*** RELATÓRIO TÉNICO ***

MULTIPLUS: A MODULAR HIGH-PERFORMANCE MULTIPROCESSOR

Júlio Salek Aude
Adriano J. O. Cruz
Ageu C. Pacheco Jr.
G. Bronstei
Alexandre M. Meslin
Sidney C. Oliveira
Norival R. Figueira
Rafael P. Azevedo
Gustavo P. Azevedo

NCE 21/91
Dezembro/91

Universidade Federal do Rio de Janeiro
Núcleo de Computação Eletrônica
Caixa Postal 2324
20001 - Rio de Janeiro - RJ
BRASIL

MULTIPLUS: A MODULAR HIGH-PERFORMANCE MULTIPROCESSOR

ABSTRACT

The MULTIPLUS project which is currently under development at NCE/UFRJ, Brazil, aims at the study of parallel processing problems in MIMD environments. The project includes the development of a parallel shared-memory architecture and a UNIX-like operating system called MULPLIX. The MULTIPLUS architecture uses an inverted n-cube multistage network to interconnect clusters of processing nodes designed around a double-bus system. As a consequence, the architecture is partitionable and modular. It can easily and efficiently support configurations ranging from workstations to powerful parallel supercomputers with up to 2048 processing nodes. The MULPLIX operating system provides MULTIPLUS with an efficient computing environment for parallel scientific applications. MULPLIX uses the concept of thread, implements busy-waiting synchronization primitives very efficiently and carefully considers data locality and scientific processing requirements in the policies adopted for memory management and thread scheduling.

MULTIPLUS: UM MULTIPROCESSADOR MODULAR DE ALTO DESEMPENHO

RESUMO

O projeto MULTIPLUS, que está atualmente em desenvolvimento no NCE/UFRJ, objetiva o estudo de problemas de processamento paralelo em ambiente MIMD. O projeto inclui o desenvolvimento de uma arquitetura paralela com memória compartilhada e um sistema operacional tipo UNIX, chamado MULPLIX. A arquitetura do MULTIPLUS usa uma rede de interconexão multiestágio do tipo n-cubo invertido para interligar clusters de nós de processamento projetados em torno de um sistema de barramento duplo. Como consequência, a arquitetura é particionável e modular. Ela pode suportar eficientemente configurações cobrindo um espectro que vai desde estações de trabalho até poderosos supercomputadores contendo 2048 nós de processamento trabalhando em paralelo. O sistema operacional MULPLIX provê o MULTIPLUS com um ambiente eficiente de computação para aplicações científicas paralelas. O MULPLIX usa o conceito de "thread", implementa primitivas de sincronização de espera ocupada muito eficientemente e considera fortemente aspectos de localidade dos dados e requisitos de processamento científico nas políticas adotadas para gerenciamento de memória e escalonamento de "threads".

1. INTRODUCTION

The Computer Center of the Federal University of Rio de Janeiro (NCE/UFRJ) has been involved with the development of research projects in the areas of computer architecture and operating systems since 1973. The motivations for the development of the MULTIPLUS project at NCE are twofold: to create a stimulating opportunity for the continuous development of the NCE research staff expertise in the areas previously mentioned and to give some contribution towards the understanding and solution of parallel processing problems in MIMD environments.

The MULTIPLUS project includes the development of a modular shared-memory architecture and a UNIX-like operating system called MULPLIX. In addition, a CMOS RISC microprocessor based on the SPARC[&] architecture definition [SUN87] is also being designed to be used within the MULTIPLUS processing nodes.

The MULTIPLUS architecture is modular and can support up to 2048 processing nodes. By varying the number of processing nodes, a wide range of computing power can be delivered by the architecture. In addition, the architecture can be partitioned in such a way that communication between processing nodes within a given partition does not interfere with the communication within any other partition. This feature fits nicely to the requirements of computing environments in which several independent processes are ready to run concurrently or in which applications can be split into parallel loosely-coupled tasks which can in turn be split into parallel tightly-coupled sub-tasks. The MULTIPLUS architecture provides such environment through the use of several bus-based systems interconnected by a multistage network.

