Núcleo de Computação Eletrônica

The Multiplus/Mulplix Project: Current Status and Perspectives

Júlio Salek Aude

NCE - 03/96
novembro

Relatório Técnico

Universidade Federal do Rio de Janeiro

THE MULTIPLUS/MULPLIX PROJECT: CURRENT STATUS AND PERSPECTIVES

Júlio Salek Aude

NCE and IM/UFRJ
PO Box 2324
Rio de Janeiro - RJ - 20001-970
Brazil
e-mail: salek@nce.ufrj.br

ABSTRACT

The MULTIPLUS/MULPLIX project aims at the development of a modular distributed shared-memory parallel architecture able to support up to 1024 processing elements based on SPARC microprocessors and at the implementation of MULPLIX, a Unix-like operating system which provides a suitable parallel programming environment for the MULTIPLUS architecture. The project includes research effort in five areas: parallel architectures, operating systems, CMOS IC design, parallel programming environments and parallel algorithms. This technical report firstly presents an overview of the MULTIPLUS architecture and describes in detail the current implementation of its four basic hardware modules: the Processing Element, the I/O Processor, the Multistage Inteconnection Network and the Network Interface. Secondly, the MULPLIX operating system definition is reviewed and the parallel programming primitives available within MULPLIX are presented. Following, developments in the area of CMOS IC designs for use within the MULTIPLUS architecture are described. The implementations of PVM and Pthreads parallel programming libraries within the MULPLIX system are also discussed. Finally the main results achieved with the parallelization of Simulated Annealing and Genetic algorithms are commented.

Ao meu saudoso pai, que me ensinou o fundamento básico da pesquisa científica: a perseverança.

1. INTRODUCTION

The MULTIPLUS project [Aude90, Aude91, Aude94, Aude95, Aude96] has been under development at NCE/UFRJ for some years now and has provided a nice and challenging framework for research work in several areas related to the world of High-Performance Computing: Parallel Architectures, Operating Systems, IC Design, Parallel Programming Environments and Parallel Algorithms.

The main objectives of the MULTIPLUS project include the development of a distributed shared-memory parallel architecture and the MULPLIX operating system. General aspects of the MULTIPLUS architecture have been discussed in previous papers [Bron90, Mesl90, Oliv90, Mesl92, Oliv92a, Bron93] as well as the main features of the MULPLIX operating system [Azev90, Azev93a]. This technical report aims at giving an up-to-date overview of the current state of development of the MULTIPLUS project as a whole.

Section 2 reviews the main features of the MULTIPLUS distributed shared memory parallel architecture. Section 3 presents the current implementation of the Processing Elements within the MULTIPLUS architecture. In Section 4, the implementation of the Multistage Interconnection Network and of its Interface to each MULTIPLUS cluster of processors is presented. Section 5 comments on the implementation of the I/O Processor and its control system. Section 6 discusses the design of VLSI circuits to be used within the MULTIPLUS architecture, including the development of the NCESPARC microprocessor. Section 7 describes the MULPLIX operating system and the parallel programming primitives which have been implemented within MULPLIX. The use of these primitives is illustrated through a very simple parallel application. Section 8 briefly describes the implementation of two parallel programming libraries within MULPLIX: M-PVM and Pthreads. In Section 9, some of the main results achieved with the parallelization of Simulated Annealing and Genetic Algorithms applied to the placement problem are summarized. Finally, in Section 10 the perspectives for the project development in the near future are presented.

2. THE MULTIPLUS ARCHITECTURE

MULTIPLUS is a distributed shared-memory high-performance computer designed to have a modular architecture which is able to support up to 1024 processing elements and 32 Gbytes of global memory address space. Figure 1 shows the MULTIPLUS basic architecture. Within MULTIPLUS, up to eight processing elements can be interconnected through a 64-bit double-bus system making up a cluster. Each bus follows a similar protocol to the one defined for the SPARC MBus, but is implemented as an asynchronous bus.

The MULTIPLUS architecture supports up to 128 clusters interconnected through an inverted n-cube multistage network. Through the addition of processing elements and clusters, the architecture can cover a broad spectrum of computing power, ranging from workstations to powerful parallel computers. With the adopted structure, the cost and delay introduced by the interconnection network is small or even non-

existent in the implementation of parallel computers with up to 64 processing elements. On the other hand, very large parallel computers can be built without the use of an extremely expensive or slow interconnection network.

The MULTIPLUS architecture can be classified as a Non-Uniform Memory Access (NUMA) architecture since a processing element access to memory can be performed in four different ways. The fastest memory access is a direct read operation on the local caches, which is performed within a processor cycle. The second fastest memory access is any read/write operation within the local bank of memory since, in principle, it does not require the use of the cluster bus system for its completion. The third fastest memory access is a write or a read access with cache failure to a memory position belonging to an external memory bank within the same cluster. In this case, the bus system must be used and the bus arbitration time is added to the access time. Lastly, there are the accesses generated by a processing element requesting information which is not in its local caches but is stored within a memory bank sitting on another cluster. In this case, the bus system of the source cluster , the multistage interconnection network and the bus system of the destination cluster need to be used for the access operation to be performed. Therefore, the arbitration times of both bus systems and the multistage interconnection newtork delay are added to the access time.

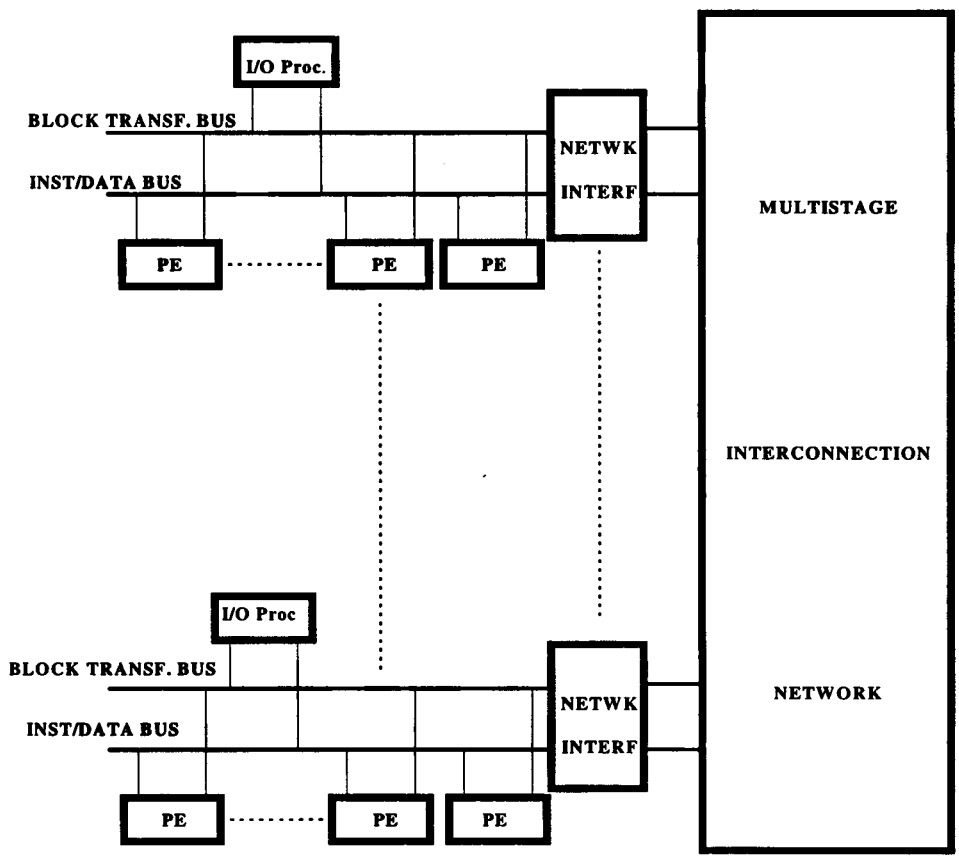

Figure 1: The MULTIPLUS Architecture

As shown in Figure 1, MULTIPLUS uses a distibuted I/O system architecture. It is possible to assign all processing elements within a cluster to a single I/O processor

which is responsible for dealing with all I/O requests to or from mass storage devices started by these processing elements.

