*** RELATÓRIO TÉNICO ***

SIMULAÇÕES DE ARQUITETURAS DE MEMÓRIA CACHE PARA O MULTIPROCESSADOR MULTIPLUS

Alexandre Malheiros Meslin

NCE 13/91
Setembro/91

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

Este artigo foi publicado nos Anais do XI Congresso da SBC, realizado em Santos/SP, de 05 a 09 de agosto/91.

SIMULAÇÕES DE ARQUITETURAS DE MEMÓRIAS CACHE PARA O MULTIPROCESSADOR MULTIPLUS

RESUMO

Este trabalho analisa algumas alternativas de arquitetura de sistemas de memórias *cache* para o MULTIPLUS, um multiprocessador de alto desempenho em desenvolvimento no NCE/UFRJ. A análise é feita através do uso de um simulador capaz de suportar diferentes configurações de arquitetura de memória *cache*. As simulações foram realizadas considerando 3 situações distintas: a ausência de memórias *cache* e o uso de políticas de *write back* e *write through* para controle da *cache*. Os resultados das simulações mostram de forma gráfica o comportamento do sistema em relação à taxa média de ocupação dos barramentos e duração média dos ciclos de processador.

CACHE MEMORIES ARCHITECTURES SIMULATIONS FOR THE MULTIPLUS MULTIPROCESSOR

ABSTRACT

This paper analyses some alternatives for the MULTIPLUS cache memory system 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 memory architecture configurations. The simulator experiments were done under 3 different situations: a non-cache system and the use of write back and write through cache control policies. The graphical simulation results show the system behaviour in relation to the average ratio of bus occupation and the average processor cycle length.

1 - Introdução

Este trabalho analisa, através de simulações funcionais, diversas arquiteturas de memórias *cache* para o multiprocessador MULTIPLUS. A necessidade de siulações para o sistema MULTIPLUS decorre de diversas inovações propostas para a sua arquitetura.

O sistema MULTIPLUS é um multiprocessador de alto desempenho com arquitetura modular que está sendo desenvolvido no Núcleo de Computação Eletrônica da Universidade Federal do Rio de Janeiro. A arquitetura do MULTIPLUS é descrita na seção 2 deste artigo.

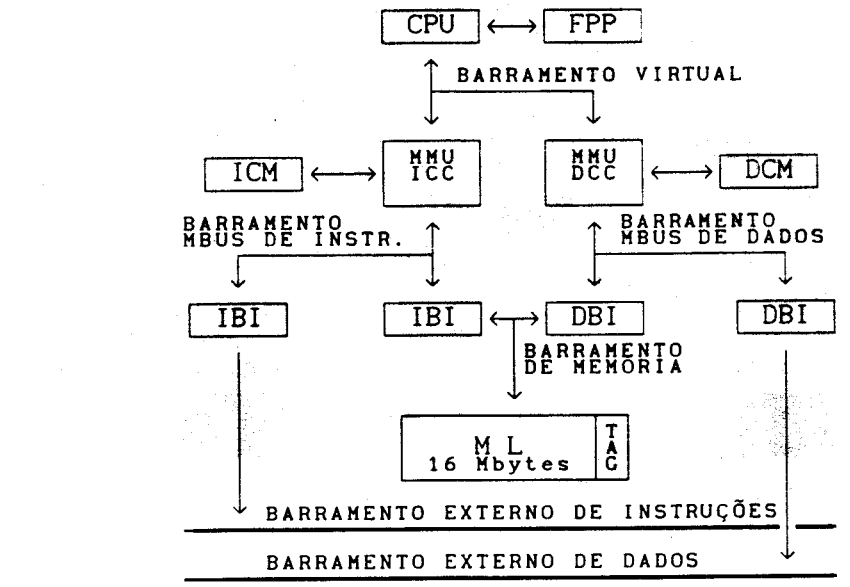
Na seção 3, é descrito o funcionamento do simulador. Ainda nesta seção, os modelos de política de *cache* implementadas no simulador são discutidos.

Na seção 4, os parâmetros de entrada necessários ao simulador são apresentados e os valores utilizados para simulação são comentados.

Na seção 5 são apresentados os resultados das simulações.

2 - Descrição da Arquitetura do Multiprocessador MULTIPLUS

O MULTIPLUS [AUDE90] é um multiprocessador de alto desempenho com arquitetura modular capaz de suportar até 2048 Nós de Processamento (NP). Cada NP (figura 1) contém um processador RISC de 32 *bits* com arquitetura SPARC [SUN87], um co-processador de ponto flutuante, 64 K*bytes* de memória *cache* para dado, 64 K*bytes* de memória *cache* para instruções, até 32 M*bytes* de memória do tipo RAM e interface para os barramentos externos.



LEGENDA:

CPU - Central Processor Unit
FPP - Floating Point Processor
ICM - Instruction Cache Memory
DCM - Data Cache Memory
MMU - Memory Management Unit
ICC - Instruction Cache Cont.
DCC - Data Cache Controller
IBI - Instruction Bus Interface
DBI - Data Bus Interface
ML - Memória Local

figura 1: Diagrama em blocos do NP do MULTIPLUS

Até 8 NP's podem ser interligados por dois barramentos de 64 *bits* de largura para dado e 36 *bits* de endereço formando um *cluster*. Os barramentos são especializados em transferência de instruções e transferência de dados.

Os barramentos conectam os NP's às interfaces da rede de interconexão multiestágio do tipo N-cubo invertido (RI). A RI pode suportar até 256 *clusters*.

A memória local de cada NP pertence ao espaço de endereçamento global do sistema, podendo ser acessada por qualquer outro NP. Para o *software*, a memória do sistema é composta por um único grande bloco de até 32 G*bytes*.

A arquitetura de E/S do MULTIPLUS é distribuída. Para cada *cluster* existem dois processadores de E/S (PES): um deles é orientado a blocos, controlando operações com discos e fitas e o outro é orientado a *byte*, realizando operações com terminais e impressoras.

A figura 2 apresenta a arquitetura total do sistema MULTIPLUS em uma configuração com 4 NP's por *cluster* e 4 *clusters*.

