DESIGN OF AN ADAPTABLE SUPERCOMPUTER
BASED ON TRANSPUTERS

BHASKARA BABA ARUN KUMAR

DR. ANIRBAN BASU

ACKNOWLEDGEMENTS

First of all, I thank my guide Dr. Anirban Basu, who exposed me to the Field of Parallel Processing, for his constant inspiration and guidance. I would like to express my thanks to Mr. Ramakrishna Murthy, Mr. Srikanth, Mr. Nagaraj Rao and Mr. Subramaniam for the invaluable discussions - in which they took part. I want to express my gratitude to the typist Sri S.K. Seth, and to the Duplicating Operator Sri Dilip Chatterjee and the staff of Dean's Office and Binding section for their co-operation.

CONTENTS

1. INTRODUCTION

Mission critical military applications which execute real - time applications such as detecting, tracking and destroying targets, on-board signal and image processing have formidable throughput requirements which can be met only by parallel processing systems with enormous power. Many parallel processing architectures such as Vector-Processor, Array-Processor, Pipelined Processor, Multicomputer, Multiprocessor, Data-flow-Processor have been developed. However, these systems yield performance gain only when the algorithm being executed matches the architecture of the underlying machine. Hence, none of the above architectures are universally suitable for all applications.

When a complex algorithm is examined closely, it is seen that the various tasks require different types of computations. For example, one task may be best computed by a Multicomputer system, the next task may be best computed by an Array-Processor and so on. It seems logical then to assign a seperate dedicated subsystem (with an architecture which matches the task's processing requirements) to each task. Since the whole algorithm is to be executed in the minimal time, this suggests having coresident in the supercomputing system, all these different types of architectures. However, the major drawback of such a solution is that there is an increase in system complexity and hardware resource underutilisation: A subsystem is engaged in the computation of

contd. 2/-

one task only - and during the execution of the remaining tasks it is idle. A better solution will, therefore, be to have a system capable of reconfiguration to form sybsystems with different architectures such as Array - Processor, Pipelined - Processor, Multicomputer etc. .. depending on the type of computation encountered in the algorithm. A parallel processing system with such reconfiguration capabilities is referred to as an "ADAPTABLE SUPERCOMPUTER".

An important issue in the design of such a system is finding a commercially available universal module from which an Adaptable super computer can be built. Transputers [whose salient features are given in Appendix-1] which have been recently developed by INMOS, may be used for this purpose.

This dissertation discusses the design of an Adaptable supercomputer which can reconfigure into an Array - Processor, a Pipelined - Processor or a Multicomputer depending on the application program. The design is based on Transputers. A reconfiguration methodology is also described, which will enable the system to switch from one architectural state to another depending on the requirements. The reconfiguration is handled by the operating system and the compiler which after analysis of an application program finds the optimal hardware configuration-needed for executing it.

contd. 3/-

In this dissertation, chapter 2 discusses the hardware organisation of the proposed Adaptable Supercomputer. Chapter 3 gives the procedures for the reconfiguration of various architectures. Chapter 4 gives the Reconfiguration Methodology. The scope for further work is given in chapter 5, alongwith the conclusions.

2. HARDWARE ORGANISATION

A number of systems have been developed, which reconfigure by changing the interconnection between different functional units. ILLIAC IV [2], SOLOMON I [3] and PEPE [4] are examples of reconfigurable array systems, whereas CRAY-1 [5] is an example of a reconfigurable pipeline system. But these systems cannot be regarded as Adaptable Systems, since they do not have the capability of forming systems with different architectures.

DCA group [6] is an example of an Adaptable System. This was developed by SI Kartashev and SP Kartashev. In this system the main adaptation is Bit-Size Adaptation. The system is assembled using LSI modules known as Dynamic Computer group. (DC group). Each group contains n computing elements, n-1 connecting units and a monitor. Each computing element includes a processing element, a memory element and an I/O element. All processing elements are identical and process h - bit words. Any set of k computing elements may be connected through k-1 connecting units - to form a single computer which handles h.k - bit words. So the n computing elements may assume 2^{n-1} architectural states each differing from others in the number and word-length of independently-operating computers. The main stress in the above system was given on Bit-Size adaptation. Moreover, the system uses LSI modules which are not commercially available.

contd. 5/-

The design proposed here, though based on the principles of the above mentioned system, does not have those limitations, since it is designed using commercially available VLSI chips, namely Transputers and has more powerful reconfiguration capabilities.

2.1 Transputer as a basic processing Element.

The first design problem of any system is the hardware organisation : the choice of basic processing elements, the interconnection between the processing elements. This design is done as an extension to the Transputer based Adaptable Pipeline [7]. There is a main host controlling all the processing elements (PEs). All PEs are identical. Transputer, whose details are given in Appendix-1 is used as the basic processing element because

-It is a commercially available VLSI chip.
The execution is very fast.
-It has in-built communication facilities using the Links. These links are for interconnection between the various PEs.
-It has a local memory and the memory can be extended. For T414 there is a 2k bytes local memory and it can address upto 4G bytes. This helps in providing a local control unit, if necessary, in the PEs.
-It has a processor having full arthmetic, floating point logic operation capabilities. Each Transputer can, thus, behave as a full-fledged independent computer in multi-computer mode.

contd. 6/-

-Ease of implementation of concurrent processes.
Transputer implements OCCAM directly. [8] [9] [10].
OCCAM is a high-level concurrent language. It is a language based on the concept of parallel, in addition to sequential, execution and to provide automatic -communication and synchronisation between concurrent processes. Thus each PE can have a concurrent execution in a multiprocessor environment.