Design decisions have been taken to simplify the problem of maintaining consistency among the private caches of the processing elements within the MULTIPLUS architecture [Mesl91]. The first one is to have in every cluster one bus dedicated to instruction and data access operations and the other one dedicated to block transfer operations which occur in I/O or in memory page migration or copy operations. Only the instruction/data bus needs to be "snooped" by the cache controller and, as a result, the cache consistency problem can be solved within a cluster with the methods usually adopted in bus-based systems. In addition, a software approach based on the works presented by Petersen and Li [Pete94] and by Kontothanassis and Scott [Kont95] has been adopted to keep cache consistency between clusters. Following the memory model based on the lazy release consistency approach [Kele92], any access to shared regions of memory must be preceded by a "lock" operation. This ensures that a single processor is accessing a particular critical region at any moment. Cache consistency is achieved with the help of the memory management hardware.

3. THE MULTIPLUS PROCESSING ELEMENT

The MULTIPLUS Processing Element is based on the use of SPARC processors. The first implementation of the processing element used the Cypress SPARC chipset and could support a 64-Kbyte cache and up to 32 Mbytes of memory belonging to the global address space. The most recent implementation of the Processing Element can have up to 2 SuperSPARC II modules running at 85 Mhz and supporting a 1 Mbyte Cache. This new Processing Element can support up to 256 Mbytes of memory. In addition to the SPARC processors and memory, the MULTIPLUS Processing Element includes: ROM, serial interface, interrupt registers and timers.

Figure 2 shows a block diagram of the current Processing Element architecture which is built around any SPARC MBus module. Only a single SPARC module is represented in Figure 2. The number of address lines followed by the number of data lines is annotated next to every bus. The cache controller works in write-through mode with invalidation of shared cache copies, which is a very simple approach and has proved to be as efficient as the write-back mode in simulation experiments carried out considering typical values for the data cache hit rate and the rate of write operations [Mesl92].

The control logic of the Processing Element is implemented with the use of four EPLDs. The first one performs the slave fuction in the Block Transfer Bus, arbitrates the use of the common bus for memory acces within the processing element and performs the DRAM control. The address decodification in the Block Transfer Bus is performed by another EPLD. In the control of the instruction/data section, two EPLDs are used. The first one performs address decoding and access control to the processing element registers and I/O devices. The second one performs the master and slave functions in the Instruction/Data Bus and the arbitration between requests issued by the Instruction/Data Bus and by the Processing Element Data Cache Controller.

Within the memory, a TAG bit is associated with each memory data block in order to indicate if a copy of this block may exist in another cache. The bit is set whenever the block is read by a different processing element sitting within the same cluster. It is reset whenever that block is rewritten by the local processing element. The importance of this bit is to reduce the need for broadcasting unecessary data access to the Instruction/Data Bus in order to maintain cache consistency. If the TAG bit is not set, the data access can be performed within the Processing Element and without the use of the Instruction/Data Bus.

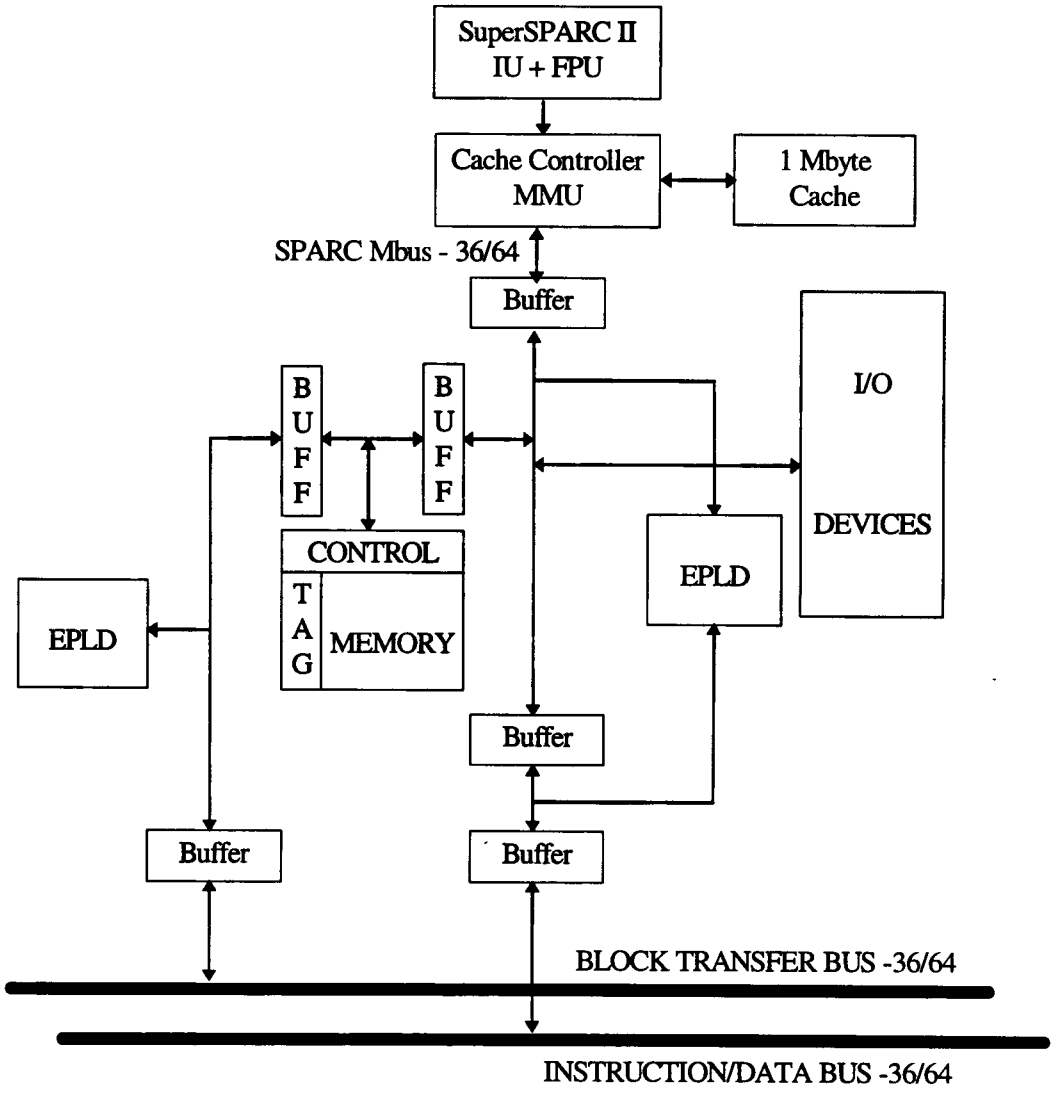

Figure 2: The Processing Element Architecture

4. THE MULTISTAGE INTERCONNECTION NETWORK

The MULTIPLUS multistage interconnection network is an inverted n-cube network consisting of 2x2 cross-bar switching elements [Bron91]. Its topology is shown in

Figure 3. Separate networks are used to interconnect the instruction/data and the block transfer busses in different clusters. The adopted network topology provides the MULTIPLUS architecture with two very desirable features: modularity and partitionability. Modularity enables the MULTIPLUS architecture to grow in numbers of clusters through a simple addition of extra switching elements to the network. No re-wiring of the interconnections between the elements already present in the network is required in such operations. The partitioning feature of the network provides the MULTIPLUS architecture with the possibility of supporting several independent or loosely-coupled groups of clusters. In fact, the network ensures that it is possible to choose groups of clusters such that the communication within a group does not interfere with the communication within any other group of clusters.

