INTRODUCTION

Recently the systematic application of advanced control algorithms to mechanical systems has been given the name 'mechatronics'. This term originates from Japan, where in 1989 the first International Conference on Advanced Mechatronics was organized by JSME [1989]. In a mechatronic design it is continuously considered whether the desired properties can be better realized by changing the mechanical construction or by adding electronic control, rather than adding the control system after finishing the design of the construction. This enables the design of systems with superior performance. Typical examples of mechatronic systems are, for instance, a compact disc player and an advanced photo camera with many electronic functions. Also robots can be considered as part of mechatronics.

The realization of a mechatronical system, advanced control algorithms and fast computer systems are needed. Because mechanical systems have very fast dynamics, a high sampling frequency is essential. Conventional, sequential, computer systems are too slow. By using parallel computing, the sampling frequency can be increased and more complex control algorithms can be realized. However, the use of parallel computing requires that the total control of the realization of a real-time digital controller be reconsidered. This paper gives a systematic analysis of the problems which are met when a real-time parallel computer system is to be set up and gives solutions to various of these problems.

Although the ideas presented are more generally applicable, main emphasis will be given to the use of transputers. The transputer is a new type of processor which was designed to be used in parallel. Simultaneously with the development of the transputer a new parallel programming language, Occam, was introduced by the EMI/BSI Laboratory. This language and the transputer are both developed to exploit the parallel features of a transputer network, the controller has to be written in a parallel form. Various forms of parallelism are discussed in Sections 4, 5, and 6. They range from a general-purpose form of parallelism, where the program is split up into basic elements which can be calculated in parallel, to a much coarser form of parallelism, where ad hoc some large parallel parts are selected. The performance of the transputer-based control system with various types of parallelism will be compared with a controller realized in conventional, sequential hardware. The paper concludes with some conclusions and suggestions for future research in this area.

2 THE TRANSPUTER AND OCCAM

One way out of the calculation versus sampling time dilemma has for long been sought in the use of parallel computing. In the past, many attempts to realize parallel systems with traditional processors, have failed due to the software overhead burden. The theory of Communicating Sequential Processes by Hoare [1978] forms the foundation of the programming language Occam. Occam uses the concept of processes. Communication between processes is done by means of channels. Due to the synchronisation properties of these channels the restriction to one processor no longer exists. The transputer hardware development was the next logical step. By using the formal specification language Z, the correct behavior of the implementation of the Occam constructs in hardware was formally proven. The hardware/software integration resulted in the transputer - Occam combination. The transputer may be considered a building block for parallel computers. Occam enables programming across these parallel transputers.

In the following paragraphs the operation of the transputer will be explained, based on the block diagram in fig. 1. From this block diagram a number of significant elements of the transputer chip can be identified. One of the typical features of the transputer is that it has four external links which enable a fast data exchange to other transputers. This forms not only the basis for the parallel architecture of a transputer-based computer system, but it also enables a systematic design of a digital controller into various layers such as an interface layer, a protection layer a control layer etc. This concept is worked out in Section 4. In Section 5 the concepts developed in this paper will be illustrated with an example of a typical mechatronical system, a flexible manipulator, where the inevitable vibrations are actively damped by a parallel control system realized with four transputers. A robust state-feedback controller, designed with a pole placement technique, is able to give the controlled system the appearance of a rigid construction. In order to exploit the parallel features of a transputer network, the controller has to be written in a parallel form. Various forms of parallelism are discussed in Sections 4, 5, and 6.

2.1 The OCCAM process

The underlying idea of Occam is the notion of a process. A process is a part of a program that starts, performs an action and then terminates. The idea is that processes may be executed in parallel. The communication between processes is done via self-synchronizing channels. A channel is a one way point-to-point connection from one process to
The transputer has an interrupt signal pin connected to the event that is activated every time the event pin is triggered. The program starts

2.3 Interrupts

channel is associated with the actual hardware pin.

because there is no data transport over the event channel. This event interrupt signal is illustrated in the following Occam interrupt process

The transputer has one high-priority and one low-priority operating mode. The interrupt process should probably be executed in the high-priority mode in order to reduce the interrupt response time. Theoretical response times of approximately 55 µsec have been reported by Welch (1997)

2.4 The Timer

The timer channel should also be considered an input channel. Times of the internal running clock are obtained by reading the timer channel and operating on the time value. The Occam code to use the timer is illustrated with the following sequential process that causes a delay.