The MULTIPLUS architecture can be classified as a Non-Uniform Memory Access Architecture (NUMA). Four different costs can be associated with processor-memory accesses depending on how far from the processor is sitting the memory information to be accessed. This characteristic impacts heavily on the memory management and process scheduling policies adopted within MULPLIX, a UNIX-like operating system which is being designed to support medium-grain parallelism and provide an efficient computing environment for parallel scientific applications.

Section 2 of this paper describes the MULTIPLUS architecture in more detail. In this section, the organization of each processing node is presented and the I/O system architecture is also described. The cache consistency problem is discussed in Section 3, which also comments on the solutions to the problem adopted within MULTIPLUS. Section 4 presents the architecture of the multistage interconnection network which makes the MULTIPLUS architecture modular and partitionable. The main results of simulation studies which have been carried out for the definition

of the architecture of the interconnection network switching elements are also discussed in this section. In Section 5, the main features of the MULPLIX operating system are described. MULPLIX initial version is an adaptation to the MULTIPLUS architecture of PLURIX, a UNIX-like operating system which has ben developed at NCE/UFRJ and is able to support multiprocessing. Finally, Section 6 comments on the current status of the project and its perspectives for the near future.

2. THE MULTIPLUS ARCHITECTURE

MULTIPLUS is a shared-memory high-performance computer designed to have a modular architecture which is able to support up to 2048 processing nodes and 32 Gbytes of global memory address space. Each processing node consists of:

. A 32-bit RISC microprocessor based on the SPARC architecture definition;
. A floating point coprocessor;
. Up to 32 Mbytes of memory belonging to the global address space;
Separate 64 Kbyte instruction and data caches;

Up to eight processing nodes can be interconnected through a 64-bit double-bus system making up a cluster. Considering an effective bus bandwidth around 45 Mbytes/second and data caches working in write-through mode, simulation studies have shown that the setting of the maximum number of processing nodes in a cluster to eight can easily be achieved with the use of a double bus system and physically local banks of memory. The MULTIPLUS architecture supports up to 256 clusters interconnected through an inverted n-cube multistage network.

Figure 1 shows the MULTIPLUS basic architecture. Through the addition of processing nodes and clusters, the architecture can cover a broad spectrum of computing power, ranging from workstations to supercomputers. This is possible mainly because the MULTIPLUS architecture consists of clusters of processors designed around bus-based systems interconnected through a multistage network. With this mixed 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 nodes. On the other hand, very large parallel computers, with more than a thousand processing nodes, 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 node 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 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 node 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 cluster the processing node belongs to, the multistage interconnection network and the bus system of the cluster to which the accessed memory bank belongs 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.

As shown in Figure 1, MULTIPLUS uses a distibuted I/O system architecture. There are two I/O processors associated with each cluster. The first one deals with terminals and printers and provides MULTIPLUS with an Ethernet interface. This I/O processor can support up to 8 terminals and provides a buffer for receiving data from the processing nodes.

The second I/O processor in a cluster deals with mass storage devices. The I/O processors sitting on different clusters can communicate with each other through a private high speed network. With this arrangement, it is possible to assign each processing node to a single I/O processor which is responsible for dealing with all I/O requests to or from mass storage devices started by that processing node. These I/O processors have a 16 Mbyte disk cache and are able to transmit data blocks through the cluster bus system.

3. THE CACHE CONSISTENCY PROBLEM

To maintain cache consistency is a well-known problem of multiprocessor architectures in which each processor has its own private cache. Inconsistency may occur whenever one of the several copies of a shared memory block is modified. This situation can only arise for shared, writable memory blocks. Cached read-only or non-shared data are always safe.

Solutions to the problem of maintaining cache consistency in bus-based multiprocessor systems generally use a bus-watching mechanism. However, within MULTIPLUS the problem becomes considerably worse because the architecture uses a double bus within each cluster and a multistage interconnection network. The solution proposed by Stenstrom [STEN89] for maintaining cache consistency through an interconnection network is not directly applicable to the MULTIPLUS architecture because the cache flag field proposed by Stenstrom would have to be 2048 bits long.

Two design decisions have been taken to simplify the problem of maintaining cache consistency within the MULTIPLUS architecture.