- availability of synchronised communication. Transputer implements OCCAM. Transputer hardware links are directly mapped onto OCCAM software channels. These channels communicate synchronously. Additional synchronisation primitives are, hence, obviated in the execution of synchronised parallel algorithms. Problems such as Newton's iteration method and partial differential Equations can be solved by synchronised parallel algorithms. [5].

2.2 Hardware Structure.

The basic organisation of the system is shown in Fig. 1. There is a sequence of n Transputers T_0, T_1, ... T_{n-1}. ($n = 2^m$ for some $m \geq 0$. n is chosen so as to have a Baseline network as a routing network). All the Transputers are connected

contd. 7/-

to Main Host, Co, and also to the main routing network. Two adjacent Transputers T_i, T_{i+1} $(0 \leq i < {}^{n-1})$ are connected via a pipe line-switch (P. Switch) P_i. End-around connection i.e., connection between T_{n-1} and T_o is provided via P-switch P_{n-1}. In one mode P-switch P_i allows the connection between the Transputers T_i and T_{i+1} and in the other mode it connects Transputer T_i to the secondary routing network.

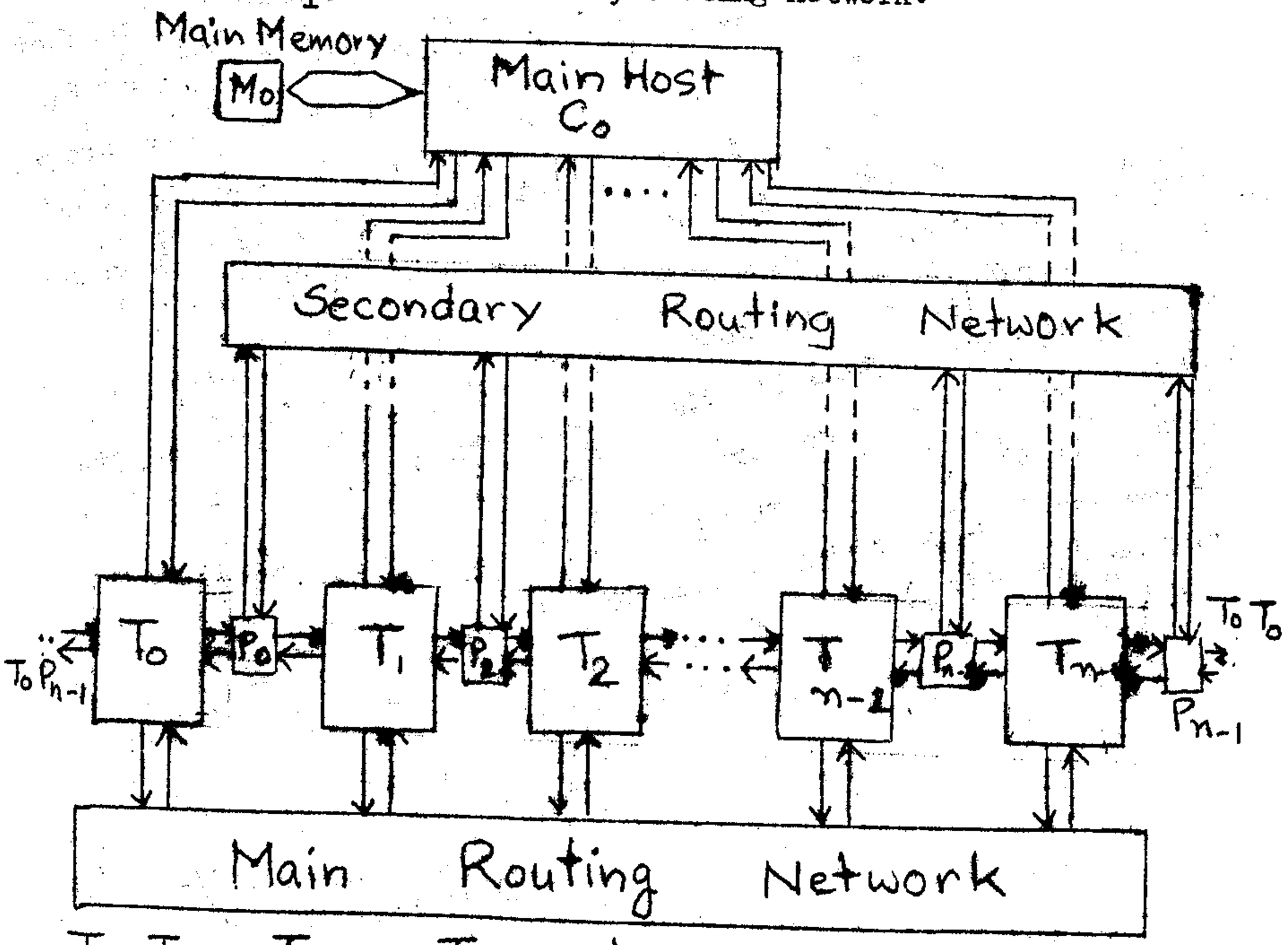


Fig. 1

Hardware Organisation.

