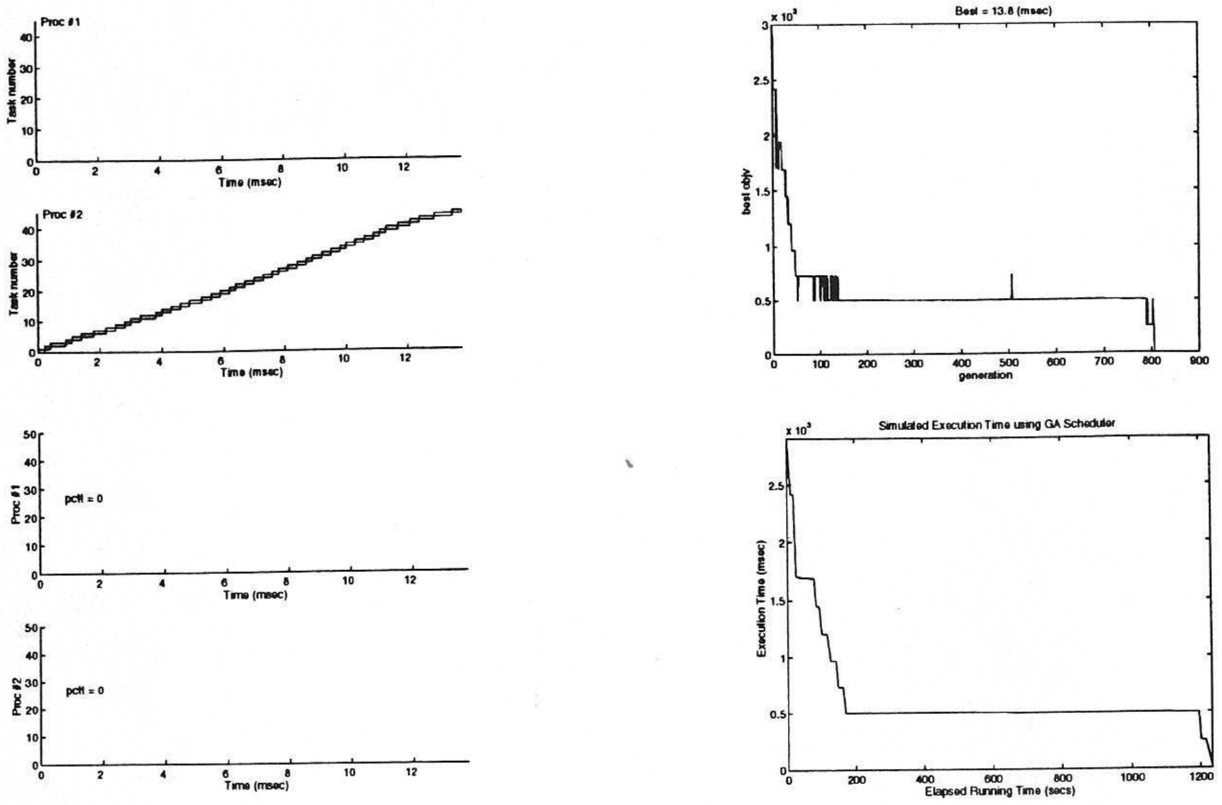


LMS

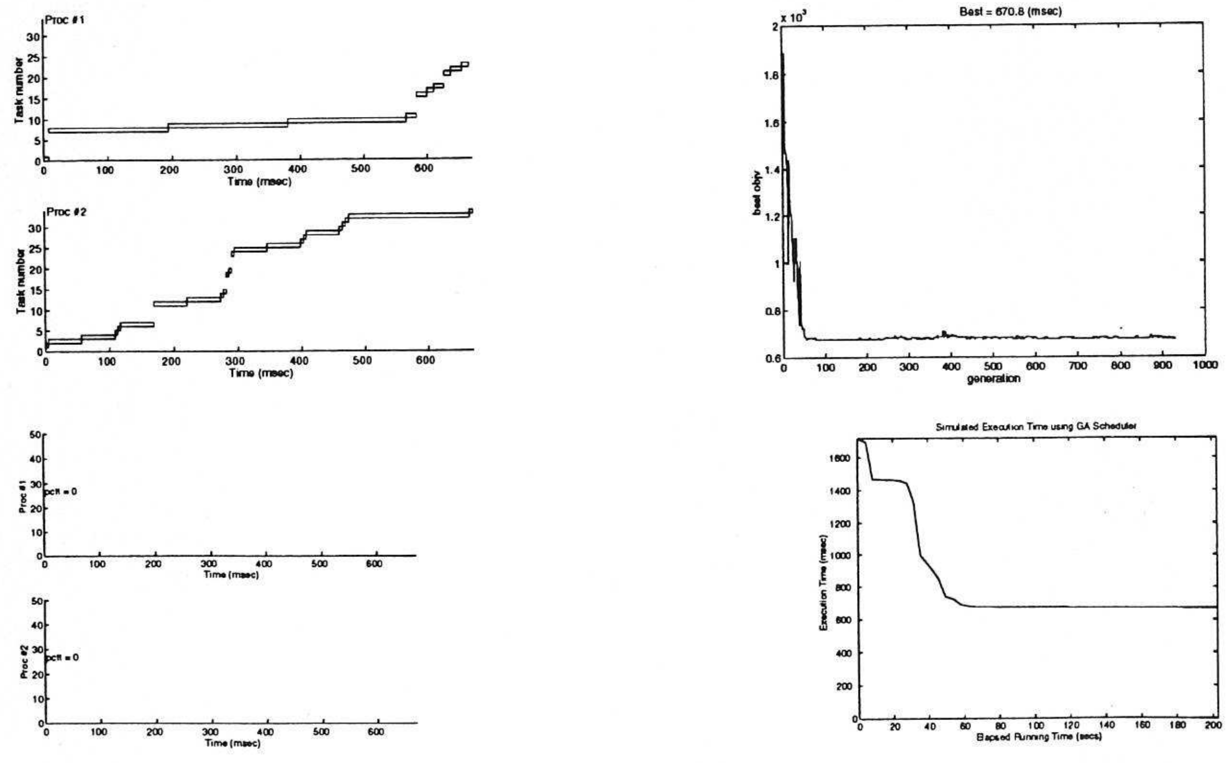


VAP

Figure 22: Objective values with mapping and scheduling of the