The first one is to have, within the clusters, one of the busses dedicated to data read/write operations and the other one dedicated to instruction read operations. Under this scheme, only the 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. The second design decision was to impose some restrictions on the type of information which is cacheable within MULTIPLUS. Read-only data and instructions are always cacheable. However, data which can be modified is only cacheable within a cluster. With this approach, cache consistency does not need to be maintained through the multistage interconnection network and the consequent loss in performance can be minimized through careful consideration of data location by the operating system.

Each MULTIPLUS processing node has separate 64 Kbyte data and instruction caches. With the design simplifications just discussed, only the data cache controller needs to snoop the data bus. The data cache controller works in write-through mode, which is very simple 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. This is due to a number of reasons. Firstly, when write-back mode is used, every write operation which results in a cache miss must be done for the whole cache block size, which happens to be 32 bytes within MULTIPLUS. Therefore, such write operations require 4 data bus cycles. In addition, if they are between two clusters, the multistage interconnection network has to transmit a 32-byte data block. Secondly, whenever a read operation generated by a processing node refers to shared data which happens to have been altered in some external data cache, the whole external cache block needs to be written back to memory before the read operation is performed. Finally, the use of 64-bit write buffers reduces in approximately 40% the time spent on write operations when write-through mode is used. For the write-back mode, the write buffers would have to be four times bigger since all write operations to memory are performed in 32 bytes.

The simulation experiments have also shown that the use of the data bus by the processing nodes is much more intense than the use of the instruction bus, since the hit rate of instruction caches is significantly higher than that of the data caches. Because of that, the instruction bus is also used for data block transfers which occur in I/O or in memory page migration or copy operations. The use of the instruction bus for these operations cannot cause any cache consistency problem since the operating system flushes all cache positions occupied by data which are to be overwritten by block transfers.

4. THE MULTISTAGE INTERCONNECTION NETWORK

The MULTIPLUS multistage interconnection network is an inverted n-cube network consisting of 2x2 cross-bar switching elements.

This network topology provides the MULTIPLUS architecture with two very desirable features: modularity and partitionability. The modularity provided by this network 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 made up of any power of two of clusters such that the communication within a group does not interfere with the communication within any other group of clusters.

The MULTIPLUS multistage network can support up to 256 clusters. Each communication path between switching elements in the newtork is unidirectional and eight bits wide. The transmitted messages can have variable length up to a maximum of 128 bytes. Wormhole routing [SIEG89] 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.
Simulation studies have been carried out to compare the network performance for three alternative architectures [TAMI88] of the newtork switching element: the use of a single buffer at each input, the use of a single buffer at each output or the use of a double buffer at each input and a multiplexor at the output. In the simulation experiments, the network performance is measured through three parameters: the average delay for a message to go through the network, the average newtork throughput and the average occupation of the buffers in the network.

Both a functional and an analytical [YOON90] model have been used in the simulation experiments which have produced similar results for both models. The simulation experiments have been carried out under the maximum performance conditions proposed by Dias [DIAS81]: a new message is ready to be sent through the network at each input in every cycle and a message can always be removed when it reaches a network output. The main conclusions which can be derived from the simulation results are the following ones:

. For a wide range of values, the size of the buffers does not affect significantly the network throughput for all three configurations of the switching element;

. The switching element architecture consisting of double input buffers and an output multiplexor leads to the highest network throughput;

. The switching element architecture consisting of single output buffers is the one which presents the smallest network average delay. In addition, for this architecture, the network average delay does not depend on the buffer size in practice. The biggest network average delay results when switching elements

with single input buffers are used.

From these results, the architecture based on the use of single output buffers has been selected for the implementation of the network switching element for two main reasons. It produces the best result for the newtork average delay and it is less expensive to implement than the architecture which uses double input buffers and an output multiplexor, particularly because it has the lowest buffer occupation ratio.

5. THE MULPLIX OPERATING SYSTEM

MULPLIX is a UNIX-like operating system which is under development at NCE/UFRJ to support medium-grain parallelism and an efficient environment for running parallel scientific applications within MULTIPLUS. In its initial version, MULPLIX will result from extensions to PLURIX, a UNIX-like operating system, developed at NCE/UFRJ [FALL89], which supports multiprocessing.