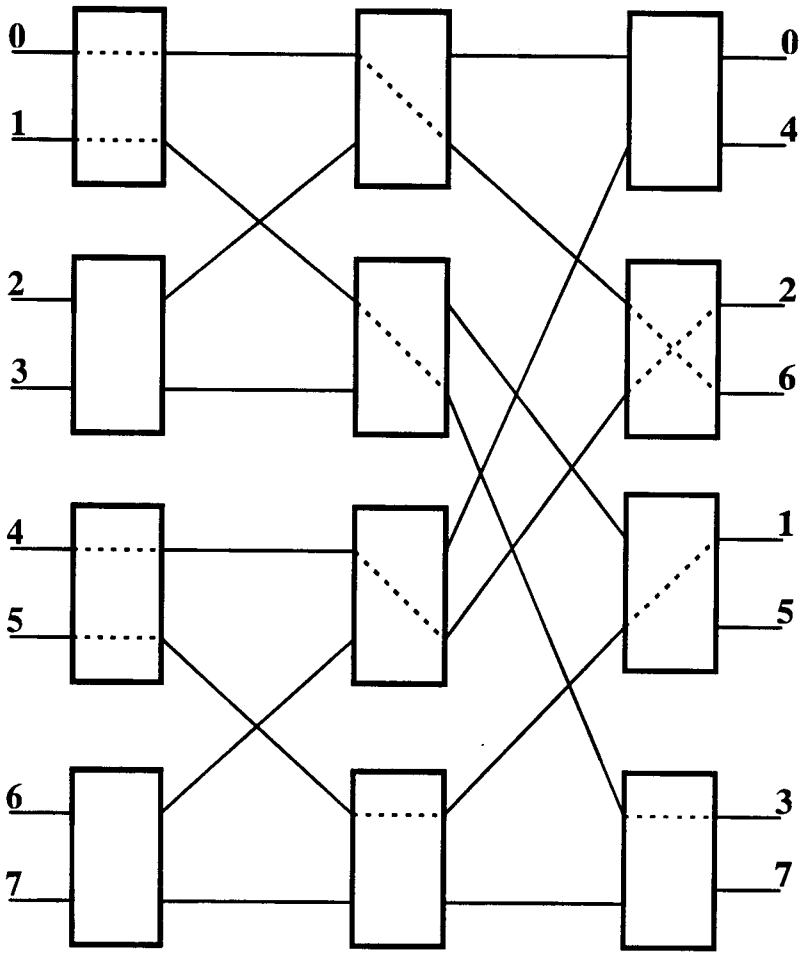

Figure 3: The Multistage Interconnection Network

The MULTIPLUS Multistage Interconnection Network can support up to 128 clusters. Each communication path between switching elements in the newtork is unidirectional and nine bits wide. The transmitted messages can have variable length up to a maximum of 128 bytes. Wormhole routing is used in the network and a single bit of the destination address field of the messages is examined by each stage of

switching elements to direct the message to the next stage.

Six message types are supported by the Multistage Inteconnection Network: Write, Read, Write Reply, Read Reply, DMA and DMA Reply. Every message can have only a single source and a single destination, therefore broadcast or multicast type messages are not currently handled by the network.

A message can be seen as a sequence of packets consisting of eight data bits and one parity bit. In general, a message has three basic sections: the header, the preamble and the data. The header is four byte long and contains information on the destination address, message size, message type and identification of the module that has generated the message within the source cluster. The preamble contains an image of the 64-bit address lines of the source cluster. It is only needed in Read, Write, DMA and DMA Reply messages.

Read and Write messages occur when a module within a cluster wants to access a memory position belonging to another cluster. The Write Reply message is used to tell the I/O Processor which has generated a block write operation that the last requested write operation has been completed. The Read Reply message returns the requested data to the processing element which had issued the corresponding Read message. A DMA message sets the Multistage Inteconnection Network to perform a block transfer of length up to 64 Kbytes from a region of memory within a given cluster to the local memory of the processing element which issued the DMA request. The DMA Reply message uses the Instruction Bus to transfer the requested data in blocks of 128 bytes between clusters. On completion of the DMA Reply operation, the Network Interface interrupts the processing element which issued the DMA request.

The architecture of the switching element of the Interconnection Network implements a 2x2 cross-bar switch with FIFO buffers assigned to each switch input. Its detailed design has been presented by Bronstein [Bron96]. Each switching element with the FIFO buffers has been implemented with a single EPLD.

The Network Interface interconnects the cluster bus systems to the Multistage Interconnection Network and also performs the functions of bus arbiter and bus reset generation. As shown in Figure 4, the Network Interface consists of two identical sections: one that deals with the Instruction/Data Bus and another which deals with the Blcok Transfer Bus. In addition, it has a DMA Controller which is programmed through the Instruction/Data Bus and performs data block transfers through the Block Transfer Bus. Within each section, the Network Interface consists of 8 modules: the bus interface module with a master and a slave section, the FIFO memory for messages to be transmitted, the message transmission module, a dual-port memory for received messages, the message reception module, registers, the bus arbiter and the logic for bus reset generation.

The implementation of the Network Interface has been carried out with 11 EPLDs, five for each section and one for the DMA Controller. The five EPLDS in each section perform the following functions: bus master; bus slave; message transmission

control; message reception control; storage of the status of the messages sent by the interface and generation of the address of the memory for received messages.

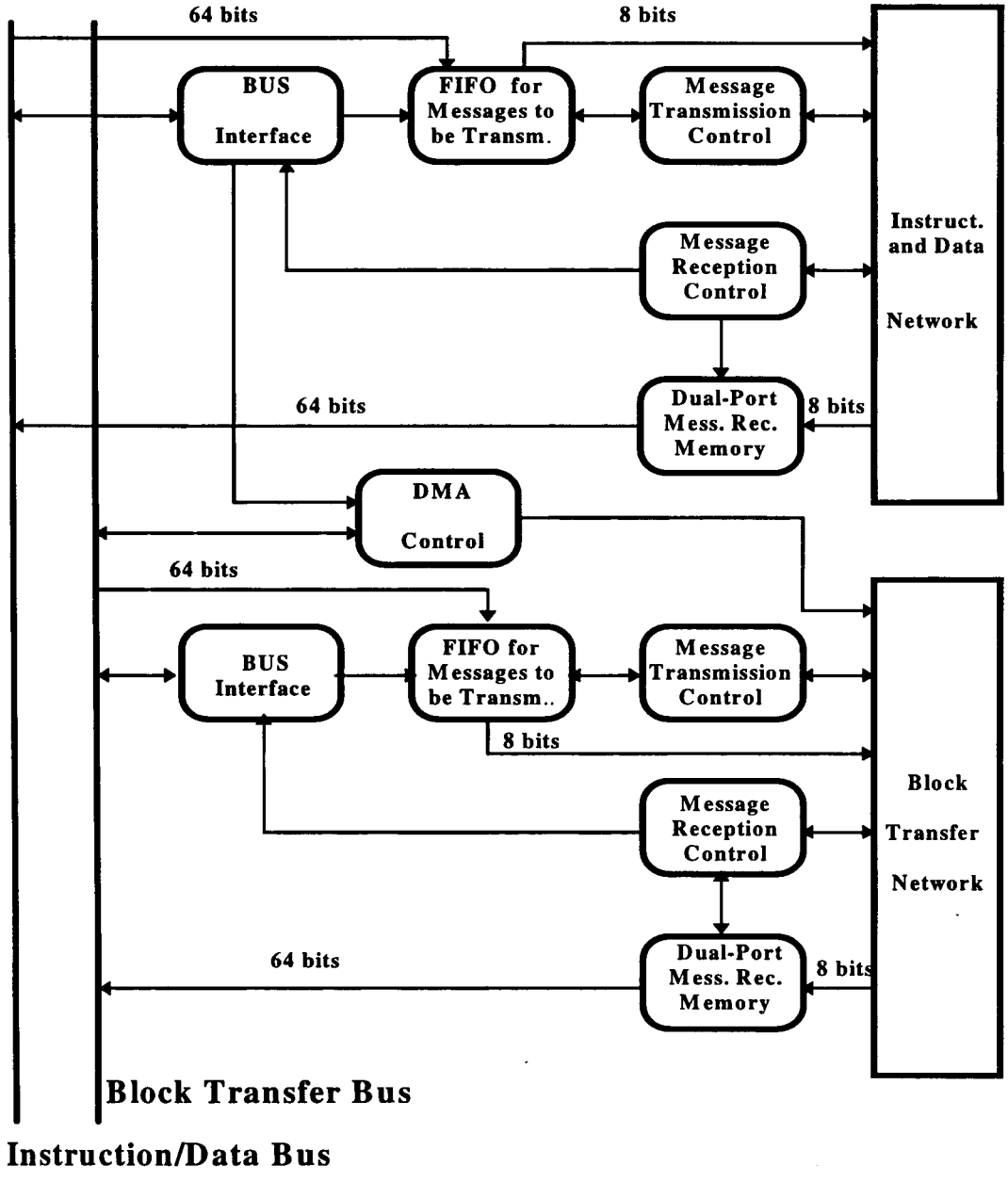

Figure 4: The Network Interface

The Master section of the Network Interface is activated when some remote Read, Write or DMA message arrives at the Interface or when a Write Reply message is received. The Slave section is activated either when a remote access is generated within the cluster or when a Read Reply message is received. In the first case, the infomation on the requested access is stored in the memory for messages to be transmitted for later processing. The Read Reply message occurs because at some point a cluster module requested a remote read operation to the Network Interface. As an immediate answer to this read request, the Slave section sends an instruction for the cluster module to relinquish the use of the cluster bus and retry the read operation

later on. Hopefully, in the meantime, the Newtork Interface has enough time to process the read request and get a Read Reply message as a result. Therefore, when the cluster module retries the read operation, the Slave section is able to send back the requested data to the cluster module. This approach avoids blocking the cluster bus while the Network Interface gets the answer for a remote read operation.