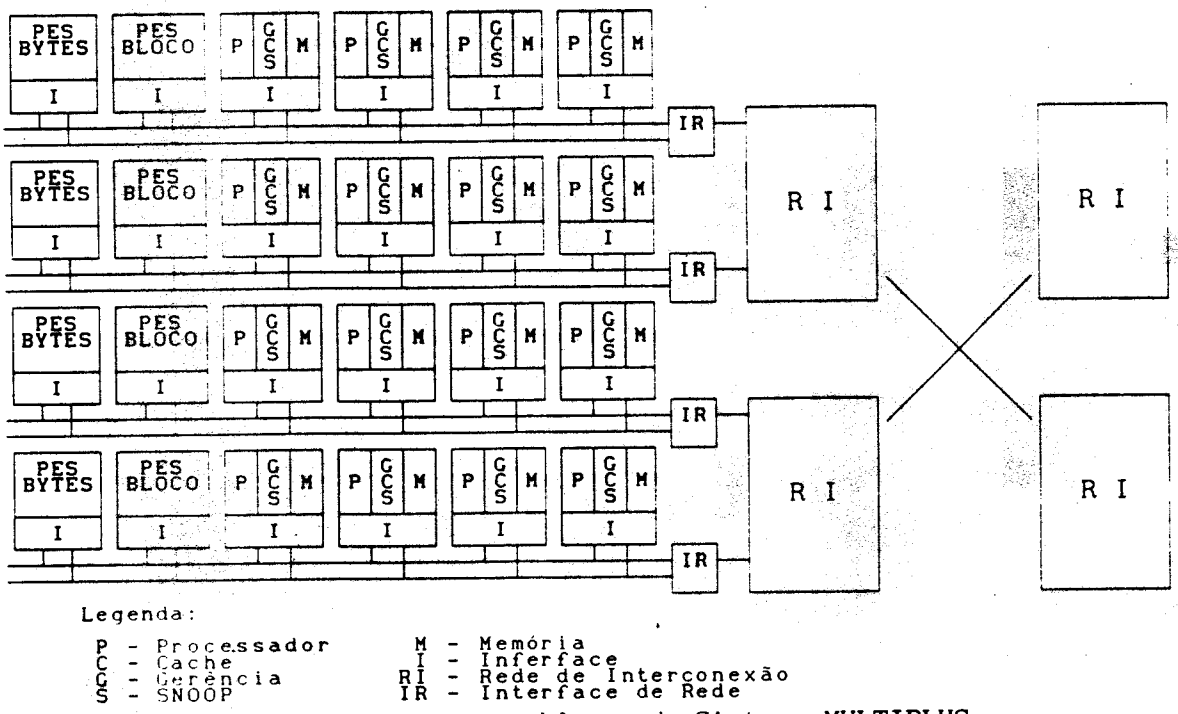


Figura 2: Diagrama em blocos do Sistema MULTIPLUS.

2.1 - Manutenção de Coerência de Cache no MULTIPLUS

A arquitetura de memória do MULTIPLUS possibilita que acessos do processador à sua memória local sejam realizados sem a necessidade de utilização do barramento externo. Como esta memória local, logicamente pertence ao espaço de endereçamento global, outros processadores também podem acessa-la, levando dados desta memória para a sua *cache* local.

A base de dados pode se tornar inconsistente quando um processador escrever na sua memória local um dado que esteja presente em um bloco armazenado em uma outra *cache* de um outro processador. Este tipo de inconsistência pode aparecer quando utiliza-se políticas de *cache* do tipo *write-through* ou do tipo *write-back*. Outro fator de inconsistência ocorre, em sistemas que utilizem políticas que não mantenham a memória local sempre atualizadas como no caso das políticas de *write-back e write-once*, quando um processador lê um dado armazenado na sua memória local cujo bloco tenha sido levado para outra *cache* de outro processador e este já tenha escrito neste bloco. Neste caso, o processador que possui o dado em sua memória local, ao buscá-lo, estará trazendo um dado antigo não atualizado.

Alguns sistemas multiprocessados resolvem este tipo de problema de

processadores sejam acessadas por outros processadores, como no caso do New York Ultracomputer .(NYU) da New York University [GOTTS3]. Em outros sistemas, como o Research Parallel Processor Prototype (RP3) da IBM [PFIS86], fazendo com que os dados pertencentes à memória local de um processador que devem ser acessados por outros processadores, não sejam armazenáveis em *cache*.

No caso do MULTIPLUS, optou-se por permitir que a memória local seja compartilhada por todos os processadores do sistema e por maximizar as possibilidades de armazenamento em *cache* de dados, e ao mesmo tempo diminuir ao máximo a utilização dos barramentos externos.

O controle da coerência da *cache* de dados na arquitetura do MULTIPLUS é dividida entre o *hardware* e o *software*. Exitem basicamente dois esquemas de manutenção de coerência, um para operações que utilizem a interface de rede e outro para operações estritamente dentro do *cluster*.

A manutenção da coerência por *hardware* é baseada na monitoração (*snoop*) do barramento de dados. Este tipo de manutenção de coerência somente é aplicável a operações de barramento que sejam realizadas dentro de um mesmo *cluster*, onde todos os NP's estão fortemente acoplados.

O controlador de *cache* utilizado possui um circuito de *snoop* que monitora continuamente as ações no barramento externo de dados mantendo o estado dos objetos contidos na *cache* sempre atualizados.

As operações entre um processador e sua memória local não são normalmente monitoradas por controladores de *cache* relativos a outros processadores, a não ser que estas sejam realizadas via barramento externo. Por outro lado, o esquema de *write-back* obriga que as operações sejam monitoradas, pois todas as operações de memória podem mudar o estado dos dados relativos a esta operação que por ventura estejam armazenados em outras *caches* neste determinado momento. Felizmente existe um artifício que pode separar os acessos à memória que certamente não irão mudar o estado de nenhum dado armazenado em outra *cache* daqueles que possam mudá-lo.Este artifício, descrito a seguir, possibilita que alguns acessos sejam realizados internamente ao NP, sem tornar incoerente a base de dados e sem a necessidade de interferência de *software*.

Para poder separar os acessos que certamente não irão alterar o estado das outras *caches* daqueles que podem, foi incluído um sinalizador de compartilhamento em cada linha de *cache* na memória principal local indicando se aquela linha já foi utilizada por alguma outra *cache* no mesmo *cluster*. Acessos não armazenáveis na *cache* não mudam o estado deste sinalizador.

Inicialmente, o estado do sinalizador é desligado, simbolizando que a linha não está presente em nenhuma outra *cache* do mesmo *cluster*. Leituras ou escritas no modo coerente[1], originadas por um controlador de *cache* remoto e do mesmo *cluster*, levam o estado do sinalizador da linha em questão para ligado. O estado do sinalizador volta a desligado quando o controlador de *cache* local endereçar esta linha com o comando de invalidação em conjunto ou não com leitura ou escrita.

Todos os endereçamentos dirigidos à memória local poderão ser executados sem a necessidade de se utilizar o barramento, com exceção de acessos no modo coerente e cuja linha, na memória local, esteja com o sinalizador de compartilhamento ligado.

A maior parte dos sistemas que utilizam redes de interconexão ([BBN85] [PFIS86]) não permitem que dados compartilhados sejam armazenados em *cache*. Esta restrição facilita a manutenção da coerência, uma vez que a memória principal é mantida sempre atualizada e não existem cópias dos dados em diversas *caches* espalhadas pelo sistema.