PROC delay (VAL INT interval)
  TIMER clock:
    INT time:
    SEQ
      clock ? time
      clock ? time
      AFTER time PLUS interval :

Internally the timer queue differs from the ready queue in that it is kept sorted in the sequence of the wake up times of the different timed processes.

2.5 The Scheduler

In addition to the channel protocol concept, there is another important difference between transputers and other microprocessors. That is the built-in scheduler. The transputer should be considered as handling tasks or processes rather than machine instructions. These processes are controlled by a built-in scheduler. The process administration is the main activity of the scheduler. The scheduler keeps track of the different processes running on transputers A and B. If the process on transputer A arrives at the input instruction on link 2 and transputer A is not yet ready to provide this output, the process running on transputer A will be de-scheduled until transputer B signals that it is ready to send the data over link 2. At that point the inputting process on transputer A is resumed and the data is transferred via link 2. This results in a synchronizing action between the processes running on different transputers. In this way the transputer channels provide the necessary system interconnection and synchronization. The only thing the program has to do is to define the appropriate input and output actions.

2.5.1 High-priority mode

The high-priority or the low-priority mode. In the high-priority mode the executing process can not be interrupted by another process (non-preemptive scheduling). De-scheduling can only occur when:

• a process is completed
• communication with a channel / link is not (yet) possible
• the process waits for a timer to elapse

2.5.2 Low-priority mode

In the low-priority mode the processes are, by means of time-slicing, scheduled in a round-robin manner. Here processes will be de-scheduled by:

• completion of the process
• awaiting link or channel communication
• expiration of a time slice

The de-scheduled process will be added to the back of the ready queue in the low priority mode, or it may be scheduled at once as a high priority process. The use of the transputer interrupt signal is illustrated in the following Occam interrupt process that is activated every time the event pin is triggered. This program starts with the definition of the process name with the external channel definition. The internal channel event is declared as the type ANY, because there is no data transport over the event channel. This event channel is associated with the actual hardware pin.
A Scheduler for Real-time Control

3.1 The Sampling Process

The sampling process of a control system may, according to Bakkers (1987), be divided into:

1. Time-bounded processes, such as the sampling or actuation actions.
2. Time-limited processes such as the calculation of the control action.
3. Background and alarm processes.

The time-bounded processes have to be scheduled at specific instants. The time-limited processes should be scheduled to meet their deadline. Synchronization between these processes is required in order to guarantee that the calculation is performed before the actual control action takes place. If a control system cannot inherently give this guarantee the resulting misfires of the correct synchronization should:

1. Result in a non-detrimental control action.
2. Result in error messages that are meaningful to the control engineer.
3. Never cause a deadlock of the system.

3.2 Real-Time Scheduling

From the theory of real-time scheduling, Liu (1973) has defined the optimum scheduling algorithm for the execution of control processes: it is the deadline driven scheduler. A sub-set of it is the monotonic rate scheduler, which can be used in the case of fixed priorities. An example of the deadline-driven scheduling rule is illustrated in fig. 2.

Fig. 2 Deadline-driven scheduling

Note the position indicated where a fixed-priority monotonic rate algorithm would have resulted in a selection of process 2 instead of process 3. In general, a schedule of a number of different processes may be defined as:

\[
\text{Schedule} = \{ T_i \}, \{ t_i \}, \{ d_i \}, \{ b_o \}
\]

where:

- \( T_i \) = one of \( n \) processes
- \( t_i \) = the time to execute process \( T_i \)
- \( d_i \) = the period of process \( T_i \)
- \( b_o \) = start of scheduling process \( i \)

This may be illustrated as follows:

The deadline-driven algorithm assigns the processor at each decision-time, to the process whose deadline is closest. A necessary condition is caused by the factor that a process at most may be kept busy at the time of:

\[
\sum b_o \leq 1
\]

Although this is a necessary condition, it has not been proven that this condition is also sufficient. Only for the sub case in which \( d_i = k \) was proven that the necessary condition is also sufficient. If we also make the assumption that in a control system all sampling actions start at the same time, we have the additional requirement of:

\[
b_o = b_k
\]

The optimum scheduling algorithm has not been implemented yet in industrial real-time operating systems (Barkovic, 1988). The main reason is the computing effort necessary to keep the priority list of the processes sorted at all times. This rule however plays an important role in the manual scheduling of processes in the traditional control system design. It must be stressed however that this optimum rule is defined only for a single processor case.