The Message Transmission Control module is responsible for taking messages byte by byte out of the memory for messages to be transmitted, packing them and transmitting them through the Interconnection Network. The Message Reception Control module receives the messages coming from the Interconnection Network, stores them in the memory for received messages and instructs the bus interface module to generate the appropriate cluster bus access.

In addition to the EPLDs, a FIFO memory has been used to implement the memories for the messages to be transmitted. This FIFO memory consists of two sections: a 64-bit wide data section and an 18-bit wide control section. The dual-port memories for message reception consist of 64-bit words and are divided into three different regions. The first one works as a FIFO for the received messages. The second one works as a RAM which stores the replies to messages sent by modules within the local cluster and the third one stores an address and access code table for the interrupt registers of all the modules within the local cluster. From one port, this memory is accessed for the reception of messages coming from the Network in 8-bit packets. From the other port, this memory is connected to the corresponding 64-bit cluster bus and can be read by the master or slave section of the Interface and written by the slave section or by the DMA

5. THE I/O PROCESSOR

The architecture of the MULTIPLUS I/O Processor [Oliv92] is shown in Figure 5. It consists of two bus systems: the CPU Bus and the DMA Bus. Attached to each bus there is a 68020 CPU. The one associated with the CPU Bus is responsible for managing the I/O requests sent by the processing elements to the 16 Kbyte dual-port Command Memory, for performing the Disk Cache control, for sending commands to be executed by the devices on the DMA Bus through the 4 Kbyte Communication Memory and for controlling a serial interface. It uses a 4 Mbyte RAM for its work area and a 64 Kbyte ROM to store the initialization procedure.

The CPU on the DMA Bus controls the execution of the internal tasks issued by the CPU Bus through the Communication Memory. Attached to the DMA Bus there are: a SCSI interface for the connection of disks, tapes and floppies; a Parallel Interface for the connection of printers; a 32 Mbyte write-through Disk Cache; a DMA Controller which is responsible for the data transfer from the SCSI and Parallel Interface to the Disk Cache; and an 8 Kbyte BIFIFO which is used as a temporary storage to transmit data between the Disk Cache and the processing elements through the Block Transfer Bus.

Two EPLDs are used to perform some control functions within the I/O Processor. The first one performs the master/slave functions on the Instruction/Data Bus. The second one performs the master/slave functions on the Block Transfer Bus and controls the burst data transfers between the Disk Cache and the BIFIFO on the DMA Bus.
The operation of the I/O Processor is started when a Processing Element writes an I/O command into its assigned region within the Command Memory. This generates an interrupt to the CPU Bus 68020 which, then, interprets the command and, if necessary, splits it into sub-tasks that will be performed by the I/O Processor hardware attached to the DMA Bus. For instance, if the command is a disk block read operation, the CPU Bus 68020 firstly checks if the block is stored within the Disk Cache. If it is, a command to transfer the block from the cache to the processing element memory is issued to the DMA Bus through the Communication Memory. Otherwise, the command is split into two tasks: the reading of data from the disk to the cache under the supervision of the DMA Controller and the data transfer from the cache to the Processing Element memory through the BIFIFO under the control of the EPLD. Again, both tasks are issued to the DMA Bus through the Communication Memory. Once all steps of a Processing Element command have been executed by the DMA Bus, the CPU Bus does a write operation to the interrupt register of the Processing Element through the Instruction/Data Bus.

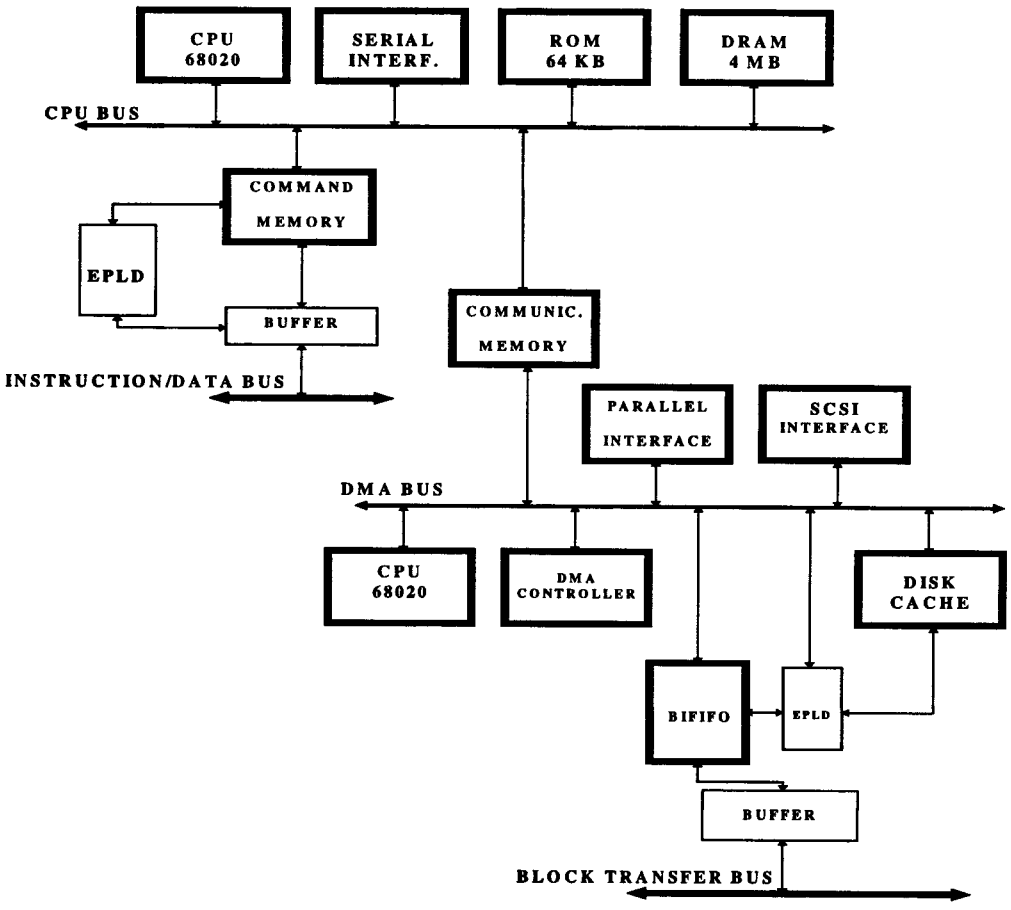

Figure 5: The I/O Processor Architecture

6. IC DESIGNS FOR THE MULTIPLUS ARCHITECTURE

CMOS designs of VLSI circuits to be used in the implementation of the MULTIPLUS architecture have been undertaken as a driving force to improve our expertise in IC design and to enhance the public domain CAD systems we have available at NCE/UFRJ. As a result, within this branch of the MULTIPLUS project, research work both in the areas of IC design and CAD tools [Lope92, Sant94, Serd96] are currently under development. Up to now the development of VLSI circuits has been carried out with the use of two public domain CAD systems: MAGIC, developed at the University of Berkeley and ALLIANCE, developed at the Laboratory MASI, Université Pierre et Marie Curie. The resulting chips have been fabricated usign the facilities available within the Brazilian PMU and the IBERCHIP Program.

The mainstream of this research effort is the design of NCESPARC, a 32-bit RISC microprocessor, using CMOS 1.0u technology. Previous works have reported on simulation analysis of alternative implementations of the NCESPARC architecture

[Silv91, Silv92], on the NCESPARC proposed architecture [Barb90], on the design of the most important modules of the NCESPARC Data Path [Barb92, Barb94, Youn96], on the design of the NCESPARC Control Unit [Aude95a] and on alternative approaches to implement the NCESPARC complete Data Path [Aude96a].

The NCESPARC architecture follows the SPARC version 7.0 definition with some simplifications: elimination of the tagged and the multiply step instructions. The 32-bit Data Path consists of a three-port Register File (two read ports and one write port), an ALU, a Barrel Shifter and a few auxiliary registers.

The architecture is implemented as a four-stage pipeline: instruction fetch; instruction decoding and operand fetching; instruction execution; and writing of the result in the register file. For each pipeline stage, there is an instruction register associated with it which stores the code of the instruction under processing at that stage. The pipeline clock cycle is 50 ns and it consists of four time steps of 12.5 ns. For each pipeline stage, a set of logic equations has been written to describe the required control logic.

The control unit commands the NCESPARC Data Path by activating: the multiplexor selection bits; the load operation on auxiliary registers; the addressing and the read/write operations on the register file; and the ALU and Barrel Shifter operations. The implementation of the NCESPARC Control Unit is based on the use of the Standard-Cell approach.

