RELATÓRIO TÉCNICO

A COMPARATIVE ANALYSIS OF CACHE MEMORY ARCHITECTURES FOR THE MULTIPLUS MULTIPROCESSOR

Alexandre Malheiros Meslim
Ageu Cavalcanti Pacheco Jr.
Júlio Salek Aude
NCE/UFRJ

NCE — 08/92
Julho

Núcleo de Computação Eletrônica
Universidade Federal do Rio de Janeiro

Tel.: 598-3212 - Fax.: (021) 270-8554
Caixa Postal 2324 - CEP 20001-970
Rio de Janeiro - RJ

Uma Análise Comparativa de Arquiteturas de Memórias Cache para o Multiprocessador MULTIPLUS

Este trabalho analisa algumas alternativas de projeto para a arquitetura do sub-sistema de memória cache para o multiprocessador MULTIPLUS. O MULTIPLUS é um multiprocessador de alto desempenho em desenvolvimento no Núcleo de Computação Eletrônica da Universidade Federal do Rio de Janeiro (NCE/UFRJ). A análise foi realizada utilizando-se um simulador que suporta diferentes configurações de memórias cache. A simulação foi realizada utilizando-se três diferentes sistemas: sem memória cache, com cache utilizando políticas de controle do tipo *write through* e *write back*. Os resultados gráficos mostram o desempenho do sistema em relação a taxa média de ocupação dos barramentos e o tempo médio de duração do ciclo do processador.

A Comparative Analysis of Cache Memory Architectures for the MULTIPLUS Multiprocessor

This paper analyses some design alternatives for the MULTIPLUS cache memory subsystem architecture. MULTIPLUS is a high performance multiprocessor system under development at NCE/UFRJ. The analysis is carried out using a simulator which supports different cache configurations. The simulator experiments have been done under three different situations: a non-cache system and the use of write back and write through control policies. The graphical results show the system behaviour in relation to the average ratio of bus occupation and the average processor cycle length.

A Comparative Analysis of Cache Memory Architectures for the MULTIPLUS Multiprocessor

Alexandre Malheiros Meslin, Ageu Cavalcanti Pacheco Jr., Júlio Salek Aude

Núcleo de Computação Eletrônica - Universidade Federal do Rio de Janeiro
PO Box 2324
20001 - Rio de Janeiro - RJ - Brazil

This paper analyses some design alternatives for the MULTIPLUS cache memory subsystem architecture. MULTIPLUS is a high performance multiprocessor system under development at NCE/UFRJ. The analysis is carried out using a simulator which supports different cache configurations. The simulator experiments have been done under three different situations: a non-cache system and the use of write back and write through control policies. The graphical results show the system behaviour in relation to the average ratio of bus occupation and the average processor cycle length.

1 - Introduction

It is widely known that a careful specification of the cache subsystem plays a fundamental role in achieving the final performance targets in the design of a multiprocessor system. This paper analyses, through functional simulations, some alternatives for the MULTIPLUS cache memory subsystem. These simulations are specially indicated in this case due to some combinations of certain architecture features introduced in the present development.

MULTIPLUS is a high performance MIMD multiprocessor system bearing modular architecture under development at NCE/UFRJ. It is described in section 2. Section 3 discusses the simulator operation and the cache control schemes which are supported by the simulator. In section 4, the input variables for the simulator are presented and commented. In section 5, the simulation results are given in graphical form and its main aspects are commented.

2 - The MULTIPLUS Multiprocessor Architecture

The MULTIPLUS System [AUDE91] is a high performance multiprocessor with modular architecture which is able to support up to 2048 processing nodes (PN). Each PN is designed around a Cypress chip-set SPARC architecture. This set comprises a 32-bit processor [SUN87], a floating point processor, separated data and instruction direct mapped cache memories (64 Kbytes/each), up to 32 Mbytes of "local" memory and external bus interface. Up to 8 PNs can be connected by a 64-bit double-bus system making up a cluster. One of the busses is specific for data transfers while the other is for instruction transfers. Up to 256 clusters can be connected by a N-Cube multistage network.

Each PN local memory is in fact just a fraction (a segment) of the whole global system memory, and can be accessed by any other PN as its own memory. The software recognizes the system memory as one single block with up to 32 Gbytes.

The MULTIPLUS I/O subsystem is based on a distributed architecture. Two I/O processors (IOP) are associated with each cluster: one of them is block oriented, controlling disk and tape I/O operations and the other is byte oriented, controlling printer and display I/O operations.

A MULTIPLUS configuration with 4 clusters and 4 PNs within a cluster is presented in fig. 1.