With the transputer hardware it is possible to construct and program a parallel machine. The question arises how the control system design should be performed with such hardware. One should realize that the process scheduler on the transputer chip is a round robin scheduler which in low-priority mode performs a process switch about every msec.

There is also a high-priority mode in which the time slicing is disabled so that the processes run till completion or are de-scheduled due to a wait for input or output. In order to realize deadline scheduling on a transputer a scheduler that accommodates priorities will have to be implemented.

In the ideal case, the allocation of the priority ordering should be performed by a hardware ordering mechanism on the microprocessor chip. A large number of priorities will have to be implemented. This will result in a fine control over the priorities. Because a deadline driven priority system the priority axis is basically the time axis, this large number of priorities results in a fine resolution of the time axis, together with a reasonable cycle time for the total process cycle: of a few days. In order to restrict the length of the priority sort list to an hardware-technical implementable value, the processes that are a part of a set of processes or tasks should get the same priority as their main process. This allows the use of one process queue per priority class.

The scheduling in a deadline driven scheduler should be a preemptive scheduler. This requires far more overhead than the present low-priority scheduler that schedules only on certain instructions, like a jump. Thus ensuring that the data structure (stack) that has to be saved is minimal. If this way of thinking is adopted for the priority scheduler a so-called semi-preemptive scheduler will result that performs well within the specification of the real-time requirements and also maintains a fast process switch time. The proposed priority system should accommodate:

- Time bounded processes at Hi-Priority preemptive
- Alarm processes at High priority
- Time limited processes at Medium priority
- User process at Low priority
- Background processes at Round-robin scheduler

Future implementation of a deadline driven scheduler for the time bounded control tasks should be considered by the hardware manufacturers.

The assignment of the priorities is the task of the control engineer. He will have to analyze the time requirements of the system processes beforehand. If the task is not performed, the scheduler should fall back on the lowest, or round-robin, scheduler without loss of performance.

A sub-optimal solution is the following. Allow enough time for all the time-limited tasks to be completed on time. Add a safety margin. Because the completion of the tasks can never be guaranteed, timer guards should be installed to detect processes exceeding the time limit. A transputer timer process is well suited for this purpose.

4 System Architecture

The use of transputers in a control system design necessitates the design of a transputer network that reflects the parallelism of the particular control system. There is no standard procedure to convert a sequential program into a transputer topology with a corresponding parallel program. Therefore, the system architecture should be tailored to the (control) problem. There are basically three methods to obtain a parallel architecture, i.e.

1. Parallelism by recognizing various layers. This is further worked out below.
2. Ad-hoc based on the properties of the problem. In this situation it is necessary to recognize dedicated parallelism of the control system. An example of this technique is used with the controller for the flexible robot arm that is described in Section 5.
3. By means of a mechanism that creates massive parallelism from repetitive use of basic building blocks. For distribution of these blocks over a number of processors a method developed by Hilloran (1987), should be used to obtain an optimal distribution of the tasks over a given network. This is described in more detail in Section 6.

As a result of experience gained in the realization of several transputer-based control systems by Bakkers (1986) and Stavenuiter (1989), a layered architecture is proposed. The basic idea is that there are several layers in a practical control system as indicated in fig. 3 for a robot control system. A more detailed description of the different layers is given below.
4.1 First Layer
The first layer consists of the hardware and software directly connected to the sensors of the control system. It can be realized as the first transputer layer. This interface layer includes, for example: Analog-to-Digital, Digital-to-Analog and Resolver-to-Digital converters.