contd. ...8/-

2.3 Main Host:

Main Host is the central control unit. This is the main computer C_o, with its own main memory M_o. The programs are compiled in C_o. The compiler is assumed to generate the code with architectural switch instructions, wherever necessary. The parallel streams in the code, if any, must be explicitly stated. This can be done using FORK, JOIN constructs [15] [5]. The parallel instruction streams, if any, are seperate Tasks of the same program. At any stage there may be one or more Tasks belonging to the same program. sequential mode, one task is there and it is run on one PE; in pipe-lined mode one task is there, but it is run on more than one PE; In Array mode there is one task, which is loaded in the Main-Host, running on more than one PE; In multi-computer mode there are n ($n \geq 1$) tasks running on n PEs. The operating system in the Main-Host assigns the PEs to the tasks. Main Host loads the tasks in the PEs, if necessary, and starts-up the tasks. (Loading the task implies-loading both the code and data of the task). The Main-Host communicates with the PEs via the links. The Link L_o of every Transputer (PE) is connected to the Main-Host.

contd. 9/-

2.4 Processing Elements (PEs):

As mentioned earlier, the basic processing elements are Transputers. Each Transputer has 4 links. They are used as follows:

-Link L_0 - For communication with the Main-Host.

-Link L_1 - For communication either with the next Transputer or any other Transputer other than the next one via secondary routing network depending on the setting of the associated P-switch.

-Link L_2 - For communication with any other Transputer via main routing network.

-Link L_3 - For communication with the previous Transputer via p-switch [redacted]

[redacted]

Once the Main-Host passes the control to the PE (i.e. starts up the task in the PE), the PE controls the execution till

i) the task is finished, or

ii) it encounters a run-time error or

iii) it needs a change to new architectural state.

Depending on the architectural state of the task running in the PE, the execution is controlled. This control code is stored in the Transputer (PE) memory permanently. This is the local control code.

contd. 10-

2.5 Routing Networks :

The main bottleneck in the design of multiprocessor system is the design of Interconnection Network. A shared-bus Interconnection is the least complex, but it does not allow more than one processor at a time to access a shared memory. A large number of processors means a long wait for the bus. On the other hand, a crossbar supports all possible distinct connections between the processors and memories simultaneously. However, the cost of such a network is prohibitively high. [14]. The cost and performance of Multistage Interconnection Networks and Multiple-Bus Networks hit a reasonable balance between those of shared-bus and crossbar. An n x n Multistage Interconnection Network connects n input elements to n output elements. For n, a power of two, it employs $\log_2 n$ stages of 2 x 2 switches with n/2 switches per stage. Each switch has two inputs and two outputs. The connection between an input and output depends on control bit, c, provided by the input. When c=0, the input is connected to the upper output; when c=1, it is connected to the lower output [11]. Baseline Network is chosen as Routing Network in this design. In Baseline network the control is decentralized one, i.e. the processor requesting an interconnection generates the destination address and it is sent to the switches. Depending on this the local switches are set establishing the connections. The switching is done entirely locally without any global control. The network is packet-switched, each

contd. 11/-

packet consisting of 64 bits. (32 bits for address and 32 bits for data). The interconnection uses bidirectional switches, so that the communication protocols of the Transputer are carried out between two communicating PEs.

The simulation studies done on the interconnection networks have shown that the Baseline network is reasonably well achieving a balance between the cost and performance. [12]. One of the simulation studies done on the Transputer based Adaptable Pipeline system has shown that the blocking factor is less than 1%. [13] Hence, the Baseline network is chosen for the main and secondary routing networks.

2.6 P-swtiches :

If two adjacent Transputers are forming pipeline then there must be interconnection between them. In other modes, this interconnection is not always needed. The links involved in this interconnection may be used for communication with some other Transputer. So a switch called pipeline switch or P-switch is introduced between the connection of two adjacent Transputers. A switch P_i is introduced between two adjacent Transputers T_i and T_{i+1} ($0 \leq i < n-1$). P_{n-1} is introduced between T_{n-1} and T_o Transputers. In total, n p-switches are used for a system having n PEs.

contd. 12/-

The p-switch P_i, can be either in P-mode or G-mode. In P-mode it establishes connection between Transputers T_i and T_{i+1} (pipeline mode). In G-mode it connects the Transputer T_i to the secondary routing network, for communication with some other Transputer T_j $(i \neq j)$. (General Mode). Though, the switch introduces a small delay, it enhances the interconnection capacity, increasing the performance gain. The PE T_i, in general, sets the switching mode of P-switch, P_i, depending on the architectural state to which it belongs. The PE T_{i+1} sets the switching mode of P_i only when it is not in pipeline mode and it wants to communicate with T_i.

contd. 13/-

3. ADAPTATION TO VARIOUS ARCHITECTURES

The system can have any one of the following architectures:

1. Sequential Processor
2. Pipelined Processor
3. Array Processor
4. Multicomputer
5. Mixed Mode, i.e. coresidence of the above types of architectures.