Procurou-se estender a possibilidade de armazenamento em *cache* ao maior número possível de tipos de variáveis. Foram necessárias, no entanto, algumas restrições para garantir a coerência da base de dados. Primeiro, ficou estabelecido que nenhuma *cache* poderia armazenar dados de leitura e escrita que pertencessem ao espaço de endereçamento da memória local de outro *cluster*. Segundo, dados que são acessados pela rede (compartilhados remotamente) mas que pertencam ao espaço de endereçamento local a um *cluster* podem ser armazenados em qualquer *cache* deste *cluster*.

---

[1]No modo coerente, todas as operações são necessariamente de blocos inteiros de *cache* para o esquema de *write back*. No esquema de *write through* esta restrição se refere apenas a leituras.

Fica a cargo do núcleo do sistema operacional programar corretamente as tabelas de gerência de memória, avisando ao controlador de *cache* quais as páginas que podem ser armazenadas em *cache* ou não, e de manter a exclusão mútua das variáveis migratórias.

Na arquitetura do MULTIPLUS, a coerência da *cache* de instrução é garantida pela não modificação de código durante a execução de um programa. Para atualização de páginas de programas com novos códigos, é necessário que o sistema operacional envie para o controlador de *cache* um comando de invalidação (*flush*) da *cache* antes de atualizar determinada página.

3 - Descrição do Funcionamento do Simulador

Para avaliação de diferentes alternativas de implementação de arquiteturas para o MULTIPLUS, e mais precisamente, para a arquitetura do sistema de memoria *cache*, foi desenvolvido um simulador capaz de analisar o desempenho do MULTIPLUS variando-se o número de barramentos externos (1 ou 2), a política de atualização da memória principal (sem *cache*, *write through*, e *copy back*) e considerando-se o uso ou não de *write buffer* para tentar igualar o sistema *write through* ao *copy back*, de desempenho reconhecidamente superior em sistemas monoprocessados com processadores CISC. O simulador foi escrito em PASCAL e roda em microcomputadores do tipo IBM-PC/XT, AT ou 386.

Para se entender o funcionamento do simulador, primeiramente é necessário que se conheçam algumas definições utilizadas neste texto:

- **Recursos:** são todos os objetos necessários para que um acesso de um processador se complete. Os elementos que sempre são alocados simultaneamente foram agrupados sob o mesmo nome, como por exemplo, o *snoop* e o barramento imediatamente conectado a este, sendo, então, o conjunto chamado apenas pelo nome de **SNOOP**.

- **Processador:** é o elemento com capacidade de processamento próprio do sistema. No caso do **MULTIPLUS**, a Unidade de processamento Inteira (IU).

■ **Estado:** é um atributo relativo aos processadores e recursos. Os estados são divididos em três grandes grupos, a saber: **FREE** (livre), **W** (relativo somente aos processadores, composto por diversos "sub-estados", simbolizando que o processador está parado, esperando pela alocação dos recursos necessários para que este realize o acesso correspondente) e **X** (também composto por diversos "sub-estados", tem relação com o estado **W** que o antecedeu e é relativo a processadores e recursos em geral que estejam correntemente executando um acesso). Os sub-estados dos estados **W** e **X** estão relacionadas ao tipo de acesso iniciado pelo processador.

3.1 - Alocação de Processadores

Inicialmente todos os processadores estão no estado disponível (**FREE**), simulando um estado inicial do tipo *reset*, quando a máquina é ligada. Cada processador será então alocado para executar um tipo de tarefa que consiste basicamente de um acesso de dado ou instrução. Embora, inicialmente, em um sistema real, a *cache* estivesse vazia e o primeiro acesso do processador fosse de busca de instrução, para fins de simulação, é considerado que a *cache* já está com taxa de acerto compatível com o seu estado de regime, eliminando-se assim problemas de estado transitório para que o número de iterações da simulação possa diminuir. Da mesma forma, o sinalizador de compartilhamento e o tipo de acesso a ser executado também iniciam a simulação no tempo como se já estivessem em regime. A alocação de tarefas (acessos) aos processadores é feita levando-se em conta apenas os percentuais dos tipos disponíveis de acessos conforme definição do usuário.

3.2 - Alocação de Recursos

Os diversos recursos disponíveis a cada processador são: o *cache*, o *snoop* e a memória local de seu NP; o barramento de dados, o barramento de instrução, as memórias locais e o *snoop* de NP's de seu *cluster*; e o barramento de dados, o barramento de instruções, a memória local e o *snoop* de NP's de outros *clusters*.

No caso de sistemas sem *cache*, não há necessidade de se alocar nenhum *cache* e *snoop*. Em simulações de sistemas com apenas um barramento, o

barramento de dados passa a ser também o de instruções, sendo, portanto, chamado de barramento único ou, somente, barramento.

Cada recurso possui dois atributos. O primeiro atributo indica o seu estado atual que pode ser **FREE** (livre) ou qualquer estado **X** (executando um acesso). O segundo atributo, que somente é válido quando o primeiro estiver no estado **X**, representa o tempo necessário para que o acesso corrente termine.

3.3 - Fluxo do Simulador

O procedimento executado pelo simulador pode ser dividido em três grandes partes que são: inicialização de variáveis, iteração de execução e contabilização. A iteração de execução, que constitui o núcleo do simulador, é composta basicamente de quatro rotinas:

- **Rotina de Alocação de Tarefas:** aloca para cada processador do sistema, que esteja no estado **FREE**, uma tarefa, ou seja, um tipo de acesso à memória, com distribuição aleatória, através de um gerador de números aleatórios variando de 0%, inclusive, a 100%, exclusive. Este número é comparado com o percentual estipulado pelo usuário no início da simulação, como por exemplo, taxa de acessos de busca de instrução, leitura de dados e escrita de dados:

```
if random < percentual de acessos para busca de código then
    rotina de escolha do tipo de acesso para busca de código
else if random < percentual de leitura de dado then
        rotina de escolha do tipo de acesso para leitura de dado
    else rotina de escolha do tipo de acesso para escrita de dado
```

Ao sair desta rotina, todos os processadores estarão ou no estado **W**, no caso de processadores recém-alocados, ou no estado **X**, já em execução.

- **Rotina de Alocação de Recursos:** uma vez que todos os processadores já possuem tarefas para executar, todos os que ainda estiverem no estado **W** procurarão alocar todos os recursos necessários para que possam iniciar o ciclo de acesso de acordo como seu tipo de estado. O processador somente muda de estado **W** para **X** quando conseguir alocar todos os recursos necessários.

correspondente ao estado W do processador que o alocou. O seu contador de tempo de acesso é carregado com o tempo total que este passara alocado durante este tipo de acesso.

■ **Rotina de Tempo:** este é o procedimento que emula a passagem do tempo para o sistema. Em todos os processadores e recursos que estejam em um estado W, executando um acesso, o contador de tempo é decrementado de uma unidade e comparado ao valor zero. Caso seja igual, o recurso ou processador é colocado no estado **FREE**.