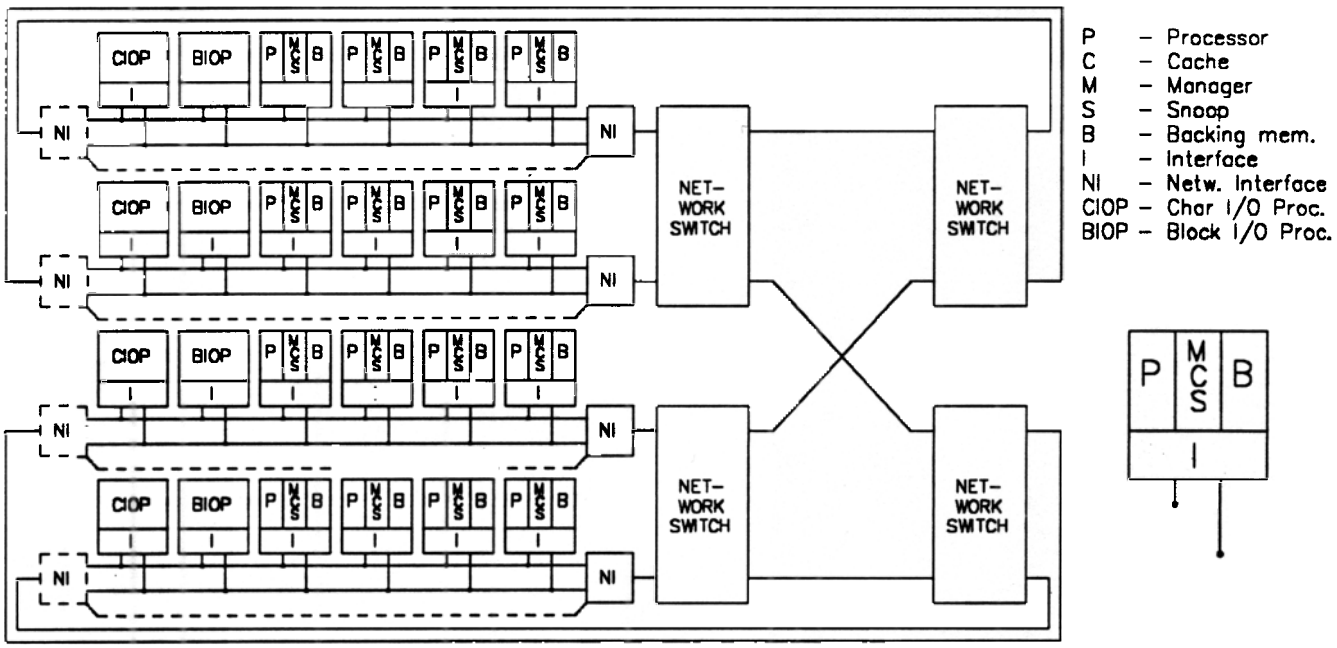

Fig. 1 - MULTIPLUS System block diagram

2.1 - MULTIPLUS Cache Coherence

The MULTIPLUS architecture allows PN*i* to access its own local memory without using the external busses. At the same time, another PN*j* can access PN*i* memory storing the data into its cache memory. A coherence problem in the data base may be caused when PN*i* writes to its local memory at a address stored in PN*j*'s cache. This problem may occur when write back or write through schemes are used.

Another coherence problem that may occur is when backing memory is not always updated during processor write cycles due to the use of write back coherence scheme. That is when one processor reads one position of local memory data that happens to be stored and written (modified) at another processor cache, the data read by the first processor is not the most recent one.

Some multiprocessor systems solve this coherence problem by not allowing remote data to be accessed by other processors, as in the New York Ultracomputer (NYU) [GOTT83]. Others, like the IBM Research Parallel Processor Prototype (RP3) [PFIS86], do not allow remote data to be stored in cache memory.

Within the MULTIPLUS architecture, we would like to share local memory among all processors in all clusters, to maximize storing data in cache memory, and, at the same time, to minimize the use of external busses.

In order to achieve these goals, we have decided to split the burden of maintaining cache memory coherence between software and hardware within the MULTIPLUS architecture. A software scheme works on data accessed via the multistage interconnection network (inter cluster transactions) and the hardware treats accesses within a cluster.

The hardware cache coherence is based on the data bus snoop. This coherence scheme can only be used when the bus operation involved is performed within a cluster, where all PNs are strongly coupled. The cache controller is able to continuously snoop the data bus in order to maintain the stored data status always updated.

2.2 - Remote Memory and Data Cache Coherence

Data transfers between a processor and its local memory are not normally snooped by other PN cache controllers, since external data busses are not used. On the other hand, for write back schemes it is important to snoop all data transfers because any type of operation can change the related cache data status. Fortunately, there is one algorithm that can distinguish between data accesses that WILL NOT change other PN cache data status from others that may or may not change the status. With this algorithm, it is possible for some data access to be done

inside a PN, without causing any incoherence problem and without requiring software intervention.