This chapter discusses the adaptation of the system to various types of architectures.

The Main Host first analyses the given program and then produces the suitable code, which includes Architectural Switch Instructions whereever necessary. An Architectural Switch instruction is a special type of instruction of the form $S \dashrightarrow S^*$, where S is the present architectural state and S^* is the new architectural state. [This is described in more detail in chapter 4]. It is assumed that the system possesses the compiler necessary for such a code generation. The compiler partitions the program into executable tasks on the system. These tasks include the Architectural Switch Instructions, wherever necessary. The necessary hardware resource requirements are also included in this instruction. First, the system is described assuming that only one program, which does not

contd. 14/-

need a mixed architecture, can be executed. Then, the description is given assuming that a number of such programs can be executed. The modifications necessary for executing the programs requiring mixed architecture is given at the end. In all the cases the Main Host assigns the PEs to the tasks of a program. It then loads the tasks (both code and data) in the PEs. Once the loading is over the control is transferred to the individual PEs to execute the tasks that are loaded.

3.1 PROCESSING ELEMENTS :

Each PE executes one task. The following information is stored in the PEs to help in the reconfiguration and the execution of the tasks. Three fixed locations are assigned in every PE for this purpose. These locations are in the on-chip memory of the Transputer. For universality these locations are called registers.

1. Type Register, having the Type Code T_i
2. State Register, having the State Code S_i
3. Position Register, having the Position Code P_i

The TYPE CODE T_i specifies the architectural state of the task being executed in the PE. Since there are only four states, this needs only two bits. (If mixed architecture is also there then it needs three bits).

contd. 15/-

Type Code	Architectural State
00	Sequential mode
01	Pipelined mode
10	Array mode
11	Multicomputer

Depending on this code the execution control of the task in the PE will change. Even when a program is in a mixed architectural state the given task can be associated only to a single architectural type. For example, let the present architectural state of the system be in a mixed architectural mode having both pipelined mode and multicomputer mode. Then any task can either be in pipelined mode or in multicomputer mode. The control program in the PE will be similar to

```
IF TYPE CODE = 00 THEN

    sequential execution from the code starting point
ELSE IF TYPE CODE = 01 THEN

    pipeline execution of the loaded task
  ELSE IF TYPE CODE = 1o THEN

    array processor mode : wait for the instruction from
    the Main Host
.
    ELSE IF TYPE CODE = 11 THEN

    multicomputer execution of the task ; load the
    memory address translation program.
```

contd. 16/-

The STATE CODE S_i in a PE gives the information of all the various PEs involved in the execution of the tasks of the program, to which the task being executed in the PE belongs. One bit is given to each PE. If there are n PEs in the system then n bits are needed. A 1 in the i th bit indicates that the i th PE is executing a task belonging to the same program. This code is used for communication between different concurrently running tasks.

The above two codes are loaded in the PEs by the Main Host whenever there is a change in the architectural state.

The POSITION CODE P_i gives the number of the PE. This is a unique number given permanently to each of the PEs. If there are n PEs then this needs a minimum of $\log_2 n$ bits. However, in this design this is having n bits, reducing the complexity of computing the processor number. All bits are 0, except the i th bit in the Position Code of the i th PE. This is used in communication with the Main Host in specifying the information regarding the PE that is communicating. This is also useful in the inter-PE communication, if some protection has to be incorporated. These details are not dealt in this design.

3.2 ADAPTATION TO SEQUENTIAL PROCESSOR :

The Main Host loads the Type Code 00 in the type register. In this mode the Status Code is not loaded. The Task Code is loaded in to a fixed location. Then the control is passed on to the PE.

contd. 17/-

The PE starts execution of the code from a fixed location. In this mode the P-switch setting can be anything, since there is no need for communication with the other PEs. The P-switch is set to G-mode.

3.3 ADAPTATION TO PIPELINED MODE :

The main Host loads the Type Code 01 in the Type Register and the Status Code is loaded in the Status Register of all the PEs involved in this pipeline. The Main Host assigns only adjacent PEs to the execution of a pipelined task. The task code and data are loaded in the PE, which is the first stage of the pipeline. All the PEs, except the last PE set their corresponding P-switch to P-mode, thus establishing a linear array of PEs forming the pipeline. The instructions and data flow from the first stage to the last stage. The instructions have the following format.

Opcode	No. of stages remaining	Task number

The opcode and the no. of stages remaining specifies the operation to be performed. In this each pipeline stage is capable of performing any operation. Each PE, except the first PE, in the pipeline waits for the instruction and data from the previous stage. The first PE fetches the instructions from its own memory, where the task is loaded. Every PE, except the last PE, sends the instruction and data to the next stage. The last PE sends the results

contd. 18/-

to the first stage via either the main routing network or the secondary routing network. The synchronised communication protocol of the Transputers assures a proper synchronisation of all the stages in the pipeline.