■ **Rotina de Totalização Parcial:** em cada uma das rotinas descritas acima, contadores auxiliares são atualizados para que esta rotina possa contabilizar os diversos parâmetros de saída do simulador.

A rotina de totalização geral é chamada ao final da simulação totalizando os resultados obtidos pelo procedimento anterior, e, opcionalmente, salvando em disco os resultados para um posterior trabalho de comparação, geração de gráficos e tabelas, etc.

3.4 - Modelos de Política de "Cache" no Simulador

Três políticas de *cache* foram implementadas no simulador: sem *cache*, *write through* e *copy back*. O algoritmo utilizado para cada uma destas políticas é descrito a seguir.

■ Sem *cache*: Esta arquitetura foi simulada apenas para mostrar a influência da hierarquia de memória na redução do uso do barramento comum aos NP's. Nesta arquitetura só existem três tipos de acessos: à memória local do NP, ao *cluster* e a outro NP de outro *cluster*. Os acessos ainda podem ser de busca de instrução, leitura de dado ou escrita de dado.

■ *Write through*: Esquema de *cache* muito popular, foi implementado sem alocação de blocos em escritas (*write allocate*) como ocorre no controlador de *cache* Cypress CY7C605 [CYPR90]. Neste esquema, toda escrita realizada pelo processador passa automaticamente para a memória principal. Em caso de *hit* (acerto na *cache* privada local) na escrita, o dado da *cache* também é atualizado. Se em uma operação de leitura do processador houver

ausência na *cache* (*miss* ou *invalid*), o bloco inteiro que contém o dado faltoso é lido da memória principal e armazenado na *cache* como válido. Subsequentes leituras deste bloco se processarão apenas entre processador e *cache*.

Algumas restrições para manter a coerência foram feitas, assim como declarar como não possível de se armazenar em uma determinda *cache* de um NP 'variáveis (dados) que são acessadas por este pela rede de interconexão. A busca de instruções é sempre realizada trazendo-se para a *cache* um bloco inteiro, mesmo através da rede, porque a manutenção da coerência fica, neste caso, a cargo do Sistema Operacional.

Os acessos do processador à sua memória local se processam sem a utilização do barramento externo, com exceção de escrita de dados no modo coerente (de variável armazenada em *cache*), quando o sinal de compartilhamento deste dado na memória local estiver ligado. Neste caso, o processador é obrigado a realizar, em paralelo com a escrita na memória local, um ciclo de invalidação de bloco no barramento externo.

- *Copy Back*: Segue também a implementação da Cypress [CYPR90] com alocação de blocos em escrita (*write allocate*). A restrição de não armazenar em *cache* imposta na implementação anterior foi estendida também a todas as variáveis (dados) que podem ser acessadas por outro processador em outro *cluster*, mesmo no caso de residir na memória local de um determinado processador.

Quando um determinado processador **i** requisita um dado que não está presente na sua memória *cache*, este será lido da memória principal. Se houver outra cópia deste mesmo dado em outra *cache* **j** ($j \neq i$), o dado será armazenado como *shared*. Caso não exista nenhuma outra cópia, o dado será armazenado como *private*.

Em qualquer dos casos anteriores, um acesso do processador **i** a este mesmo dado já armazenado na *cache* **i** ocorrerá sem que o barramento externo seja utilizado, ou seja, um acesso a um dado presente na *cache* não gera acesso ao barramento, com exceção de uma escrita a um dado armazenado em *cache* no estado compartilhado (*shared*). Este acesso é necessário para que os outros controladores de *cache* saibam que o dado mudou de estado (de

shared para *modified*).

Operações de escrita em um bloco marcado como *private* provocam mudança no seu estado para *modified* enquanto que em blocos no estado *shared*, causam um acesso ao barramento para que os outros controladores de *cache* possam invalidar o bloco, como já foi dito anteriormente.

Quando um dado presente na *cache* i é acessado na memória principal por um outro processador j, o seu estado muda para *invalid* se o acesso for de escrita, ou para *shared* se for de leitura. Caso o estado inicial seja *modified*, o ciclo corrente é interrompido, o controlador i atualiza a memória principal e o processador j reinicializa o ciclo antes interrompido.

Os acessos do processador à sua memória local somente precisarão do barramento externo quando forem de dados, no modo coerente e o sinal de compartilhamento do bloco referenciado estiver ligado.

Quando um bloco é trazido da memória principal, é simulada a escrita do bloco antigo na memória principal (*write back*). A necessidade desta escrita tem probabilidade especificada pelo usuário e reflete o percentual de blocos alterados na *cache*.

4 - Parâmetros de Entrada

Nesta seção serão apresentados os parâmetros de entrada para o programa de simulação. Serão ainda fornecidos exemplos de parâmetros típicos para os sistemas simulados.

Para melhor exemplificação, os parâmetros foram divididos em dois grandes grupos: parâmetros de *hardware*, dependentes apenas do *hardware* do sistema a ser simulado, e parâmetros do sistema, dependentes do tipo de política de *cache* implementada, da hierarquia de memória e, principalmente, da carga de trabalho a ser simulada.

4.1 - PARÂMETROS DE "HARDWARE"

Os parâmetros de *hardware* são de três tipos: de temporização, de configuração e de arquitetura do sistema.

Com os parâmetros de arquitetura do sistema é possível configurar o número de barramentos (1 ou 2) que interligam os NP's, o número de NP's (1 à 16) por *cluster*, o número de *clusters* do sistema (1 à 256) e a profundidade do *buffer* de escrita (*write buffer*).

Os parâmetros de temporização são referentes aos tempos de acesso aos diversos niveis da hierarquia de memória. Para poder simular acessos em rajada (*burst*), os tempos de acesso à memória foram separados em tempo entre acessos em rajada e tempo para se acessar uma única palavra. A unidade de tempo é o periodo de relógio do processador do NP: 25ns.

- **Acesso à RI:** o atraso da rede de interconexão é fornecido por estágio e foi calculado em aproximadamente três períodos de relógio [BRON90].