The algorithm is based on the use of a flag near each local memory which indicates if a data block in local memory can be shared by others PNs. There is one flag bit for each cache block mapped onto the local memory. This flag shows if its associated memory block was already accessed by other PNs inside the same cluster. Non *cacheable* data accesses do not change this flag status.

During the power on sequence, this flag is **RESET.** Coherent reads and writes from a remote cache controller, performed within a given cluster, change the accessed block flag status to **SET** meaning that this line may now be shared. The status flag is **RESET** when the local cache controller accesses this memory block.

All local memory accesses can be done without the use of external busses, except for data accesses with the block shared flag set.

Most of network coupled systems [BBN85] [PFIS86] do not allow shared data to be stored in cache. This obviously simplifies data coherence maintenance, as backing memory is always updated and there are not data copies in other caches. Within MULTIPLUS, the adopted approach is to extend cache store possibility to the greatest amount of variables at a small expense. The decision was to restrict any cache memory to only store read/write data sitting in a memory block within the same cluster.

The operating system is responsible for maintaining all memory management tables in control of the pages that can or cannot be stored in cache memory. It is also responsible for maintaining mutual exclusion of migratory variables.

Coherence is naturally maintained during instruction accesses by not allowing memory code to be modified.

3 - The Simulator Operation

A simulator has been developed to analyse different MULTIPLUS architecture implementations, specially for the MULTIPLUS cache memory subsystem. This simulator is able to verify the performance of different architecture alternatives which include the number of external busses in use (1 or 2), the adopted cache control schemes (without cache, write through and write back) and the use of write buffer. The simulator has been written in PASCAL. Its source code has around 6000 lines and, at the moment, it is running on IBM-PC compatible microcomputers.

In order to fully understand the simulator operation, some concepts used in the following sections need to be clearly defined.

. **RESOURCE:** it is anything necessary for the processor to access the desired memory data. Resources that are always allocated

together are called by the same name, ie, the cache controller snoop and the internal bus physically connected to the cache controller are called SNOOP.

. **PROCESSOR:** it is the MULTIPLUS Integer Unit (IU) which sits on every processing element.

. **STATUS:** it is assigned to each one of the processors and resources. Three status are used: **FREE, W** (Wait - related only to a processor and having various sub-status indicating that the processor is waiting for free resources) and **X** (eXecuting - which has also various sub-status. It is related to the preliminary **W** status). **W** and **X** sub-status are strong related to the type of access the corresponding processor is assigned to execute.

3.1 - Processor Allocation

Simulation begins with all the processors in the **FREE** state, which is the condition established by the power on sequence. One type of memory access, basically data or instruction access, is allocated for each processor. Although, at power on, in a real system, the cache memories would be empty and the first memory access would be a code fetch, in the present simulation, the machine begins at steady state with a populated cache memory and a stabilized cache hit rate to avoid transient problems and minimize the simulation time needed for the generation of reasonable results. Similarly, the shared flag and the type of processor access are also in the steady state at the beginning of the simulation. Accesses assignment is made only randomly according to the input from the simulator user.

3.2 - Resource Allocation

The available processor resources are:
. Inside a PN: code cache, data cache, data snoop, local memory,
. Inside a cluster: code bus, data bus, other PNs local memories, other PNs data snoops,
. Outside a cluster: code bus, data bus, local memories and data snoops.

For systems with only one bus, the data bus is also the code bus and, therefore, this single bus is called BUS.

Each resource has two attributes. The first one shows its current status that can be **FREE, X** or **W**. The second one, which is valid only when the first one is **X**, is the time needed for the processor (or resource) to finish the current access.

3.3 - Simulator Flow

The simulator operation consists of three major parts: initialization, running and counting. The running part is the simulator kernel and comprises four routines:

. **Job Allocation Routine:** it allocates for each FREE processor one job (one type of memory access) with random distribution throughout a random number generator from 0%, inclusive, up to 100%, exclusive. This number is compared with the number set by

the user at the beginning of the simulation. This roughly works as the sequence of decisions shown below:

IF random < code fetch rate **THEN**
    type of code fetch selection routine
ELSE IF random < data read rate **THEN**
    type of data read selection routine
ELSE
    type of data write selection routine

When the simulator leaves this routine, all processors have status **W** (closely allocated) or **X** (already doing memory access).

. **Resource Allocation Routine:** once all the processors have what to do, every processor in **W** state will need some resources to execute its assigned memory access according to its status. Processor status only change from **W** to **X** when all needed resources are already allocated. Once the resources are allocated, the resource status changes from **FREE** to the corresponding **X** status of its master processor. Its time counter is loaded with the access corresponding amount of time.