The first stage cannot send instructions continuously. This is because execution of certain instructions may require traversing the entire length of the pipeline many times and time of visit of their visit to the first stage may coincide with the time of initiation of a new instruction. To avoid such collisions, the time of initiation of instructions has to be determined from the analysis of Reservation Table [16]. The conditional branch instruction problem may be solved by having two pipelines simultaneously, when there is a provision for mixed mode. This feature is not included in this design.

3.4 ADAPTATION TO ARRAY PROCESSOR MODE :

. The main host loads the Type Code 10 in the Type Register and the corresponding Status Code in the Status Register of all the PEs in this mode. In this mode the Main Host acts as the supervisor. The corresponding task is run in the Main Host. Only data elements are loaded in the PEs. For access to various substructures of the matrix the skewed storage may be used [17]. The P-switch is set to P-mode to provide an easy access to the adjacent elements in the array. The Main Host broadcasts the instructions to all the PEs. The PEs act just as computing elements.

contd. 19/-

The masking of certain PEs may also be done if a Mask instruction is provided. The Mask instruction broadcasts the instruction to all the corresponding PEs. The operand should have n bits for a system having n PEs. The i th bit will be 1 if the operation must be masked in the i th PE. This operand is ANDed with the Position Register in the PE. If the result is not 0 then the operation is masked else it is executed. This will be stored in the Mask Register of the PEs. This is changed whenever there is a Mask Instruction and all the Mask Registers in different PEs must be set to their corresponding values, before the execution of a task in Array-processor mode.

3.5 ADAPTATION TO MULTICOMPUTER MODE :

In this mode all the parallel tasks are loaded into different PEs (which may not be adjacent). The Main Host loads the corresponding code and data in the PEs. The Type Code 11 and the Status Code are loaded in all the PEs. The Main Host starts up all the tasks. All the tasks proceed in parallel. Only one of the tasks will be having the architectural switch instruction (see chapter 4). Except this task, when the task is over the PE will be free and the PE is assigned to some other task, if needed. For the task having the architectural switch instruction, when the architectural switch instruction is encountered the task is kept in waiting state till all the other tasks have completed their execution. Then the necessary architectural switch is performed. Though this waiting time of the

contd. 20/-

particular PE may be utilised for running some other task, using multiprogramming, it increases the complexity of the system and therefore not considered here.

The memory is paged memory. Each task will have one or more pages of memory. All the local variables of the task can be found in those pages. There is a strong empirical evidence that most programs exhibit 'local' behaviour in the sense that memory references tend to be concentrated in particular regions at any given time and the region of activity changes as the program enters one region from another [15] As a program enters a region it accesses a particular part of data and this data is placed in that PE's local memory. Therefore most of the data is fetched from its local memory.

All tasks are assumed to have equal number of pages. The tasks are assumed to be loaded such that the consecutive pages are stored in the PEs whose Position Codes are in the increasing order. This makes the address translation simple. The address translation is of two stages. First stage is to find the page number and the next stage is to find the PE in whose local memory that particular page is. For example let there be n pages per task. If the page referenced is p then that page is in the local memory of task k, where k = p div n (integer division). The PE which is running the k th task can be found out from the Status Code. It is the bit position of the k th 1 in the Status Code. If the above convention is not followed then the page map table giving all the details of the tasks and the PEs assigned to them must be maintained. This table must be referred whenever there is a memory reference.

contd. ...21/-

Generally, all the P-switches are set to G-mode. If the memory reference is mapped to an adjacent PE, then the P-switch is set in P-mode and the data element is fetched. If the memory reference is mapped to some other PE then the communication is done via either the main routing network or the secondary routing network.

3.6 ADAPTATION TO MIXED MODE :

As mentioned earlier, the Type Code will be loaded. Depending on the type of communication allowed, the Status Information must be stored. If there is no communication between the tasks in different architectural modes then the above mentioned designs for different modes holds good in this case also. If such restriction is removed then the information regarding all related PEs and their architectural modes must be stored in each PE. This case is encountered very rarely and this is not included in this design.

As in the case of the Multicomputer mode, only one of the tasks will have the architectural switch instruction and that task is kept in waiting till all the other concurrent tasks, executing in different architectural modes, finish their execution.

contd. 22/-

4. RECONFIGURATION METHODOLOGY

This chapter gives the form in which the compiler must generate the code. It also gives the reconfiguration methodology.

In general, any architectural reconfiguration of a set of reconfigurable hardware resources must be supervised by the monitor interconnected with these resources, since this monitor must find the moment of time when such reconfiguration is possible and should resolve the conflicts among the programs with conflicting reconfiguration requests [1]. The Main Host is the monitor in this design.

The compiler and operating system of an Adaptable system are specialised ones in the sense that i) the compiler must be able to provide the code in a suitable form with architectural switch instructions in the appropriate place, ii) the operating system must be able to execute the architectural switch instruction and reconfigure its resources into the architetural mode as required by the program.