The software of this layer performs primarily the sampling and control action. This layer is also the right place to perform the necessary filtering of the measured data. The processed measurements are available at the link interface input and outputs. Schematically, the first layer and its interconnections to the second layer and the monitor and display may be represented as in fig. 4.

4.2 Second Layer
The second layer consists of transputers that execute the safety software. This layer completely isolates the sensors from the rest of the control system. Actual sensor data arrive here and may be compared with the control signals to check whether the combination of the two may lead to a dangerous situation. If so, the safety software has priority and will set the control signals at a safe value. The topology of this layer could include one or more transputers, as illustrated in fig. 5, where T800 transputers are used.

4.3 Third Layer
The third layer typically contains the first calculation layer, in robot terms this would be the calculation of the robot dynamics. It is not very likely that this calculation can be performed in one transputer. Therefore, as an example, a transputer network of eight transputers is given in fig. 6. For the distribution of the code over this topology, the method of Hilhorst (1987) may be used. This method, based on the list scheduling analysis, also takes communication times via the links into account. By using a critical path analysis, an (almost) optimum distribution of the various processes over the different transputers can be obtained.

4.4 Test Layer
This layer collects the properly buffered test data and sends it to a graphical display. It acts as a software test rig. The test data may also include warnings for synchronization errors and sensor data. The synchronization error messages enable the control system designer not only to spot errors in the system design, but also associates these errors with the exact location of the erred sampling or control process. Schematically, this layer may be represented as in fig. 7.

One has to realize that parallel programs can not be tested for correctness by the insertion of a number of write statements in the program. The ad-hoc parallelism will be illustrated with the example of a flexible robot link as described by Stavenhuller (1989) and Krulse (1988). It is constructed as a long aluminum strip (100 x 6 x 0.4 cm) driven by a DC motor. When controlled as a rigid system, the flexible robot link will heavily vibrate. Therefore, the vibrations and bending have to be taken into account in the controller design. The bending of an accelerating link has been described by Fukuda (1983) as:

\[ W_i(t) = \sum_{n=1}^{\infty} W_n(t) M_n(t) \]  
(4)

where: 
\[ W_n(t) \], called the ‘mode shape’ function, describes the shape of the arm at its \( n \)th resonance frequency.

\[ M_n(t) \], the modal function, is a time dependent function which may be described by:

\[ M_n(t) = \frac{1}{2\pi} \frac{1}{\omega_n^2} \left( 1 - e^{-\frac{\omega_n^2 t}{2\pi}} \right) \]  
(5)

where:
\[ \omega_n \] = the \( n \)th resonance frequency
\[ \Phi_n \] = the acceleration of the arm
\[ z_i \] and \( A_i \) are arm dependent parameters.

The combination \( Y_n(t)M_n(t) \) is called a mode. Clearly, the mode shape function cannot be controlled. However, the flexible arm can be made to rotate like a rigid one by controlling the acceleration in such a way that the modal function decays rapidly. In practice, it is not possible to control an infinite number of modes, but the model of the arm can be simplified by assuming that the motor bandwidth is less than \( \omega_{\text{max}} \), which implies that excitation of frequencies higher than \( \omega_{\text{max}} \) may be disregarded. Furthermore, higher modes tend to have smaller amplitudes and therefore, they may be disregarded as well.

Simulations by Kruse (1989) indicate that control of the first three modes yields a satisfactory response. Furthermore, as these modes are independent, the control signal for the entire system can be computed by adding the control signals for the three modes. Taking the Laplace transform of (5) yields:

\[ M_n = \frac{2\pi M_0}{s^2 + 2\pi M_0} \]  
(6)

omitting subscripts, can be written as

\[ M = \frac{2\pi M_0}{s^2 + 2\pi M_0} \]  
(7)

This equation can be translated into the block diagram of fig. 8. The modal function \( M \) is controlled by using state feedback, where the states are defined to be \( \Phi \), \( JM \) and \( \int JM \), where \( M \) stands for:

\[ \int M \, dr \]  
\( \frac{1}{0} \)

From these states only \( \Phi \) can be measured, \( JM \) and \( \int JM \) have to be estimated. The estimation is updated by measuring the vibrations of the arm by means of the strain gauges. In the case that three modes are considered to contribute to the vibrations of the arm, three pairs of strain gauges are used. A similar controller and estimator is implemented for each mode. The controller and estimator for one mode is depicted in fig. 8.
In this case a quite natural parallelism is present that may be translated into a four transputer network configuration. The calculations of each mode will be assigned to one slave transputer. The master transputer collects the data and performs preliminary data analysis. This configuration may be represented as illustrated in fig. 9.