PLURIX main goal is 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, assumes that the underlying machine is implemented by a Uniform Memory Access architecture and does not provide any special facility for scientific applications.

Considering the MULTIPLUS environment, such features are undesirable. Therefore, in its initial version, MULPLIX tries to overcome these limitations and extends or alters some of the policies and facilities implemented within PLURIX.

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 MULPLIX definition is the concept of thread [ACCE86]. 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 the synchronization primitives used by the Plurix kernel: busy-waiting binary semaphores, preemptible binary semaphores,

generalized sempahores and events. In addition, MULPLIX implements the busy-waiting primitives in a different way, since within the MULTIPLUS architecture 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: 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 not allowed. According to LeBlanc [LEBL89], an operating system for a NUMA multiprocessor with a large number of processors must allow memory sharing and reduce the impact of the differences in access times. Therefore, 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 node; 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 node; and definition of stack segments in the user and supervisor modes for each thread.

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 interruptions 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. It should be stressed that, as a nice side effect, the use of separate queues also reduces the traffic through the bus systems and interconnection network, which is specially important to the MULTIPLUS architecture.

6. CURRENT STATUS AND PERSPECTIVES

The MULTIPLUS architecture definition and detailed logic design have just been completed. Currently, we are starting to work in the implementation of an initial prototype with at least 16 processing nodes. In this initial version, the processing nodes will use SPARC chipset from Cypress Semiconductor Corp. which is able to deliver around 25 MIPS and 6 MFLOPS at 40 Mhz.

In parallel, we are undertaking the design of our own CMOS SPARC microprocessor chip which will be used at a later stage within the MULTIPLUS processing nodes. The design has gone through the simulation of several alternatives to the implementation of the SPARC architecture, definition of an architecture implementation and detailed logic design. Currently we are working on the chip layout using ES2 CMOS 1.2 micra design rules.

The implementation of the MULPLIX initial version as an evolution of PLURIX has just been started. SUN Sparcstations are in use as our main development platform together with Brazilian made computers which run under Plurix. The MULTIPLUS prototype must be running under the MULPLIX initial version by the third quarter of 1991. By this time, we also hope to have completed the adaptation to the MULTIPLUS environment of a C compiler developed at NCE/UFRJ within Plurix.

Hopefully by 1992, MULTIPLUS will be available for use by other research groups at NCE which are currently involved with work in several areas that may benefit from the MULTIPLUS computing power and parallel environment: artificial intelligence, image and signal processing, robotics, performance modelling of complex systems, neural networks, algorithms, electric simulation of VLSI circuits, etc. 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 authors would like to thank FINEP, CNPq and NCE/UFRJ, Brazil for the support given to the development of this research work.

REFERENCES

[ACCE86] Accetta, M. et al., "Mach: A new kernel foundation for Unix development", Proceedings of the Summer 1986 USENIX Technical

Conference and Exhibition, pp. 93-112, Summer 1986
[ANDE90] Anderson, T.E., "The performance of spin lock alternatives for shared memory multiprocessors", IEEE Transactions on Parallel and Distributed Systems, vol. 1, no. 1, pp. 6-16, January 1990
[DIAS81] Dias, D.M., Jump, J.R., "Packet switching interconnection networks for modular systems", IEEE Computer, vol. 14, no. 12, pp. 43-54, December 1981
[FALL89] Faller, N., Salenbauch, P, "Plurix: A multiprocessing Unix-like operating system", Proceedings of the 2nd Workshop on Workstation Operating Systems, IEEE Computer Society Press, Washington, DC, USA, pp. 29-36, September 1989
[TAMI88] Tamir, Y. Frazier, G.L., "High-performance multi-queue buffers for VLSI communication switches", ACM Computer Architecture News, vol. 16, no. 2, May 1988
[SIEG89] Siegel, H.J. et al., "Using the multistage cube network topology in parallel supercomputers", Proceedings of the IEEE, vol. 77, no. 12, pp. 1932-1953, December 1989
[STEN89] Stenstrom, P., "A Cache consistency protocol for multiprocessors with multistage networks", ACM Computer Architecture News, vol. 17, no.3, pp. 407-415, June 1989
[SUN 87] "The SPARC Architecture Manual", Sun Microsystems Inc., October 1987
[YOON90] Yoon, H. et al., "Performance analysis of multibuffered packet-switching networks in multiprocessor systems", IEEE Transactions on Computers, vol. 39, no. 3, March 1990