4.1 ARCHITECTURAL SWITCH INSTRUCTION (ARS INSTRUCTION) :

The basic steps in architectural reconfiguration are

1. setting up of new control instructions in each of the computing modules of the architectural state to be established

2. establishment of new interconnections in the interconnection networks used

3. start up of programs assigned to a new architectural state.

contd. 23/-

The architectural reconfiguration is performed by a special type of instruction $S \dashrightarrow S^*$, where S is the present architectural state in which the program is in and S^* is the next architectural state. This instruction belongs to the program that requests reconfiguration. The operand of such an instruction will be the number of PEs the new state needs. For the sequential processor it is 1. For a P stage pipeline it is P ; for an n element array it is n ; for a multicomputer with m parallel data streams it is m.

4.2 REQUIREMENTS OF A COMPILER FOR THE SYSTEM :

The compiler must be able to partition the program into various tasks and then determine the most efficient architectural modes in which the tasks must be run. The compiler must possess the capability of finding the parallel streams. The compiler must produce a code with necessary synchronisation and architectural switch instructions. For the tasks running as either a sequential process, pipelined process or an array process the code is simple one. At the end of each such task there will be either a STOP instruction or an ARS instruction. In the case of pipelined task the instructions are as specified in section 3.3. The code generated for the parallel tasks to be executed in multicomputer mode is different. The code can be generated using the FORK and JOIN constructs. All but one task should contain a STOP instruction after JOIN instruction. Only one task will have either a STOP or an ARS instruction after the JOIN instruction. This is the task

contd. 24/-

that will continue the execution. When mixed mode is allowed in the system then the same method followed for the multicomputer mode is followed except that in this case each mode is considered as a seperate task and all such tasks are running in parallel. The same FORK and JOIN constructs may be used in this case also.

4.3 MAIN HOST :

As mentioned earlier we take up three cases.

Case a) When only one program (with no mixed mode) is allowed to run

Case b) When more than one program (with no mixed mode in any program) are allowed to run

Case c) When mixed mode is allowed.

The Main Host keeps the following information :

- Idle List : The state of all the PEs in the system. A PE may be either in execute mode or in idle mode.
- Program List : The details of all the programs being executed on the system.
- Outstanding Request List : The details of the programs that are kept in waiting state.

contd. 25/-

Idle List gives the list of all idle PEs. For a system having n PEs, n bits are used for this purpose. If i th bit is 1 then the i th PE is in execute mode else it is in idle mode. Whenever some PE is assigned to execute a particular task, the corresponding bit in the Idle List is made 1. When the task being executed in a PE is over or it is interrupted, the corresponding bit is set to 0.

For example, let there be two programs running on a 8-PE system. First program running in pipelined mode on PEs 0,1, and 2 and the second running in multicomputer mode on PEs 3,5, and 7. Then the Idle List will be having 8 bits and it will be 1111 0101.

Program List, keeps track of all the information regarding the programs (along with their numbers) that are being executed in the system. In case a) this is not needed as there is only one program. In case b) it needs n+2 bits for storing the information of a single job. (n bits for n PEs and 2 bits to specify the architectural mode, since mixed mode is not permitted). For the above example the Program List will be

1. program 1	1110 0000	01
2. program 2	0001 0101	11

In case c) for every program a seperate list for each type of architectural mode is stored. If there are n PEs then it needs 4*n bits as there are 4 modes. For example, let a program be running as follows.

contd. 26/-

multicomputer tasks on PEs 0 and 2

pipelined task on PEs 4 and 5 then

the entry for program 1 will be

0000 0000 0000 1100 0000 0000 1010 0000

the first sequence for sequential mode, second sequence for pipelined mode, third sequence for array mode, and the last sequence for multicomputer mode. To get the details of all the PEs involved in this program all the four sequences are ANDed. In the above example the result will be 1010 1100.

Outstanding Request List gives the status of all the programs whose request for an architectural switch has not yet been met. This is there in cases b) and c) only. This list contains the program number, the outstanding architectural switch instruction, which includes the previous architectural state and the next architectural state requested.

4.4 RECONFIGURATION :

The Main Host upon receiving the architectural switch instruction perfoms the following operations.

-deallocation of resources

-assignment of new resources

-loading of the corresponding tasks and control information

-starting of execution in the new architectural state.

contd. 27/-

Since all the PEs in the system are identical the task can be loaded in any of the PE. Depending on the Position Code of the PE, from which an architectural switch request has come, and using the Program List the program number can be found out.

4.4 (a) Switch to Sequential Mode :

When a task needs a switch to sequential processor mode, only one PE is needed. The previous state will be having more than one PEs. Out of these PEs one is selected arbitrarily and the others are deallocated. Appropriate changes are made in the Program List of the program to which the task belongs. The data is swapped back into the main memory and the relevant data and the task code is loaded in the PE along with other control information. The Main Host then starts up the task in that PE.