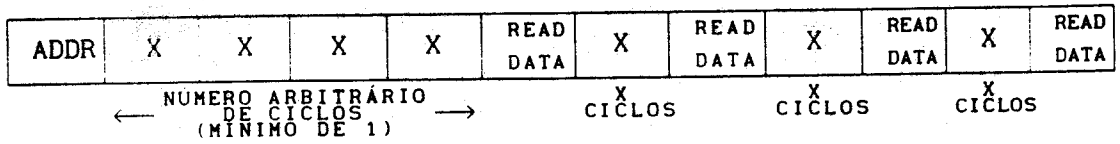


figura 3: Leitura no Barramento Externo

- **Acesso ao Barramento:** um acesso através do barramento que interliga os NP's pode ser dividido em três fases [CYPR90]: de aquisição do barramento, de endereço e de dado. A fase de aquisição do barramento tem tempo variável e impossível de ser fixado, mas é simulado pelo árbitro do Simulador. A fase de endereço (figura 3) possui tempo fixo e igual a uma unidade de tempo. A este tempo é somado o tempo gasto para que o acesso gerado pela *cache* seja decodificado dentro do NP, o que consome três ciclos de relógio, que são distribuídos entre a *cache* acionar as linhas de acesso depois de ter conseguido o barramento interno ao NP, a fase de endereço interna ao NP e a decodificação propriamente dita. A fase de dados é ditada pelo tempo de acesso do *slave* da operação e é proporcional ao número de palavras que é transferido.

acesso foi dividido em duas partes para refletir o acesso no modo rajada de memórias dinâmicas a serem utilizadas [MICR90]. Dependendo do tempo de acesso da memória, que atualmente pode ser de 80ns a 120ns, o tempo de acesso e o tempo entre acessos podem variar. Supondo-se uma memória com tempo de acesso igual à 100ns, o tempo entre acesso no modo rajada (*burst*) é de aproximadamente dois períodos de relógio. É necessário mais um período para ser calculado o código de correção (ECC), totalizando três ciclos de relógio. O tempo que antecede o primeiro acesso é igual a aproximadamente três ciclos de relógio.

- **Acesso à "Cache"**: o tempo de acesso à *cache* é de um ciclo de relógio para leitura e de dois ciclos para escrita. Não existe modo rajada para a memoria *cache*. Existe também um parâmetro que é o tempo necessário pelo controlador de *cache* para iniciar um acesso quando ocorre falha na *cache*.

4.2 - Parâmetros do Sistema

Os parâmetros do sistema são extremamente dependentes da carga de trabalho a que o sistema a ser simulado está sendo submetido, sendo, portanto, difíceis de serem estimados. Estes parâmetros são os seguintes:

- **Percentual de Acerto na "Cache"**: a carga de trabalho esperada no **MULTIPLUS** é basicamente de processamento científico, onde uma grande massa de dados é tratada simultaneamente em diversas iterações, o que favorece ao acerto na *cache*. É previsto que a taxa de acerto na *cache* de instruções seja de aproximadamente 95% [SMIT82] e a da *cache* de dados em torno de 85%

- **Percentual de Acerto no "Snoop"**: é o percentual de transações de memória no barramento imediatamente posterior ao controlador de *cache* que irão causar invalidação em uma determindada *cache*. Esta taxa foi estimada em 10% do total de acessos que podem causar invalidação. Este valor foi obtido a partir do percentual de variáveis compartilhadas. Em [WEBE89] este valor foi calculado em 1% do total de acessos. O valor de 10% foi escolhido em função do alto percentual de variáveis compartilhadas (ver item a seguir).

- **Percentual Compartilhado:** é o percentual de acesso à memória

local de um processador referenciando-se a uma variável compartilhada. Em [WEBE89] foi calculado um percentual de variáveis compartilhadas de 2% enquanto que em [FEIT90] e [RUDO85], um percentual de 5%. Ambos os sistemas analisados nestes trabalhos possuem memória compartilhada mas não são tão fortemente acoplados como no caso do Sistema **MULTIPLUS**. Para simular a influência do Sistema Operacional distribuido, que possui muitas variáveis compartilhadas e poder avaliar uma carga de trabalho não muito ideal para o sistema, o valor utilizado foi de 20%, tanto para código como para dados.

- **Percentual de Acesso Local:** é uma percentagem do total de acessos gerados pelo processador referindo-se à memória local do próprio NP. Foi estimado um valor de 80% [AZEV91].

- **Percentual de Acesso no "Cluster":** possui a mesma formação do percentual de acesso local. É o percentual de acessos gerados pelo NP nos barramentos externos que necessitam da rede de interconexão. Foi utilizado 80% para dados e instruções. No caso de instruções, este percentual reflete a política a ser adotada pelo Sistema Operacional de transferir todas as páginas de código para a memória do *cluster* que a esteja utilizando.

- **Percentual de Leitura, Escrita de Dados e Busca de Código:** é a maior diferença entre simulações de sistemas baseados em processadores RISC e CISC. Enquanto que em processadores CISC os percentuais mais comuns para leitura, escrita de dados e busca de código são 35%, 15% e 50% respectivamente [SMIT82], os percentuais referentes a processadores RISC são aproximadamente 13%, 7% e 80% [NAMJ88] [TAMI81].

- **Percentual Escrito:** refere-se ao percentual de dados armazenados na *cache* de dados que necessitam de atualização na memória principal (*copy back* ou *write back*) quando forem substituídos por outro bloco na *cache* ou por um pedido de atualização de outro *cache*. Seu valor foi estimado em 10%.

5 - Resultados Obtidos

Nas experiências de simulação, o cálculo da convergência dos resultados mostrou ser de custo muito alto, sendo mais razoável a pré-fixação do tempo total de simulação em ciclos de relógio.

O custo de simulação, em relação ao tempo real, é de 11ms por NP, por ciclo de relógio, quando o simulador é executado em um computador compatível com IBM-PC com processador Intel 8088 com relógio de 4.77MHz.

Sem considerar o resultado obtido por apenas um ciclo de simulação, o desvio padrão de uma amostra de simulação, para até 1.000 ciclos de relógio, está próximo de 30%, caindo rapidamente para apenas 5% com 10.000 ciclos de relógio. Com 100.000 ciclos, o erro máximo é de 2%, o que não representa um ganho muito significativo em relação ao aumento do tempo total de simulação, principalmente, lembrando que este é o pior caso. Portanto, o tempo de simulação foi fixado em 10.000 ciclos de relógio.

A seguir serão apresentados os resultados obtidos pelas simulações.

■ **Taxa de Ocupação dos Barramentos:** Pode-se observar pelas figuras 4 e 5 que o aumento do número de processadores devido ao aumento do número de *clusters*, com relação NP/*cluster* constante, pouco altera as taxas de ocupação dos barramentos, o que é uma das grandes vantagens características das redes de interconexão.