The overall architecture of the NCESPARC 32-bit Data Path is shown in Figure 6. The main components of this Data Path are the Register File, the ALU and the Barrel Shifter. Auxiliary registers are inserted between these main modules to isolate operations between the NCESPARC pipeline stages. After the instruction fetch and decoding, the normal operation of the pipeline second stage consists of the reading of two operands from the Register File. These operands are stored in Registers A and B for the use of the pipeline third stage. This stage usually performs an operation in the ALU or in the Barrel Shifter using the contents of Registers A and B. The result of this operation is stored in Register S. The fourth and last pipeline stage writes the contents of Register S back to the Register File.

The Register File has two read ports and one write port, allowing the NCESPARC pipeline to read two operands of a given instruction while the result of a previous instruction is stored back in the Register File. The register file is organized into 8 windows. Each window consists of 24 registers with an overlap of 8 registers with each neighbouring window. In addition, 8 global registers (R0 to R7) are available. Therefore, the total number of 32-bit registers available is 136 and the address for each access port of the Register File is 8-bit wide. Register R0 is permanently set to 0.

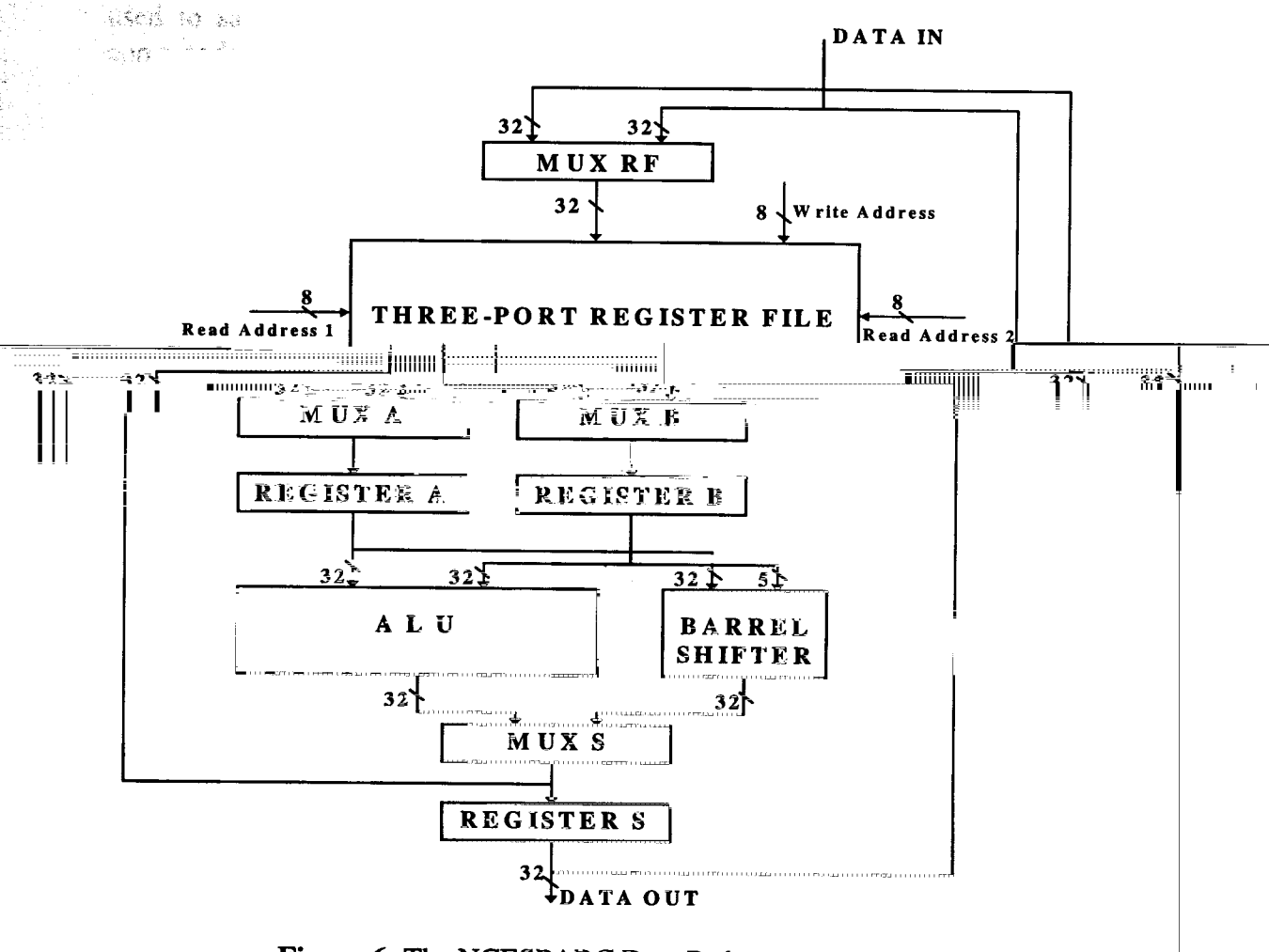

Figure 6: The NCESPARC Data Path

The ALU can perform the following 10 operations on its inputs A, B and Cin (Carry input): A + B; A + B + Cin; A - B; A - B - Cin; A and B; A and (not B); A or B; A or (not B); A xor B; A xor (not B). The ALU must also provide information on the occurrence of overflow, a result equal to zero, a negative result and a carry output. Registers A and B are normally loaded with data contents from the Register File. However, Register B can also be loaded with an external data input used for immediate operands. Register A can be alternatively loaded with data coming from the Multiplexor S. This option implements a by-pass which is used by the Control Unit whenever one instruction uses a Register File operand which is modified by the result of the immediately previous instruction. If this by-pass were not provided, at least one NOP instruction would have to be inserted between the two instructions since the result of an operation is written to the Register File one cycle after the reading of the operands by the next instruction. The selection of the data to be stored in Registers A and B is commanded by the Control Unit through the Multiplexors A and B, respectively.

The 32-bit Barrel Shifter shifts the operand stored in Register A. The shift count is applied by the 5 least significant bits of Register B. Three operations can be performed: logic left shift, logic right shift and arithmetic right shift. One control bit is

applied by the 5 least significant bits of Register B. Three operations can be

used to say if the shift operation is logic or arithmetic and another control bit commands if the shift operation will be to the left or to the right.

In addition to the design of the NCESPARC microprocessor, a CMOS implementation of the MULTIPLUS Bus Arbiters has been performed using CMOS 1.0u technology. This chip has recently been received back from fabrication and has performed according to the specifications. A detailed description of the design of this chip was presented by Barbosa et al. [Barb96]. Experiments of designing VLSI chips to implement some finite state machines within the MULTIPLUS Network Interface have also been reported by Pinto [Pint96].

7. THE MULPLIX OPERATING SYSTEM

MULPLIX [Azev93] is a UNIX-like operating system designed to support medium-grain parallelism and to provide an efficient environment for running parallel applications within MULTIPLUS. In its initial version, MULPLIX will result from extensions to Plurix, an earlier Unix-like operating system developed to support multiprocessing within the Pegasus architecture [Fall89].

Plurix main goal was to provide an efficient environment for running general-purpose processes on an architecture consisting of a few processors and a global memory which can be accessed with the same time penalty by all processors. Therefore, Plurix supports only large-grain parallelism or concurrency and assumes that the underlying machine is implemented by a Uniform Memory Access architecture.

For the MULTIPLUS environment it is essential for the operating system to be very efficient in supporting applications which consist of a large number of processes that may run in parallel, demanding synchronization and, consequently, a lot of context switching operations. One of the basic conditions to reach this goal is to heavily reduce the overhead in such operations.

To solve this problem, one major extension to Plurix included in the MULPLIX definition is the concept of ***thread***. Within MULPLIX, a thread is basically defined by an entry point within the process code. A parallel application consists of a process and its set of threads. Therefore, when switching between threads of a same process, only the current processor context needs to be saved. Information on memory management and resource allocation is unique for the process as a whole and, therefore, remains unchanged in such context-switching operations.

In relation to synchronization, MULPLIX makes available to the user synchronization primitives for the manipulation of mutual exclusion and partial order semaphores. In addition, MULPLIX implements the busy-waiting primitives in a different way, since it is essential to avoid hot spots through the interconnection network. The algorithm which has been adopted for the solution to this problem is an adaptation of the one proposed by Anderson [Ande90] and is based on the following ideas [Azev90]: the use of a circular buffer to implement the queue of processors waiting for the binary semaphore and the detection of the availability of a binary semaphore by testing a

cacheable local variable.