. **Time Routine:** this routine emulates the simulator elapsed time. It decrements the time counter for each processor and resource in **W** state running an access. For every counter that hits ZERO, this routine changes the respective processor or resource status to **FREE**.

. **Partial Totalization Routine:** there are auxiliary counters in each one of the above described routines that are updated here for partial totalization and displays.

The main totalization routine is called at the end of the simulation. It evaluates all partial results and optionally saves them in an ASCII file for subsequent plotting.

3.4 - Simulator Cache Schemes

There have been implemented three different cache schemes: without cache, write through and write back. The without cache option was simulated only to show the strong influence the high hierarchical memory has on the MULTIPLUS performance.

. **Write Through:** it has been implemented without write allocate as Cypress does in its cache controller CY7C605 [CYPR90].

Some restrictions are required in order to maintain data coherence. Data which is only accessible through the network (extra cluster data) is not *cacheable*. Code is always *cacheable* and the code fetch is done in coherent mode because the coherence is maintened by software (by the operating system).

Local memory access can be done without the use of external busses, except coherent data write with the block shared flag set. In this case, the memory access is done using the external data bus to update (block invalidate) other PN cache status inside the same cluster.

. **Copy Back:** implemented as in the Cypress CY7C605 [CYPR90] cache controller with write allocate. All restrictions applied to the write through scheme are expanded to all data accessed by any processor outside the cluster, even if the data is at the local memory.

When a processor **i** requires data which is not present in its own cache memory, such data must be read from the backing memory. If there is another copy of this same data in another cache memory **j** (**j** ¹ **i**), the data must be stored as **SHARED**, otherwise, it is stored as **PRIVATE**.

Future memory references to data **i** will produce a hit in the cache and the access will be performed without the need of any external busses. Cache hit data accesses do not need external busses, except for data write to a data cache with the **SHARED** status on. In this case, it is necessary to use the external bus to inform other cache controllers that this cache block status has changed (from **SHARED** to **MODIFIED**).

Writing to a **PRIVATE** block changes its status to **MODIFIED**. Writing to a **SHARED** block needs an external bus invalidation access to invalidate all other cache block copies.

When a processor **j** performs a remote data write which produces a hit at a snoop cache **i**, its block cache status is changed to **INVALID**. If it is a read memory access, the status changes to **SHARED**. Otherwise, if the initial status is **MODIFIED**, the cache controller **i** aborts the current memory cycle and performs a backing memory update (copy back), allowing processor **j** reads or writes to the most recent value (in backing memory).

A processor only needs to use the external data bus during an access to its local memory if a coherent data access is to be performed and the memory block shared flag is on.

4 - User Input Parameters

The simulator input parameters will be presented in this section with their respective default values and some explanations.

The input parameters are divided into two groups: hardware parameters, concerned only with hardware implementation aspects, and system parameters, concerned with cache schemes, memory hierarchy and, mainly, the workload.

4.1 - Hardware Parameters

There are three types of hardware parameters: timing, configuration and architectural.

The architectural parameters allow the simulation to configure a system with 1 or 2 busses connecting from 1 to 16 PNs per cluster. The system can have from 1 to 256 clusters and work with a backing memory write buffer.

The timing parameters are concerned with the access time in all levels of the memory hierarchy. The timing for local memory access is divided into two parts for the simulation of burst memory accesses.

All timings in the simulator are quantified in clock cycles with a clock frequency of 25Mhz, giving a period of 40ns

. **Network Access:** the delay for each network stage is about 2 clock cycles [BRON90]

. **Bus Access:** the bus access is divided in three parts [CYPR90]: bus arbitration phase, address phase and data phase. The bus arbitration phase has a non deterministic delay and is simulated by the system arbiter routine included in the simulator. The address phase has a fixed time (1 clock cycle or 40ns). The time needed for the cache controller to decode and initiate a memory access inside its own PN (three clock cycles) is added to this time, making up four clock cycles. The data phase timing is given by the slave (backing memory) and the transfer length.

. **Local Memory Access:** this time depends on how fast is the memory used for the backing memory. Assuming the use of an 80ns memory, both the access time and the burst time can be set to 2 clock cycles.

. **Cache Access:** the data read and code fetch accesses take 1 clock cycle. Data write accesses take 2 clock cycles

4.2 - System Parameters

The system parameters are extremely workload dependent. They are very difficult to be estimated. These parameters are the following ones:

. **Cache Hit Rate:** since most MULTIPLUS expected workload is scientific, where big data blocks are used in various iterations, favoring cache hit rate, we set, initially for the experiments, the data cache hit rate at around 85%, and the code cache hit rate at 95%

. **Snoop Hit Rate:** the snoop hit rate represents the rate of memory accesses that will invalidate a cache block. This rate was set to 10% of the amount of coherent memory accesses with invalidate command. Although Weber [WEBE89] has assumed this rate as 1%, we decided to use 10% due to the high shared variable rate (see below).