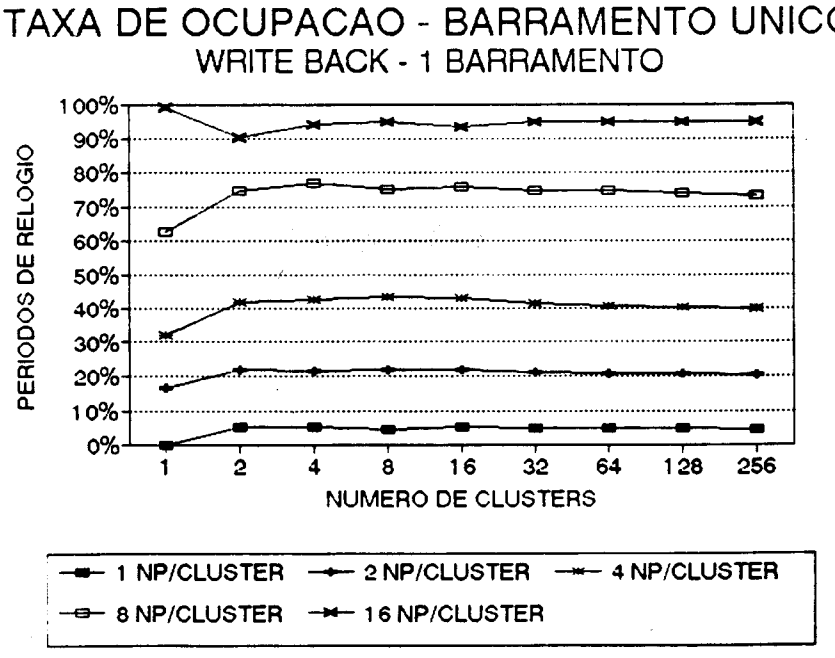


figura 4: Taxa de ocupação do barramento único

Para qualquer sistema simulado, o uso do barramento externo é nulo

para a configuração de 1 NP/*cluster* e 1 *cluster*, uma vez que não existe nada conectado ao barramento, com exceção da interface de E/S que não foi simulada. A taxa de ocupação para o barramento de código é idêntica nos sistemas com *cache* utilizando-se *write-back* ou *write-through*.

Pode-se observar, comparando os gráficos de taxa de ocupação de barramento de dados (figura 5) que o método de *write-back* reduz o tráfego no barramento em sistemas com pouco compartilhamento, ou seja, com poucos NP's para, depois, igualar-se ao *write-through* com o aumento do número de NP's.

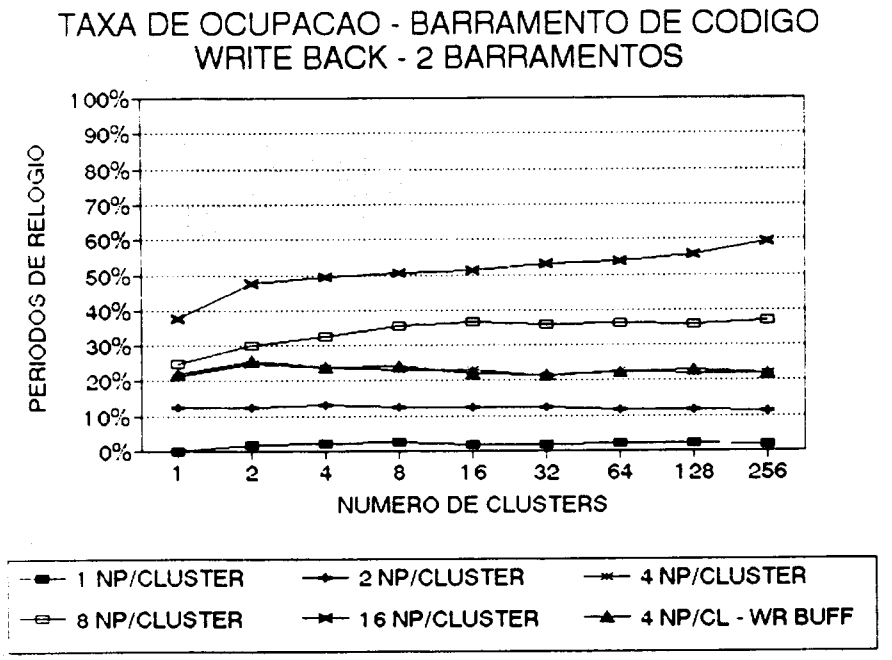


figura 5.a: Taxa de ocupação do barramento código.

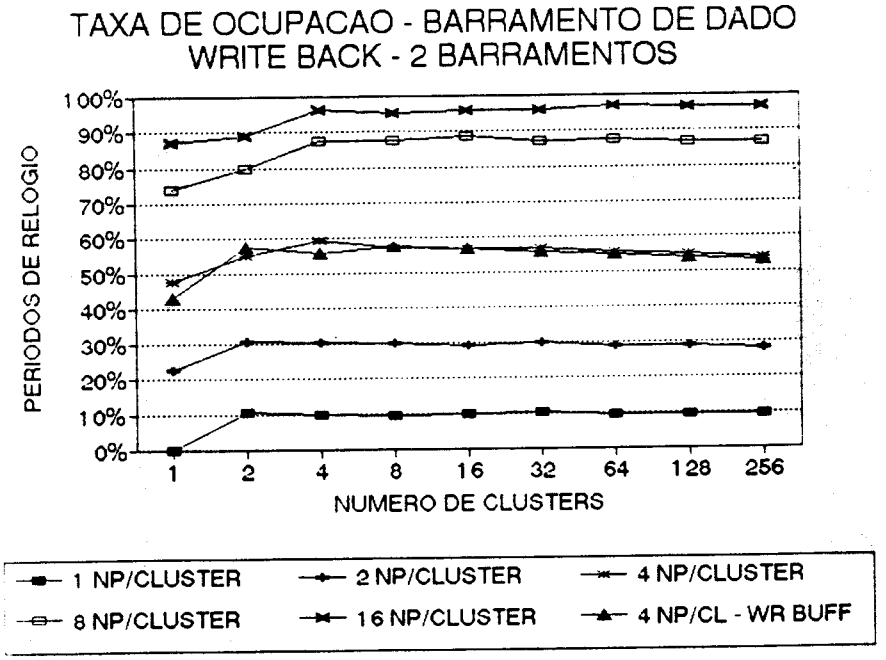


figura 5.b: Taxa de ocupação do barramento dado.

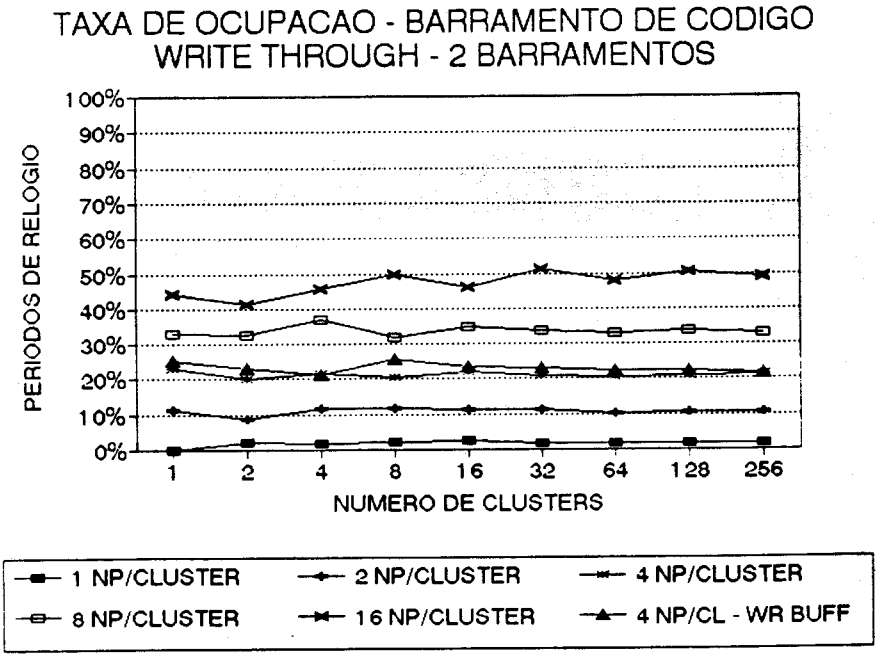


figura 5.c: Taxa de ocupação do barramento código.