The use of transputers makes it necessary to use parallel programming techniques. The transputer environment may be programmed in Occam (the preferred parallel transputer language) or in parallel C. Methods of parallel programming are given in Bakkers (1988, 1989). Code from different programming languages may be cast into an Occam harness to exploit the parallelism between processes.

5.1 Natural system parallelism.
In the case of the flexible robot parallelism is naturally obtained by distributing the tasks of calculating the estimator and the controller of the three vibration modes over three transputer. This way the natural parallelism of the different modes is reflected in the architecture of the transputer system as illustrated in fig. 9. Because this is not a rule that can be applied to any system, this technique should be considered an ad-hoc technique. The advantage is that the processing speed remains the same whether one or ten modes are calculated, of course requiring one or ten transputers. The combination of this ad hoc system allocation together with the basic building block method, to be discussed in the next section, has been applied to the flexible robot arm.

6 THE BASIC BUILDING BLOCK METHOD
A systematic way to introduce parallelism, is to start with very small processes and build a system with these elementary processes. This approach is similar to methods used in simulation programs. The internal scheduler on the transputer will realize parallelism while running these processes. If this philosophy is applied to the controller design of the flexible arm we can realize the controller of fig. 9 with the aid of only five basic elements i.e.

- an integrator block: int
- a summer block: sum
- a minus block: min
- a gain block: gai
- a splitter block: tee

Let's first define these basic elements and then write the controller program using these basic elements.

PROC tee (CHAN OF REAL32 in, out0, out1)
REAL32 x:
SEQ
in ? x:
PAR
out0 ! x:
out1 ! x:

PROC gai (CHAN OF REAL32 in, out, init)
REAL32 x:
SEQ
init ? k:
WHILE TRUE
PRI ALT
init ? k:
SKIP
in ? x:
out ! k * x:

PROC sum (CHAN OF REAL32 in1, in2, out)
REAL32 x1, x2 :
WHILE TRUE
SEQ
PAR
in1 ? x1:
in2 ? x2:
out ! x1 + x2:

PROC min (CHAN OF REAL32 in1, in2, in3, out)
REAL32 x1, x2, x3:
WHILE TRUE
SEQ
PAR
in1 ? x1:
in2 ? x2:
in3 ? x3:
out ! -((x1 + x2) + x3):

PROC int (CHAN OF REAL32 in, out, init)
REAL32 x:
SEQ
init ? dx:
WHILE TRUE
PRI ALT
init ? x:
SKIP
TRUE & SKIP
int ? dx:
SEQ
out ? dx:
--- Input in parallel
out ! x:
--- with output
x := x + (dx * t):

The one mode estimator and controller for the flexible robot arm as illustrated in fig. 8 may be converted to consist of the basic building blocks mentioned above. The resulting block diagram is illustrated in fig. 10. The sign of some of the GAIN blocks has been changed to standardize on one SUMMER block. Note that the insertion of the SPLITTER blocks is necessary because the Occam channels are point-to-point connections. Installation of a SPLITTER block looks like a solder joint.

After the definition of the basic building blocks, the complete calculation process for one mode looks like the following Occam process. The external channels [13] and the internal channels [27] are not mentioned in the block diagram to avoid cluttering.
The overall conclusion of the experience gained over a number of projects is that the application of transputers allows the design of high-performance control systems. For complex systems the layered structure is the best approach to realize the parallel architecture. However, ad-hoc parallelism, as used in the control of the flexible robot arm, will give the best performance, because it minimizes the inter-transputer communication.

The basic building block technique together with the recognition of dedicated parallelism quickly results in an error-free design. For the design of more "general purpose" software for higher levels of the controller, the automatic task allocation program still has to be refined further.

The optimum scheduler for real-time systems is the deadline driven scheduler. It is recommended that this scheduler be implemented in future processors. A sub-optimal solution has been presented, where the timer process of the processor is used to guard the timing sequence.