. **Shared Rate:** it is the amount of processor memory accesses which refer to shared data. Weber [WEBE89] found a rate of 2% while [FEIT90] and [RUDO85] found 5%. Both of the analysed systems in those works had shared memory connected only by a network and not by busses. In order to simulate the existence of a distributed operating system with a lot of shared variables and a non favorable workload, we have decided to use a high value such as 20% for code and data share.

. **Local Access Rate:** it is the rate of processor local accesses (inside the PN). Its value was set to 80% [AZEV91].

. **Cluster Access Rate:** it is similar to the local access rate. It gives the rate of memory accesses inside a cluster for each PN. Its value has been estimated to be 80% of the non-local memory accesses.

. **Code Fetch, Data Read and Data Write Rates:** they are responsible for the most important differences between simulations of RISC and CISC processors. While CISC rates are around 50%, 15% and 35%, respectively [SMIT82], the RISC rates are 80%, 13% and 7% [NAMJ88] [TAMI81], and those are the values used here.

. **Written Rate:** it is the rate of modification on data stored in the cache. Its value has been estimated as 10% [MESL91].

5 - Results

The simulation results are shown below.

. **External Bus Busy Rate:** we can notice from figs. 2 and 3 that the bus busy rate increases with the number of PNs/clusters and it is quite independent on the number of clusters. This is one of the advantages obtained from the present network design [BRON90].

The null bus busy rate for 1 PN/cluster and 1 cluster observed in figs. 2 and 3 is due to the non existence of an external bus (since the IOPs were not included in the present simulation). The code bus busy rate is the same for write back based systems or write through based systems since there is no write bus operations at the code bus and, therefore, no copy back is required. With a high number of processor elements per cluster (high share degree), the data bus busy rate is quite similar for both the write back and the write through schemes.

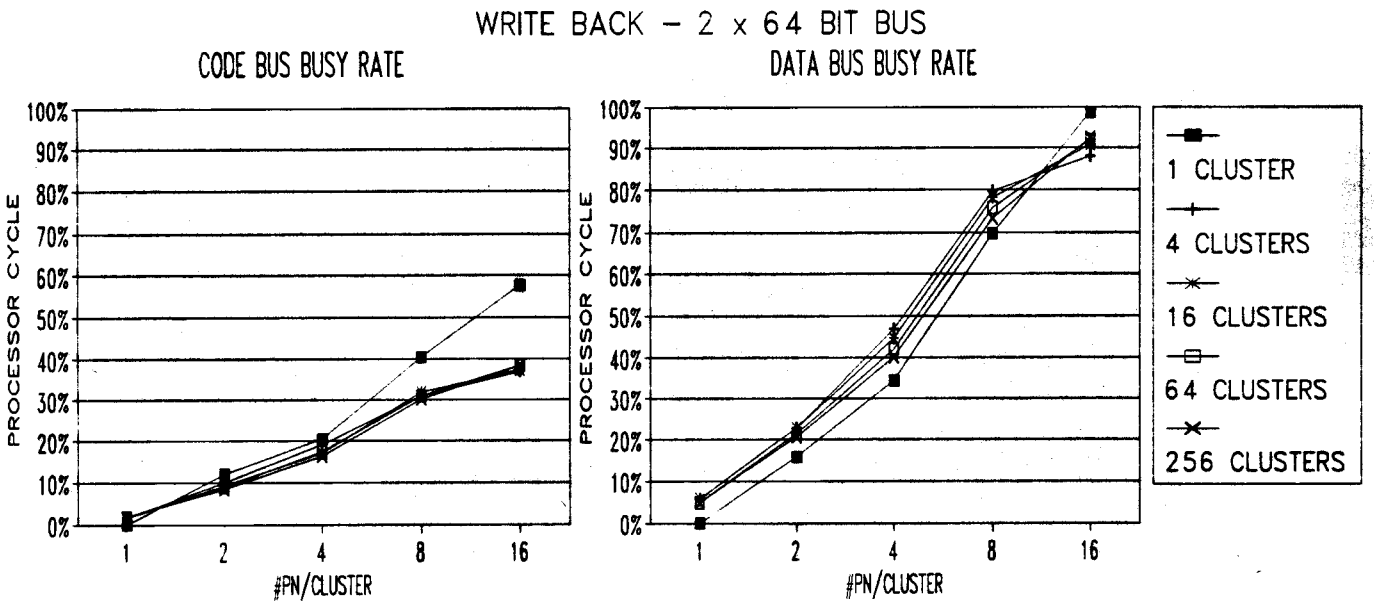

Fig. 2: write back code (a) and Data (b) bus busy rate.