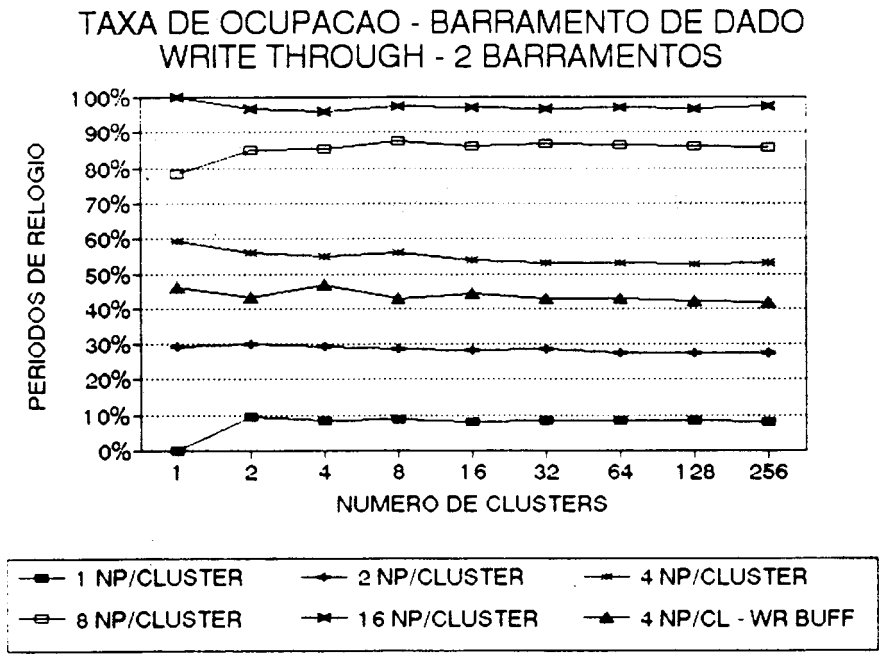


figura 5.d: Taxa de ocupação do barramento de dado.

■ **Duração Média dos Ciclos:** Através do número total de ciclos executados pode-se notar que para até 16 NP's por *cluster*, o desempenho do sistema acompanha o número de NP's. Principalmente, observa-se que existe pouca degradação com o aumento de *clusters*, o que se traduz graficamente pelas linhas quase horizontais nos gráficos. Em compensação, a duração média dos ciclos aumenta muito com o aumento do número de NP's por *cluster*.

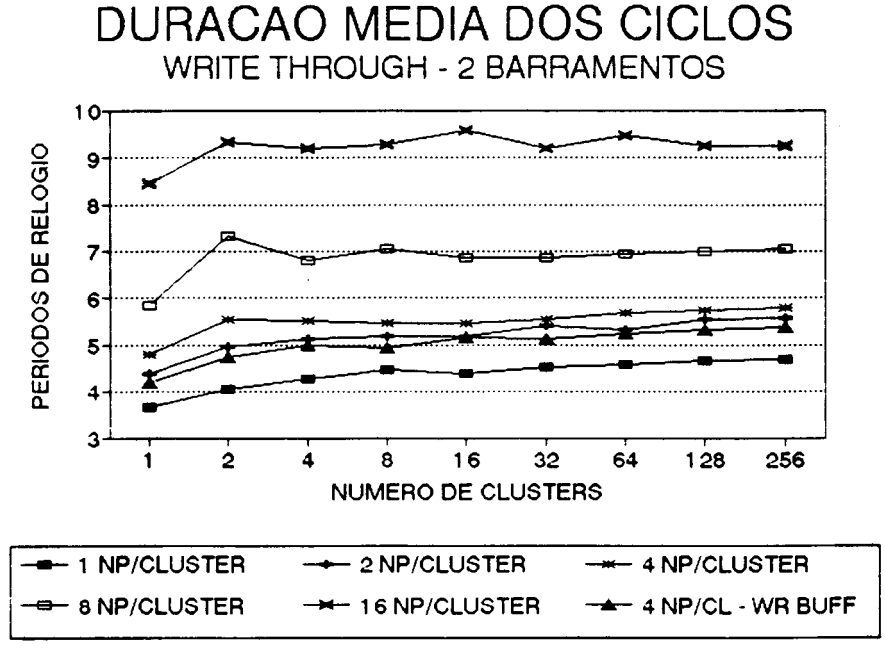


figura 6.a: Duração média dos ciclos em *write through*.

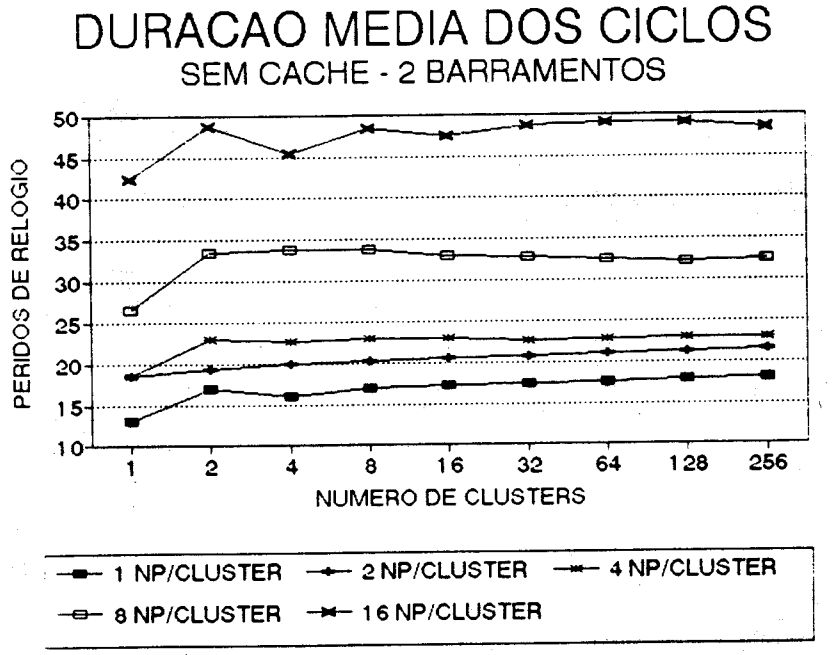


figura 6.b: Duração média dos ciclos sem *cache*.