4.4 (b) Switch to Pipelined mode :

The main task of the Host will be finding the required number of contiguous PEs. The choice is made such that maximum number of PEs of the previous state are included in this new state. First the maximum number of contiguous PEs are found out from the previous state and check is made to see whether that or the extension of that will meet the requirements. If it is not possible then another sequence is chosen and it is checked. The same procedure is repeated until all the sequences are exhausted or a choice has been made. If all the choices are exhausted, a new list is formed from the Idle List, after making the necessary deallocation of the PEs. If no choice can be made then it is placed in the Outstanding Request List.

contd. 28/-

4.4 (c) Switch to Array-mode :

In the Array Mode the problem is just to find the required number of PEs, not necessarily contiguous, as per the requirements of the task. However, a check has to be made whether the request can be satisfied using the PEs involved in the previous architectural state. This helps in reduced number of memory swaps. If this is not possible then a try is made to allocate PEs from the Idle List after

- swapping back all the memory from the PEs in the previous architectural state
- updating the Idle List. (The PEs involved in the previous state are free now)

If the requirement cannot be met then that switch instruction is kept in the Outstanding Request List.

4.4 (d) Switch to Multiccmputer Mode :

In this case also the procedure is similar to the one used for the Array-Processor mode. The only difference is that the Main Host loads the tasks in the PEs and in the Array-Processor mode the task is executed in the Main Host itself, since the Main Host can broadcast instructions to all the PEs in the system.

If fixed mode is permitted in the program, then also the assignment problem is the same one as mentioned above. The synchronisation of various modes is taken care of in the code generation stage itself.

contd. 29/-

In general, the assignment and allocation problem is an important problem in the design of the Main Host. In the above mentioned procedures there is no priority for any program. This can also be incorporated. The assignment strategy is a simple one. The general principle followed is to meet the requirement from the PEs in the previous state of the task, as far as possible. This reduces the memory swaps required. The memory swaps can still be reduced if the compiler is able to decide the common tasks of the previous state and the next state. This information may be sent as an operand of the Architectural Switch instruction. This helps in reducing the memory swaps. The PE allocation strategy may be decided beforehand if deterministic model of the programs is developed. This is done in the DCA [6]. But deterministic modelling does not hold good in all cases. This can be dealt with by using Stochastic Modelling [15].

contd. 30/-

5. CONCLUSIONS

The Adaptable Supercomputer is designed as an extension to the Adaptable Pipeline System designed already. The Reconfiguration Methodology is described for the given system. The Architectural Switch instruction and its execution are also discussed.

5.1 Scope for further work :

In the design discussed, the compiler part is not covered. A suitable compiler generating the code with appropriate architectural switch instructions can be developed. The assignment strategy can be improved to reduce the swap time and to allow for priority among the programs running. The assignment problem is the same for the Multiprocessor and Multicomputer systems. The hardware details of the routing network are not given. A suitable network can be designed which is compatible to the Transputer. The system may be extended to allow mixed mode to exist in a program.

contd. 31/-

APPENDIX - 1

TRANSPUTER DETAILS

The basic processing element used in the design are Transputers, a VLSI chip manufactured by INMOS.

A Transputer is a VLSI chip. It is a microcomputer with its own local memory and links for connecting one Transputer to another Transputer. The processing capability may be general purpose, or may be optimised to a specific purpose. The on-chip memory may be extended off-chip by a suitable interface. The architecture of the Transputer is defined by reference to Occam. Occam provides the model of concurrency and communication for all the Transputer systems.

Occam is a block-structured concurrent language which is based on the ideas of concurrency and communication. It is designed as a systems implementation language for both single and multiprocessor system. An Occam program is a collection of Occam processes which can be executed sequentially or concurrently. The concurrency and sequence are explicitly defined by the user. If one process is running concurrently with another, then communication with that process is via Occam channels. This is a well defined communication mechanism which eliminates the shared variables. The channels map onto the hardware links between machines [10]. Concurrent processes communicate only by using channels and communication is synchronised.

contd. 32/-

If a channel is used for input in one process and output in another, communication takes place when both the inputting and outputting processes are ready. The processes then proceed and the value to be output is copied from the outputting process to the inputting process [18].

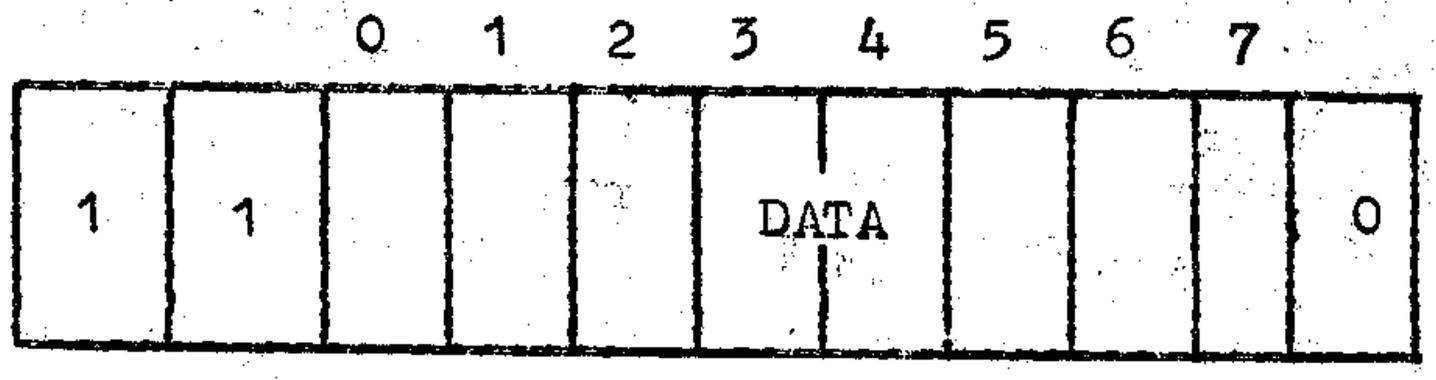
The Transputer chip has 4 links : Link 0, Link 1, Link 2, and Link 3. Each link provides two Occam channels, one in each direction. Communication via any link may occur concurrently with communication on all other links and with program execution. Synchronisation of processes at each end of link is automatic and requires no explicit programming.