Within Plurix the memory space allocated to a process consists of a data segment, a code segment and a stack segment for the user and supervisor modes. Memory sharing between processes is very limited. It only allows the implementation of Unix pipes between two processors in memory. Within MULPLIX, it is essential for the memory management system to worry about data locality, to support the concept of a process consisting of several threads and to allow memory sharing between threads of the same process. The following facilities are supported by the MULPLIX memory management system: replication of the MULPLIX kernel code in every processing element; replication of the process code in every cluster where a given process is running; definition of an additional non-shared local data segment for each thread; definition of an additional local data segment in supervisor mode which is shared by all threads running on the same processing element; and definition of stack segments in the user and supervisor modes for each thread. In addition, the MULPLIX memory management system is also concerned with the problem of maintaining cache consistency between MULTIPLUS clusters.

Process scheduling is another area in which MULPLIX must use a different approach to the one adopted in Plurix. Within Plurix, there is a single queue of processes which are ready for execution and the scheduling policy does not take into consideration data locality. In addition, time-sharing between processes is always used. Within MULPLIX, a specified number of processors will not run in time-sharing mode. Such processors will be scheduled to run threads of parallel scientific applications. The non-time sharing policy ensures that these threads may run as fast as possible and without interrupts as long as they can or wish. On the other hand, the execution of interactive processes is ensured by the fact that there will always be a fraction of processors running with time-sharing.

Data locality is taken into consideration by the MULPLIX scheduling system through the use of separate queues of threads which are ready to be run in each cluster. Every queue can be accessed by any processor. However, a free processor will only look for a thread to run in another cluster queue if it finds its own cluster queue empty.

7.1 MULPLIX Parallel Programming Primitives

The MULPLIX system provides a set of system calls for the development of parallel programming applications within the MULTIPLUS architecture[Azev93a]. These primitives deal with the following aspects: the creation of threads; memory allocation; and synchronization. The current implementation of MULPLIX is running on an EBC 32020, a 68020 based machine to which the Plurix operating system had been previously ported. Due to the limitations imposed by this environment, the implementation of some of the primitives has not been performed yet fully in accordance to the original specification.

The system call, "***thr_spawn***", is provided for the creation of a group of threads. The number of threads to be created, the name of the procedure to be executed by these threads and a common argument are the basic parameters of this system call and the

ones which are supported by the MULPLIX current implementation. However, one optional parameter will be added to this system call to define preferential processing elements for the execution of each thread to be created. This facility will allow an experienced user to enforce the assignment of a particular thread to the processing element which is known to host the set of data to be mostly used by that thread. A second version of this system call, ***"thr_spawns"***, will allow the creation of threads in synchronous mode. If the thread creation is synchronous, the parent thread will suspend its execution until execution completion by all the children threads it has started

Three additional primitives for thread control have also been made availeble within MULPLIX. The first one is "***thr_id***" which returns the identification number of a thread, ***tid***, within MULPLIX. The second one is "***thr_kill***" which allows any thread to kill another thread within the same process. All the descendants of the killed thread are also killed. The only parameter of this system call is the ***tid*** of the thread to be killed. The last primitive is ***"thr_term"*** which allows a forced termination of the thread.

The memory allocation primitives can perform shared and private data allocation. For shared data, the primitive "***me_salloc***" offers two options: a concentrated and a distributed memory space allocation. In the first case, it is expected that most of the accesses to the memory space to be allocated will be performed by the thread which has performed the system call and, therefore, all memory space is allocated within the local memory of the thread preferential processing element. The distributed allocation is used when a uniformly distributed access pattern among the threads is expected. Within the EBC 32020, there is only a single processing element and the concentrated/distributed option is meaningless. Therefore it has not been implemented yet. The primitive which performs private memory allocation is "***me_palloc***".

The MULPLIX operating system offers two explicit synchronization mechanisms. The first one is used for mutual exclusion relations and the second one is employed when a partial ordering relation is to be achieved. For the manipulation of mutual exclusion semaphores, primitives are provided for creating ("***mx_create***"), allocating ("***mx_lock***"), extinguishing ("***mx_delete***") and releasing ("***mx_free***") a semaphore. In addition, the primitive ***"mx_test"*** allows a thread to allocate a semaphore if it is free without causing the thread to wait if the semaphore is still occupied. Simple and multiple mutual exclusion synchronizations are supported. With multiple mutual exclusion, a maximum of a given number of threads can execute the critical region simultaneously.

For partial ordering semaphores, which implement barrier-type synchronization, primitives for creating ("***ev_create***"), asynchronous signalling ("***ev_signal***"), waiting on the event occurrence ("***ev_wait***"), synchronous signalling ("***ev_swait***") and extinguishing ("***ev_delete***") an event are provided. The primitives ***"ev_set"*** and ***"ev_unset"*** have also been implemented to allow unconditional setting and resetting of an event. This may be useful in test, debugging or in error recovery procedures.

The following example illustrates the use of some of these primitives in the implementation of a parallel dot product , ***vectc = vecta . vectb***, assuming that the vectors are of size ***n*** and that ***P*** processing elements are available to run the algorithm.

```
#include <threads.h>
#include <stdio.h>

float vecta[n], vectb[n], vectc[n];
EVENT product;

main ( )
{
   int i;
   float sum = 0.0;

   product = ev_create (P, 1);
   thr_spawn (P, dot_prod, 0);
   ev_wait (product);
   for (i = 0; i < P; i++)
       sum += vectc[i];
   printf ("Dot Product: %f\n", sum);
   ev_delete (product);
}

dot_prod (arg, p)

int arg;
int p;
{
   int i;
   vectc[p] = 0.0;

   for (i = p*(n/P); i< (p+1)*n/P; i++)
      vectc [p] += vecta[i] * vectb[i];

   ev_signal (product);
}
```

In this example, the system call "thr_spawn", issued by the main thread, starts *P* threads to run the procedure *dot_prod* with no common argument. The main thread waits on the event *product*, which has been defined as an event to be signalled by P threads (first parameter of the *ev_create* system call) and to be recognized by a single thread (second parameter of the *ev_create* system call). Alternatively, a synchronous thread spawn could have been issued and, in this case, the waiting for the event *product* would be implicit. Each of the P threads receives from the system information on its order in the group of threads that has been created through the variable *p*, calculates the dot product associated with the section number *p* of length *n/P* of *vecta* and *vectb*, stores the result in the corresponding *p* position of vector *vectc* and signals

the event *product*. The main thread restarts on the occurrence of the event *product* and sums up all the elements of *vectc* to find the final result of the dot product.

8. PARALLEL PROGRAMMING ENVIRONMENTS

In addition to the native parallel programming environment to be provided by the MULPLIX operating system, three other environments are currently under implementation as libraries: the standard PVM, M-PVM and Pthreads.

The standard PVM (Parallel Virtual Machine) [Geis94] implementation will provide the MULTIPLUS/MULPLIX platform with a widely used message-passing environment which will enhance portability of several parallel applications to the system. The PVM implementation within the MULTIPLUS/MULPLIX plataform will use "pipes" within the shared memory space to implement communication between PVM tasks. Different threads within the same process will, in principle, handle communication work and the task processing itself.

M-PVM is an implementation of PVM which lacks total compatibility with the standard PVM, but can provide higher performance within the MULTIPLUS/MULPLIX platform. Each PVM task is mapped on a MULPLIX thread and the message passing functions are implemented using the MULPLIX shared memory among threads of the same process. M-PVM is in fact a hybrid environment which provides applications with efficient implementations of PVM message passing functions and with the possibility of using shared memory. An initial implementaion of M-PVM is currently in operation within the MULPLIX system which is running on an EBC 32020 computer. Recently, M-PVM has also been made available on Solaris with the implementation of a library of MULPLIX primitives on top of Solaris Light Weight Processes.

Pthreads [Sun95] is the POSIX threads standard which has been defined by the POSIX work group 1003.4a. The implementation of Pthreads within the MULPLIX system aims at offering to the user a more powerful and simple to use multi-threaded parallel programming environment than the native MULPLIX environment. In addition, such implementation opens new possibilities for a direct porting of parallel applications to the MULTIPLUS/MULPLIX platform.

The current Pthreads implementation within the MULPLIX system [Barr96a] is running on the EBC 32020 computer or within Solaris SPARCstations. It supports: the creation of detached and joinable [Barr96a, Sun95] threads; facilities for the manipulation of thread attributes; the termination of threads; the definition of a private memory for a given thread; mutual exclusion semaphores; conditional semaphores [Barr96a, Sun95]; and the unique execution of routines associated with unique execution keys [Barr96a, Sun95].