The curves in figs. 2.b and 3.b seem to confirm an early observation [SMIT82] which suggests that a well implemented buffered write through policy gives similar performance as the write back scheme.

. **Average Processor Cycle:** we can see from fig. 4.a and 4.b that there is some independence between the average processor cycle and the number of system clusters. But the system performance goes down as the number of PNs/cluster increases. This situation is due to the low network traffic and, perhaps, to the inaccuracy of the network simulation, which neglects the network contention. Still from figs. 4.a and

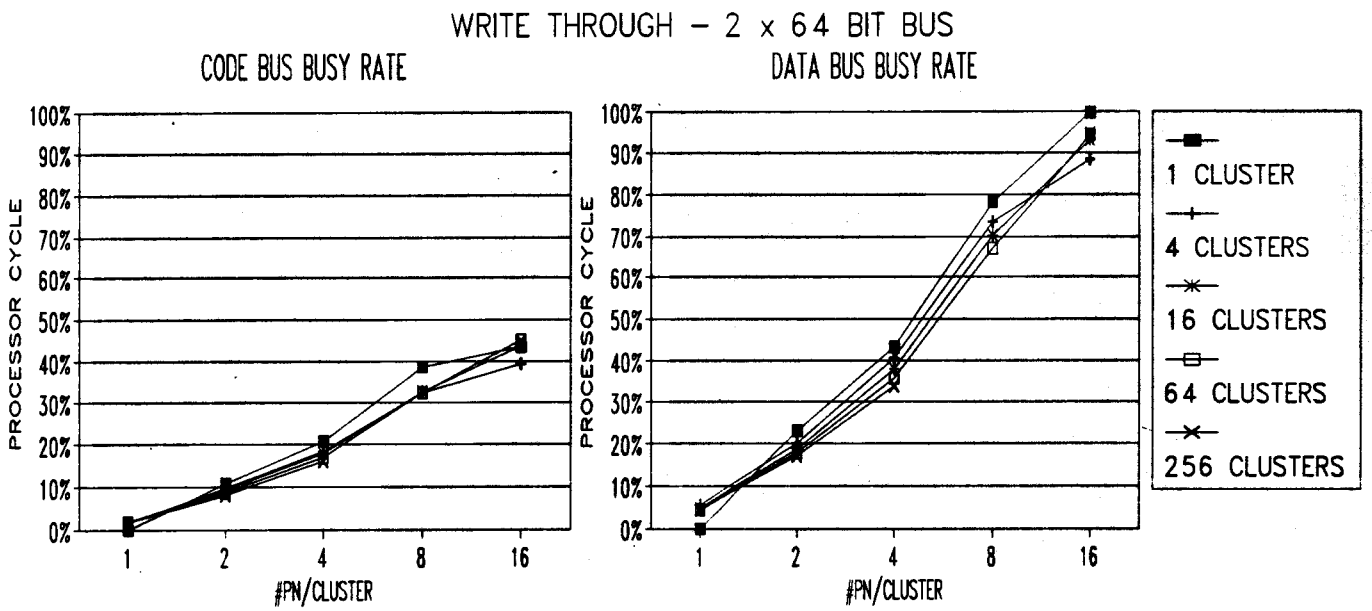

Fig. 3: write through with write buffer code (a) and data (b) bus busy rate.