Communication through a link involves a simple protocol, which provides the synchronised communication of Occam channels. Each message is transmitted as a sequence of single byte communications, requiring only the presence of single byte buffer in the receiving end to ensure that no information is lost [19]. Each byte is transmitted as a start bit followed by a one bit and followed by eight data bits followed by a stop bit. After transmitting a data byte the sender waits for the acknowledgement from the receiver; this consists of a start bit followed by a zero bit. The acknowledge signifies that the process was able to receive the acknowledged byte, and that the receiving link is ready to receive another byte. The sending process may proceed only after the acknowledge for the final byte of the message has been received.

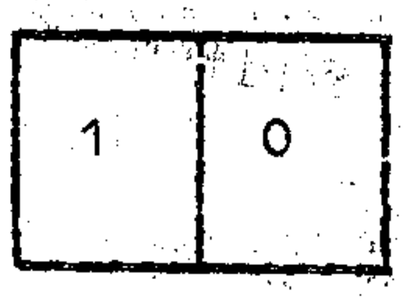
contd. 33/-

Link Protocol :

Data Packet



Acknowledge Packet



Data bytes and acknowledge are multiplexed down each signal line An acknowledge is transmitted as soon as reception of a data byte starts. Consequently, transmission may be continuous with no delays between data bytes.

* * *

APPENDIX - 2

REFERENCES

[1] Svetlana P. Kartashev and Steven I. Kartashev, "Adaptable Software for Dynamic Architectures", Computer, Vol. 19, No. 2, February 1986, pp 61 - 68.

[2] Barnes G., [illegible] Brown, Maso Keto, D. Kuck, et al, "The ILLIAC - IV computer", IEEE Transaction on Computers, Vol. C-17, No. 8, August 1968, pp 746 - 757.

[3] Slotnick, Daniel L., et al, "The Solomn Computer", Proceedings of 1962 Fall Joint Computer Conference, pp 97 - 107.

[4] Evenson, Alf J. and James L. Troy, "Introduction to the Architecture of a 288 - element PEPE", Proceedings of the 1973 Sagamore Computer Conference on Parallel Processing, New York, IEEE, 1973.

[5] Kai Hwang and Faye A. Briggs, Computer Architecture and Parallel Processing, Mc Graw Hill Book Co., 1985.

[6] Svetlana P. Kartashev and Steven I. Kartashev, "Multicomputer system with Dynamic Architecture", IEEE Transactions on Computers, Vol. C-28, No. 10, October 1979, pp 704 - 721.

[7] A Basu, "A Transputer based Adaptable Pipeline System", Proceedings of the Second International Conference on Supercomputers, Santa Clara, USA, May 1987.

[8] Dick Pountain, A Tutorial Introduction to Occam Programming, INMOS Ltd. Publications, March 1986.

contd. 35/-

[9] Richard Taylor and Pete Wilson, "Process - Oriented Language meets demands of Distributed Processing", Electronics, November 30,1982, pp 89 - 95.

[10] Stephen Brain, "Writing Parallel Programs in Occam', Electronic Product Design, 1984.

[11] Laxmi N. Bhuyan, 'Interconnection Networks for Parallel and Distributed Processing", Computer, Vol. 20, No. 6 June 1987, pp 9 - 12.

[12] George H. Barnes and Stephen F. Lundstrom, "Design and validation of a Computer Network for a Many Processor Multiprocessor system", Computer, Vol. 14, No. 12, December 1981, pp 31 - 41.

[13] Arun Kumar B.B., "Design and performance evaluation of an Interrconnection Network for and Adaptable Pipeline System", M.Tech. Project Work, ISI, Calcutta, 1988.

[14] Stone, Chen, Flynn, et al, Introduction to Computer Architecture, second edition, Science Research Associates Inc., USA, 1986.

[15] Ramamoorthy C.V. and Li H.F., "Pipeline Architecture", ACM Computing Surveys, Vol. 9, No. 1, March 1977, pp 61 - 102.

[16] Budnick P., Kuck D.J., "The Organisation and Use of Parallel Memories", IEEE Transactions on Computers, Vol. C-20, 1971, PP 1566 - 1569.

[17] INMOS, Transputer Reference Manual, September 1985.

[18] Colin Whitby and Stevens, 'Transputer", IEEE Transactions on Computers, Vol. C - 32, 1985, pp 292 - 300.

* *