9. PARALLEL ALGORITHMS

In this area of research, the study of efficient parallelization techniques for Simulated Annealing and Genetic algorithms when applied to the placement problem in VLSI circuits is under consideration. At first, the use of a dedicated Ethernet cluster of homogeneous IBM 25T workstations has been considered in this practical study.

Holland [Holl75] has proposed the Genetic algorithms as programs which could reproduce the evolution process which is found in nature. The Genetic algorithms manipulate a population of potential solutions for an optimization or search problem. They operate on an encoded representation of the solutions. Each solution has associated with it a measure of its quality, which is called *fitness*. A *selection* mechanism forces the continuous evolution of the quality of the generations. An individual (a solution) ,with a high fitness value has a greater probabiblity to reproduce and survive. The recombination of genetic material is simulated through a *crossover* mechanism where pieces of two parent solutions are exchanged to make up a new individual. Another operation, called *mutation*, may cause random alterations in the encoded representation of an individual.

Through a detailed experimental analysis [Knop96], it was obeserved that the correct tuning of the control parameters is essential for the Genetic algorithm to produce nearly optimal results for the placement problem. Once the correct tuning of the control parameters was obtained, the algorithm proved to be very suitable for parallelization. A proposal for the parallel implementation of the algorithm which achieves a considerable reduction of the need for communication among processors has been presented by Knopman [Knop96, Knop96a]. This implementation has produced results with similar quality to the sequential version and exhibited a speed-up around 6 in a cluster consisting of 8 workstations.

Simulated Annealing [Kirk 83] searches the problem solution space by performing hill climbing moves. It can escape of a local minimum by accepting, with some probability, moves that generate a worse solution. This probability decreases as the algorithm evolves to the final solution. The algorithm usually converges to a nearly optimal solution but with a very high computational cost. This has motivated research work aiming at producing efficient Simulated Annealing parallel algorithms which are reasonably scalable and able to generate solutions as good as the sequential algorithm.

Within Simulated Annealing, the temperature parameter, which controls the hill climbing moves, can deeply affect the algorithm behaviour during its execution. It has been shown that such dynamic behaviour offers new opportunities for the exlploration of parallelism within the algorithm [Knop96]. The adopted parallelization approach consists of changing the used algorithm according to the optimization process phase. In fact, different parallelization techniques are used for high and low temperature values. This aspect can be regarded as a first level of adaptation introduced in the algorithm.

The algorithm used for low temperatures is itself an adaptive version of the speculative algorithm proposed by Sohn [Sohn95]. Within this adaptive algorithm, the

number of processors allocated to the solution of the placement problem and the number of moves evaluated per processor between synchronization points change with the temperature. In its domain of operation this algorithm has produced results of the same quality as those generated by the serial version with an speedup near to 4 when a large circuit is to be placed and up to 10 processors in the cluster are used.

At high temperatures, an algorithm based on the parallel evaluation of independent chains of module moves by each processor has been adopted. It is shown that results with the same quality of those produced by the serial version can be obtained when shorter length chains are used in the parallel implementation. For large circuits, a speedup around 3 has been achieved by this algorithm with the use of 10 processors in the cluster.

10. CURRENT STATUS AND PERSPECTIVES

The MULTIPLUS architecture definition and detailed logic design have been completed. Currently, we are working in the implementation and test of an initial prototype with 8 Processing Elements and a single I/O Processor organized into up to four clusters. Most of the hardware modules have been tested isolatedly. The integration of the hardware modules within a cluster has been started considering the use of 1 I/O Processor and 2 Processing Elements.

In parallel, we are undertaking the new design of the Processing Element based on the use of two SuperSPARC-II modules. As a research goal, we intend to develop a VLSI design for the Interconnection Network which will also support multicast and broadcast message types. It is also under consideration the development of a Network Interface which may support a 2nd-level cluster cache, which will help to hide the network latency.

For the design of the NCESPARC which should be completed in the first quarter of 1997, the evolution which is envisaged is the development of NCESPARC+, including facilities for the efficient support to multi-threading in hardware and the execution of several instructions in parallel [Joao95, Joao96].

The implementation of the MULPLIX Operating System initial version as an evolution of Plurix is currently running on EBC 32020 computers. A library which implements MULPLIX system calls has already been made available at the Solaris environment using the concept of Light Weight Processes. The implementation of a first version of a MULPLIX library which supports multi-threaded programming [Barr96] has also been completed and tested. The implementation of M-PVM and Pthreads is currently available on an EBC 32020 computer or on Solaris.

Current work is concentrated on the transport of MULPLIX to the MULTIPLUS platform; on the development of a memory management system for the MULPLIX Operating System which implements techniques for maintaining inter-cluster cache consistency by software; on the implementation of a standard PVM library using the concept of "pipes" in memory; on the optimization of the M-PVM implementation

within the MULTIPLUS/MULPLIX platform; on the development of a first prototype of a visual parallel programming tool; and on the implementation of parallel versions of the Simulated Annealing and Genetic algorithms for the M-PVM and the native MULPLIX enviornments.

Up to now the development of the MULTIPLUS/MULPLIX research project has produced several papers in national and international Conferences, some technical reports, 9 M.Sc. thesis and 1 Ph.D. thesis. Currently two M.Sc. thesis and two Ph.D. thesis are under development within the project scope.

The first MULTIPLUS prototype is expected to be running under the MULPLIX Operating System version by the second semester of 1997. The goal is to make this proptotype available for use by other research groups, in particular those at the Federal University of Rio de Janeiro, which are currently involved with work in several areas that may benefit from the MULTIPLUS computing power and parallel environment. It is through such experience of use that we hope to have new insights into the problem of parallel processing and, therefore, be able to improve the performance of the MULTIPLUS/MULPLIX system.

ACKNOWLEDGEMENTS

The author would like to thank FINEP, CNPq, RHAE and CAPES/COFECUB for the support given to the development of this research work. The author would also like to thank the research team directly involved with the development of the MULTIPLUS/MULPLIX project: Alexandre M. Meslin, Alexandre M. Gomes, Aluísio de A. Cruz, Cláudio Miguel P. Santos, Delane Soares, Gerson Bronstein, Gladstone Moisés, Jonas Knopman, Luiz Fernando M. Cordeiro, Márcio O. Barros, Márcio T. Young, Mário A. S. Barbosa, Mário João Jr., Paulo A. S. Simões, Sidney de C. Oliveira

REFERENCES

[Ande90] "The performance of spin lock alternatives for shared memory multiprocessors", Anderson, T.E., IEEE Transactions on Parallel and Distributed Systems, vol. 1, no. 1, pp. 6-16, January 1990

[Aude90] "MULTIPLUS: Um Multiprocessador de Alto Desempenho", J.S. Aude et alii, Proceedings of the X SBC Congress, Vitória, ES, pp. 93-105, July 1990

[Aude91] "Multiplus: A Modular High-Performance Multiprocessor", J.S. Aude et alii, Proc. of the EUROMICRO 91, Vienna, Austria, pp. 45-52, September 1991

[Aude94] "Multiplus/Mulplix: An Integrated Environment for the Development of Parallel Applications", J.S. Aude, Proc. of the IEEE/USP International Workshop on High Performance Computing - WHPC'94, pp. 245-255, São Paulo, March 1994

[Aude95] "Implementation of the Multiplus/Mulplix Parallel Processing Environment", J.S.Aude et al., Proceedings of the VII SBAC-PAD - Canela, RS, July 1995;
[Aude95a] "Design of the NCESPARC Control Unit using the Alliance System", J.S.Aude, Proceedings of the X SBMicro Congress - Canela, RS, August 1995;

[Aude96] "The Multiplus/Mulplix Parallel Processing Environment", J.S.Aude et al. - Proceedings of the International Symposium on Parallel Architectures, Algorithms and Networks (I-SPAN 96) - Beijing, China, June 1996

[Aude96a] "A Comparative Analysis of Two Approaches to the Design of the NCESPARC Data Path", J. S. Aude, M. A. S. Barbosa, M. T. Young, A. M. Gomes, Proceedings of the XI SBMICRO Congress, Águas de Lindóia, SP, August 1996, pp. 99-105

[Azev90] "MULPLIX: Um Sistema Operacional tipo UNIX para o Multiprocessador MULTIPLUS", Azevedo, G.P., Azevedo R.P., Figueira, N.R., Aude, J.S., Proceedings of the III Brazilian Symposium on Computer Architecture - Parallel Processing, Rio de Janeiro, RJ, pp. 122-137, November 1990