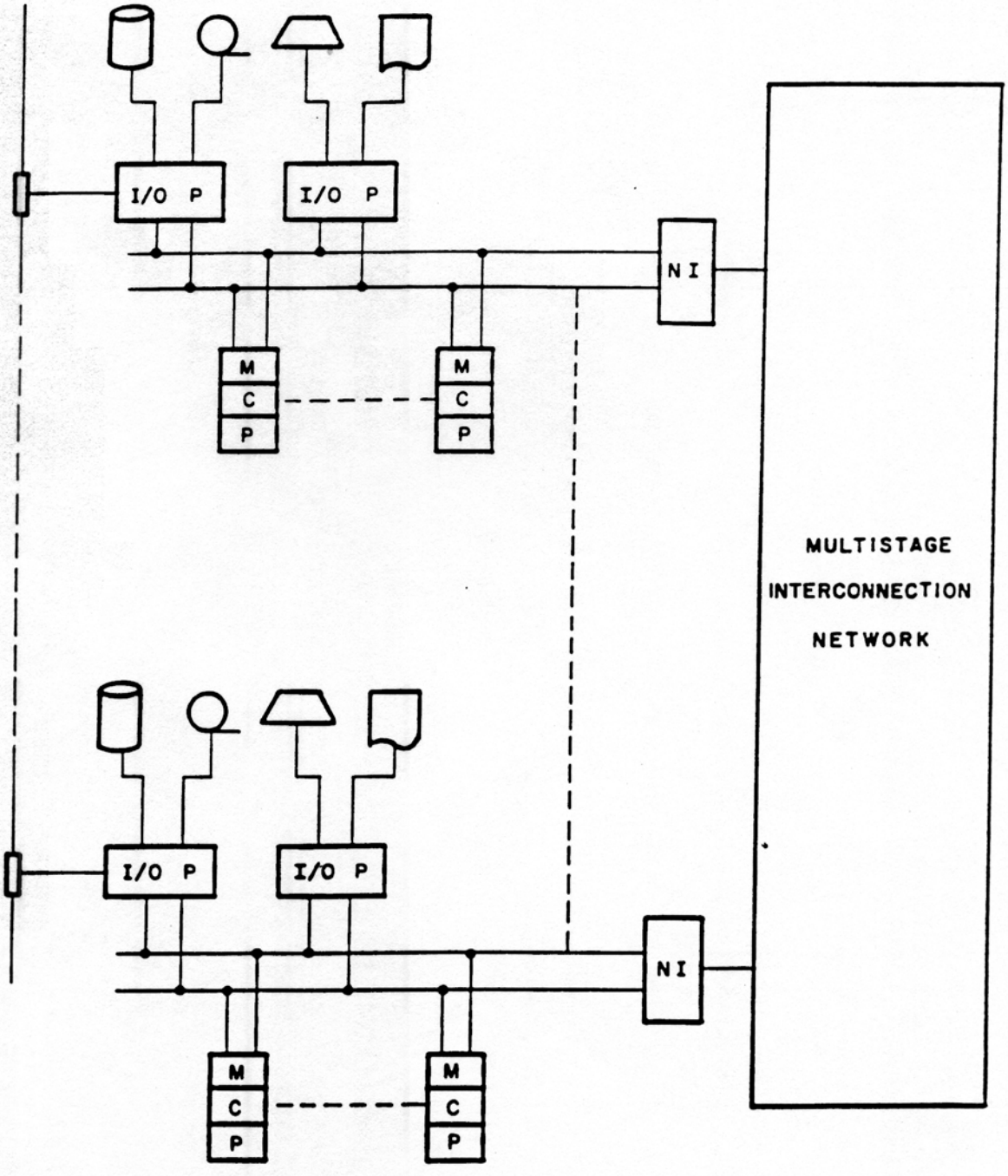

FIGURE 1: MULTIPLUS ARCHITECTURE