LMS, beam simulation, VAP, BEN2aSYS using the PBGA approach (distributed heterogeneous system).



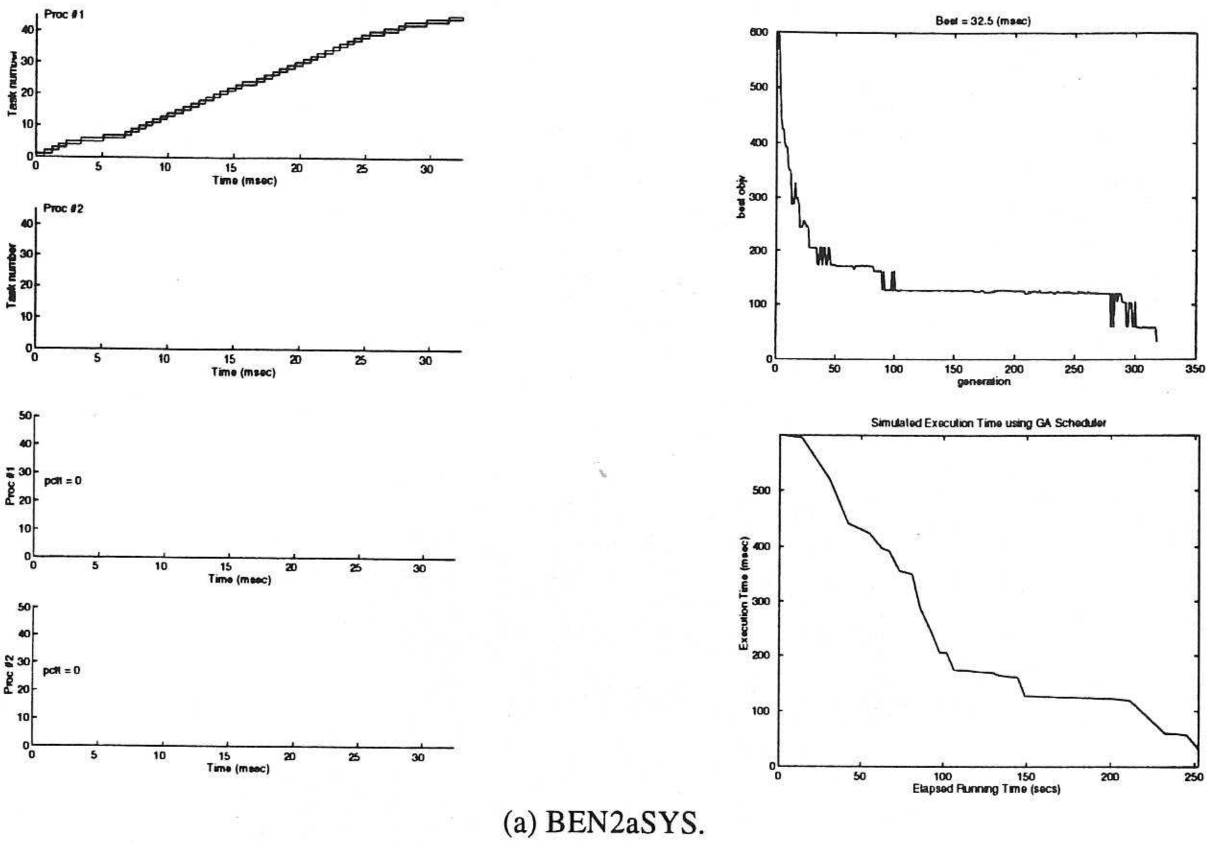
(a) BEN2aSYS.