[Azev93] "MULPLIX: Um Sistema Operacional tipo Unix para Programação Paralela", R.P. Azevedo, M.Sc. Thesis, COPPE/UFRJ, March 1993

[Azev93a] "Primitivas para Programação Paralela no MULTIPLUS", Azevedo, R.P., Azevedo, G.P., Silveira, J.T.C, Aude, J.S., Proceedings of the V Brazilian Symposium on Computer Architecture, Florianópolis, pp. 761-775, September 1993

[Barb90] "Implementação de Microprocessador RISC com Arquitetura SPARC", M. A. S. Barbosa, N. R. Figueira, G. P. Silva, J. S. Aude, Proc. of the V SBCCI, Ouro Preto, MG, October 1990, pp. 121-131

[Barb92] "Unidade Lógica e Aritmética Rápida de 32 bits", M. A. S. Barbosa, A. R. Sales, Proc. of the VII SBCCI, Rio de Janeiro, RJ, September 1992, pp. 234-244

[Barb94] "Implementação e Verificação do Chip PMU CMOS8", M. A. S. Barbosa, A. R. Sales, Proc. of the IX SBMICRO, Rio de Janeiro, RJ, August 1994, pp. 369-376

[Barb96] "Implementação em ASIC de um Árbitro de Barramento", M.A.S. Barbosa, M.T. Young, A.M. Gomes, Proceedings of the IX SBCCI - Recife, Pernambuco, March 1996

[Barr96] "Implementação de Bibliotecas Multi-Thread no Sistema Operacional Mulplix", Barros, Márcio Oliveira; Aude, Júlio Salek, - Proceedings of the VIII SBAC-PAD - Recife, PE - August 1996

[Barr96a] "Implementação do Padrão Pthreads para o Sistema Operacional Mulplix", M. O. Barros, Technical Report NCE-01/96, September 1996

[Bron90] "Análise de Desempenho de Redes de Interconexão para Máquinas Paralelas", Bronstein, G., Cruz, A.J.O, Duarte, O.C.M.B., Proc. of the III Brazilian Symposium on Computer Architecture - Parallel Processing, Rio de Janeiro, pp. 345-360, Nov. 1990

[Bron91] "Projeto de uma Rede de Interconexão para uma Máquina Paralela de Alto Desempenho", G. Bronstein, M. Sc. Thesis, COPPE/UFRJ, 1991

[Bron93] "O Subsistema de Interconexão do Multiprocessador MULTIPLUS", Bronstein, G., Proceedings of the V Brazilian Symposium on Computer Architecture, Florianópolis, pp. 166-173, September 1993

[Bron96] "Project and Implementation of a High-PerformanceSwitching Element Using EPLDs", Bronstein, Gerson, Proceedings of the XI SBMICRO Congress-Águas de Lindóia, SP - August de 1996 - pp. 93-98

[Fall89] "Plurix: A multiprocessing Unix-like operating system", Faller, N., Salenbauch, P., Proceedings of the 2nd Workshop on Workstation Operating Systems, IEEE Computer Society Press, Washington, DC, USA, pp. 29-36, September 1989

[Geis94] "PVM3 Users's Guide and Reference Manual", Geis, A., Oak Ridge National Laboratory, 1994

[Holl75] "Adaptation in Natural and Artificial Systems", Holland, J.H., University of Michigan Press, Ann Arbor, 1975

[Joao95] "Compactação Local de Código para uma Arquitetura SPARC VLIW", M.João Jr., J.S.Aude - Proceedings of the VII SBAC - PAD, Canela, RS - July 1995

[Joao96] "Compactação de Código para Arquiteturas SPARC VLIW", M. João Jr., M. Sc. Thesis, COPPE/UFRJ, January 1996

[Kele92] "Lazy Release Consistency for Software Distributed Shared Memory", P. Keleher, A. L. Cox, W. Zwaenepoel, Proceedings of the 19th International Symposium on Computer Architecture, pp. 13-21, Gold Coast, Australia, May 1992

[Kirk83] "Optimization by Simulated Annealing", Kirkpatrick S., Gelatt C.D., Vecchi M.P., SCIENCE, Volume 220, Number 4598, May 1983.

[Knop96] "Algoritmos Genéticos e Simulated Annealing: Aplicacão ao Problema de Placement e Técnicas de Paralelização", J. Knopman - D.Sc. Thesis - COPPE/UFRJ - Rio de Janeiro - May 1996

[Knop96a] "Paralelização de Algoritmos Genéticos Aplicados ao Problema de Placement em Clusters de Estações de Trabalho", Knopman, Jonas; Aude, Júlio Salek, - Proceedings of the VIII SBAC-PAD - Recife, PE, August 1996

[Kont95] "High Performance Software Coherence for Current and Future Architectures", L.I. Kontothanassis, M.L. Scott, Journal of Parallel and Distributed Computing, Vol. 29, No. 2, September 1995, pp. 179-195

[Lope92] "Síntese de Lógica Combinacional Multinível", E.P. Lopes Fo., M.Sc. Thesis, COPPE/UFRJ, September 1992

[Mesl90] "Sistemas de Memórias Multicache para uma Máquina Paralela MIMD: Projeto MULTIPLUS", A.M. Meslin, A.C. Pacheco, Proceedings of the III Brazilian Symposium on Computer Architecture - Parallel Processing, Rio de Janeiro, pp. 179-193, November 1990

[Mesl91] "Estudo de Arquiteturas de Memória Cache para o Multiprocessador Multiplus", A.M. Meslin, M.Sc. Thesis, COPPE/UFRJ - August 1991

[Mesl92] Meslin, A.M., Pacheco, A.C., Aude, J.S., "A Comparative Analysis of Cache Memory Architectures for the MULTIPLUS Multiprocessor", Proceedings of the EUROMICRO 92, Paris, France, pp. 555-562, September 1992

[Oliv90] Oliveira, S.C., Aude, J.S., "O Subsistema de Memória de Massa do Multiprocessador MULTIPLUS", Proceedings of the III Brazilian Symposium on Computer Architecture - Parallel Processing, Rio de Janeiro, RJ, pp. 298-313, November 1990

[Oliv92] "Uma Avaliação do Impacto das Operações de E/S no Desempenho do Multiprocessador MULTIPLUS", S.C. Oliveira, J.S. Aude, Proceedings of the IV Brazilian Symposium on Computer Architecture - São Paulo, SP, pp. 379-394 , October 1992

[Oliv92a] "Uma Proposta de Arquitetura de E/S para o Multiprocessador Multiplus", S.C. Oliveira, M,Sc. Thesis, COPPE/UFRJ - February 1992

[Pete94] "An Evaluation of Multiprocessor Cache Coherence Based on Virtual Memory Support", K. Petersen, K. Li, Proceedings of the 8th International Parallel Processing Symposium, Cancún, Mexico, April 1994, 158-164

[Pint96] "Projeto de Máquinas de Estado Finito usando o Sistema Alliance", S.B.Pinto, G.Bronstein, J.S.Aude - Proceedings of the II IBERCHIP Workshop, São Paulo - SP, February 1996

[Sant94] "Silence: Uma Ferramenta para Síntese de Lógica Multinível", C. M. P. Santos, E. P. Lopes Fo., J. S. Aude, Proceedings of the IX SBMicro Congress, Rio de Janeiro/RJ, August 1994, pp. 477-487

[Serd96] "FPPR: Alocador e Roteador de Macroblocos para o Sistema Alliance", H. Serdeira, J. S. Aude, Anais do IX SBCCI, Recife, PE, March 1996

[Silv91] "Estudo e Avaliação de Arquiteturas RISC para uso em Sistemas Multiprocessadores", G. P. Silva, M.Sc. Thesis, COPPE/UFRJ, June 1991

[Silv92] "Evaluation of a SPARC Architecture with Harvard Bus and Branch Target Cache", G. P. Silva., J.S. Aude, Microprocessing and Microprogramming, V. 34-p. 157-160, Jan. 1993

[Sohn95] "Parallel N-ary Speculative Computation of Simulated Annealing", Sohn,A., IEEE Transactions on Parallel and Distributed Systems, Vol. 6, No. 10, October 1995.

[Sun95] "Solaris Multithreaded Programming Guide", Sun Microsystems, SunSoft, 1995

[Youn96] "Unidade Lógica e Aritmética de Alto Desempenho", M.T.Young, M.A.S.Barbosa- Proceedings of the II IBERCHIP Workshop, São Paulo - SP, February 1996