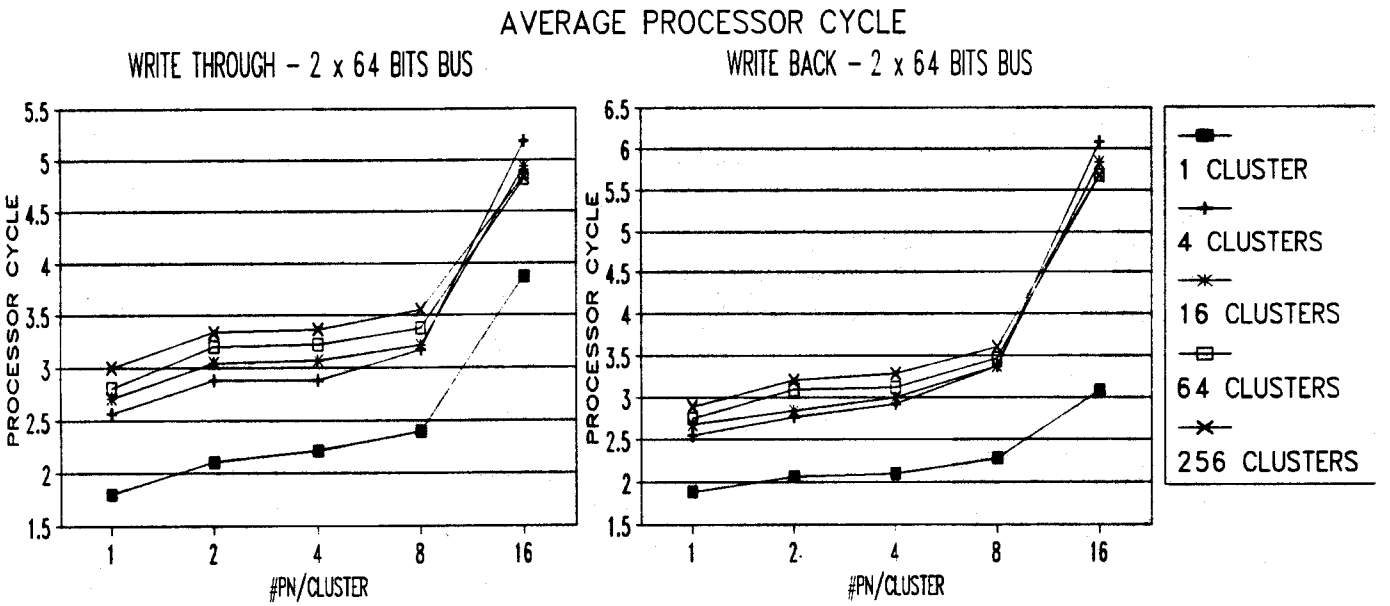

Fig. 4 - write through average processor cycle: (a) double bus (b) single bus

4.b we can see that a system configuration with 16 PNs/cluster is quite saturated. Therefore, its use is not recommended.

Also, the write back system is around 20% better than the write through when a small number of PNs and clusters is used. But its superiority goes down as the number of PNs increases.

The system configuration using write through with write buffer has its performance improved from 20% up to 25% as shown in fig. 5. Also, it seems that there is no advantage in having a write back system with write buffer. Perhaps, this is due to the fact that the write buffer option in our experiments can only store one memory access at a time, while the write back scheme needs four memory accesses to perform a block write.

Fig. 5 shows a MULTIPLUS system with 16 clusters and 4 PNs/cluster. We can see that for small values of data sharing, the write back scheme is much better than the write through. As the sharing rate increases, the write through with write buffer scheme shows a better performance than the write back approach. It can also be noticed that with the write through scheme the system performance is somewhat independent on the sharing rate.

Fig. 6 shows some configurations of the MULTIPLUS system:

a) write through with write buffer, double 64-bit bus

b) same as above, with I/O burst (2nd Y)

c) write through double 32-bit bus

d) write through with write buffer double 32-bit bus

e) write through single 64-bit bus

f) write through double 64-bit bus

6 - Conclusions and Future Work

The simulation analysis has shown that the write back scheme outperforms, though marginally, the write through scheme. However it has also been verified that when the sharing rate increases, its performance is impacted by the large amount of invalidation accesses and cache block copy back operations.

In addition, it could also be noticed that a write through system with local memory write buffer presents a performance which is very close to the one delivered by the write back system.

The next steps in the development of the simulator will be the inclusion of a more realistic model of the N-cube multistage network which will be able to take into consideration the contention effect, delivering more accurate results.

We expect to have a prototype system by the end of the year to compare the simulation results with the ones produced by the real system.

7 - Acknowledgements

We would like to thank CNPq and FINEP that sponsored this work.