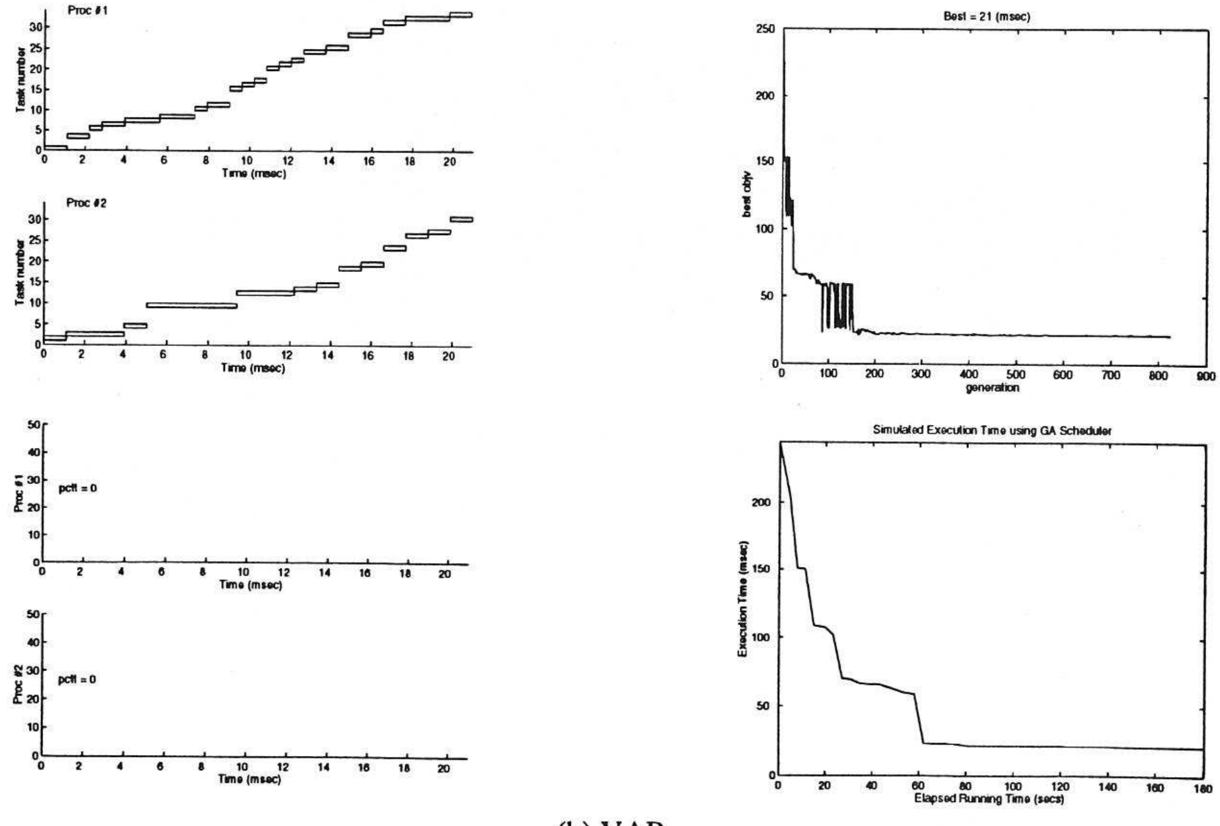
(b) VAP.

[Proc #1: T8 and Proc #2: C40. pctt: inter-processor communication.]

Figure 23: BEN2aSYS and VAP (parallel heterogeneous architecture), optimal values.



(a) BEN2aSYS.



(b) VAP.

[Proc #1: T8 and Proc #2: C40. pctt: inter-processor communication.]

Figure 24: BEN2aSYS and VAP (distributed heterogeneous system), optimal values.