The controller for the flexible robot arm has been implemented on a transputer with its built-in scheduler may at present be considered the best state-of-the-art implementation of a parallel building block for the realization of control systems.

7 RESULTS CONCLUSIONS AND SUGGESTIONS

The controller for the flexible robot arm has been implemented on a number of different processors. The results varied as follows:

- Z-80 + co-processor: 16 Hz
- 10 MHz AT + co-processor: 116 Hz
- 4 T800 transputers: 1000 Hz
- 4 T414 transputers: 4500 Hz
- 4 T800 transputers: 1000 Hz
- 10 MHz AT + co-processor: 116 Hz

With these instructions the scheduler of the transputer can go to work and schedule all these parallel processes for execution. The advantage of this method is the speed of the development of the control program.

The above program is for the control of one mode. Therefore this program is assigned to one dedicated transputer for every mode that needs to be controlled. There are, however, some remarks to be made about this design method. Because a large number of building blocks are interconnected, there is a danger of deadlock. It is a property of so-called I/O-parallel blocks, that if combined into networks, they will exhibit deadlock free properties. An I/O-parallel block executes the input and output in parallel and not sequentially. The correct use of a few I/O-parallel blocks in a network will result in a deadlock-free behavior of the network. For that reason the integrators in fig. 10 are I/O-parallel as indicated in the corresponding Occam program, resulting in a deadlock-free controller.

8. REFERENCES

Bakkers, A.W.P. (1989) editor of: Applying transputer based parallel machines. Proceedings of the 10th OCCAM User Group Technical Meeting, 3-5 April 1989, Enschede, Netherlands.

Bakkers, A.W.P. (1987). Application of parallel processing to robotics control. Proceedings of the First International Conference on Robot Technologies, TECHRO'88, September 19-26, 1988, Dijon, Burgas, Bulgaria.

Bakkers, A.W.P., R.M.A. van Rooij and L. James. (1987). Design of a real-time operating system (RTOS) for robot control. Proceedings of the 7th OCCAM User Group, Grenoble France, pp 318-327.

Bakkers, A.W.P. and W.L.A. Verhoefen, (1986). A Robot Control Algorithm Implementation in Transputers. Intelligent Autonomous Systems Conference, Amsterdam, pp 118-122.

Fukuda, T and Y. Kuribayashi. (1983). Precise control of flexible arms with reliable systems. Proc. Int. Conf. on advanced robotics, Japan Ind. Robot Assoc., Tokyo, Japan Vol I pp 237-244.

Hilhorst, R. (1987). Parallelisation of Computational Algorithms for a Transputer Network. Proceedings of the 7th OCCAM User Group Technical Meeting, Grenoble France, pp 420-424.

Horns, C.A.R. (1978). Communicating sequential processes "CSP". CACM, vol. 21, No.8, 666-677.

JSME. (1989). The International Conference on Advanced Mechantronics. May 21-24 Tokyo, Japan, The Japan Society of Mechanical Engineers.

Kruise, L. J., van Amerongen, P. L. Ploegh and M.J.L. Tiernego, (1986). Modelling and control of a flexible robot link. Proc. IUTAM/FAC Symp. dynamics of controlled mechanical systems. Liu, C.L. and J.W. Layland, (1973). Scheduling Algorithms for Multiprogramming in a Hard Real Time Environment, JACM, vol 20(1), 46-61.

Stankovic, John A., Misconceptions About Real-Time Computing A Serious Problem for the Next-Generation Systems, IEEE Computer. October 1988, 16-19.

Stavenaulter, A.C.J., G. ter Reehorst, and A.W.P. Bakkers. (1987). Algorithm Implementation in Transputers. Intelligent Autonomous Systems Conference, Grenoble France, pp 420-424.

Welsh, P. (1987). Managing hard real-time demands on transputers. Proceedings of the 7th OCCAM User Group, Grenoble France, pp 135-145.

Wijbrans, K.C.J., J. Meijer, and A.W.P. Bakkers. (1989). Real-time Sampling Sub System for the PCAT, to be published in Journal A, vol 30, nr 3.