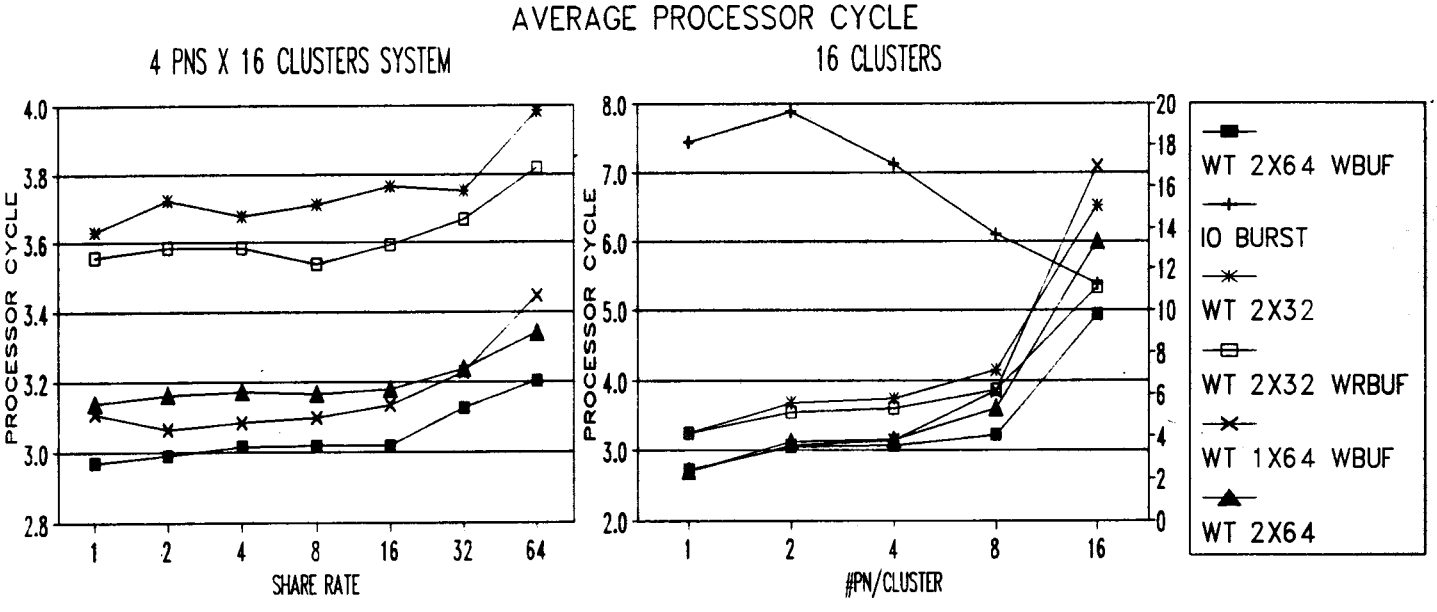

Fig. 5 - average processor cycle X sharing. Fig. 6 - general system performance.

References

[AUDE90] J. S. AUDE et al, "MULTIPLUS: A Modular High-Performance Multiprocessor", **Proceedings of Euromicro 91**, Viena, pp 45-52, Sep/91

[AZEV91] G. P. AZEVEDO et al, "MULPLIX: Um Sistema Operacional Tipo Unix para o Multiprocessador MULTIPLUS", **Technical Report RT 91-1 - Núcleo de Computacao Eletronica - Universidade Federal do Rio de Janeiro**, Jan/91

[BBN85] "Butterfly Parallel Processor Overview", **BBN Laboratories Incorporated**, version 1, 17 pp, Jun/13/85

[BRON90] G. BRONSTEIN et al, "Analise do Desempenho de Redes de Interconexao para Maquinas Paralelas", **III Simposio Brasileiro de Arquitetura de Computadores/Processamento Paralelo**, pp 345-360, Rio de Janeiro - RJ, Nov/7-9/90

[CYPR90] "SPARC RISC USER'S GUIDE", **Cypress Semiconductor Corporation**, 2nd edition, Feb/1990

[FEIT90] R. Q. FEITOSA, "O Problema de Coerencia de Memorias Cache Privadas em Grandes Multiprocessadores para Aplicacoes Numericas: Uma Nova Solucao", **X SBC Congress** - Vitoria - ES - Brazil, pp 138-156, Jul/22-27/90

[GOOT83] A. GOTTLIEB et al, "The NYU Ultracomputer - Designing an MIMD Shared Memory Parallel Computer", **IEEE Transactions in Computers**, vol. C-32(2), pp 175-189, Feb/1983

[MESL91] A. M. MESLIN, "Estudos de Arquiteturas de Memorias Cache para o Multiprocessador MULTIPLUS", MSc Tesis, **COPPE - UFRJ**, Aug/91

[NAMJ88] M. NAMJOO et all, "CMOS Custom Implementation of the SPARC Architecture", **33th IEEE Computer Society International Conference**, pp 18-20, California, Feb 29 - Mar 04/88

[PFIS86] G. P. PFISTER et al, "An Introduction to the IBM Research Parallel Processor Prototype (RP3)", IBM T. J. Watson Research Laboratory, **Research Report**, 32 pp, Yorktown Heights, NY, 13/Jun/86

[RUDO85] L. RUDOLPH et al, "Dynamic Decentralized Schemes for MIMD Parallel Processors", **Proceedings of the 12th International Symposium on Computer Architecture - ACM SIGARCH Newsletter**, vol. 3(3), pp 340-347, 1985

[SMIT82] A. J. SMITH, "Cache Memories", **ACM Computer Surveys**, vol. 14(3), pp 473-530, Sep/1982

[SUN87] SUN MICROSYSTEMS INC, "The SPARC Architecture Manual", Mountain View - CA, 199 pp, 1987

[TAMI81] Y. TAMIR, "Simulation and Performance Evaluation of the RISC Architecture", **University of California, College of Engineering and Computer Sciences, Computer Science Division**, Mar/81

[WEBE89] W-D WEBER, et al, "Analysis of Cache Invalidation Patterns in Multiprocessors", **ACM SIGARCH Computer Architecture News**, New York, NY, vol 1 (2), pp 243-256, Apr/89