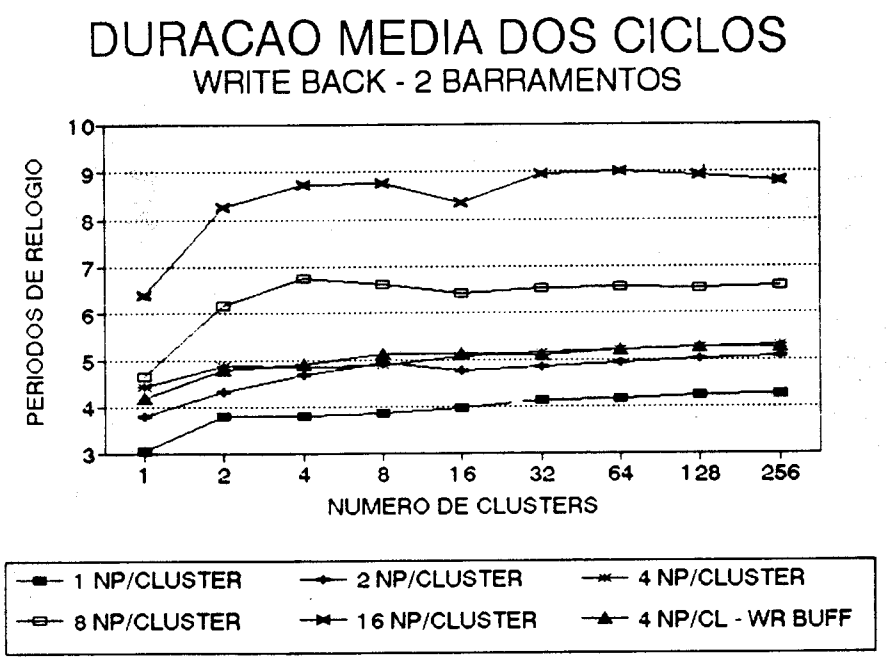


figura 6.c: Duração média dos ciclos em *write back* com 2 barramentos.

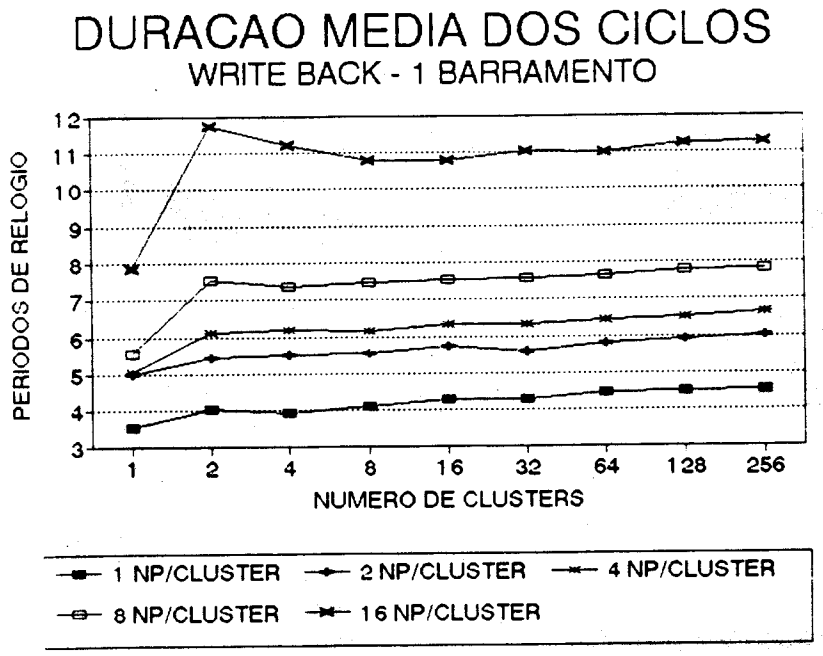


figura 6: Duração média dos ciclos em *write back* com 1 barramento.

Através da análise dos gráficos de duração média dos ciclos (figura 6), pode-se constatar a grande superioridade, em termos de desempenho, dos sistemas com *cache* sobre os sem *cache*. O método *write-back* apresenta um desempenho 20% superior ao método *write through* quando poucos NP's são utilizados. No entanto, esta taxa cai para cerca de 4% quando o número de NP's cresce muito

A inclusão de *write buffer* nos sistemas que utilizam *write through* melhorou significativamente (de 20% a 25%) o desempenho do sistema. Já nos sistemas que utilizam a política de *write back* não houve modificação observável. O principal fator que contribuiu para a pouca melhora está no fato de que a maior parte das escritas realizadas são de blocos inteiros da *cache* (32 *bytes* ou 4 acessos) e o *write buffer* implementado somente possui capacidade para armazenar um acesso por vez.

A figura 7 mostram a superioridade do sistema de dois barramentos sobre o sistema de um barramento quando o número de NP's por *cluster* é maior ou igual a 8.

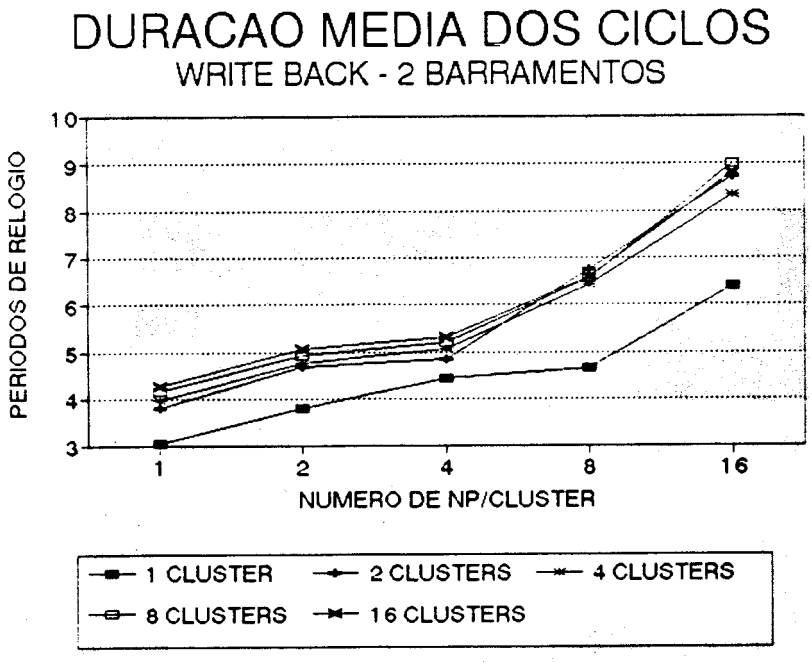


figura 7.a: Duração média dos ciclos em *write back* com 2 barramentos.

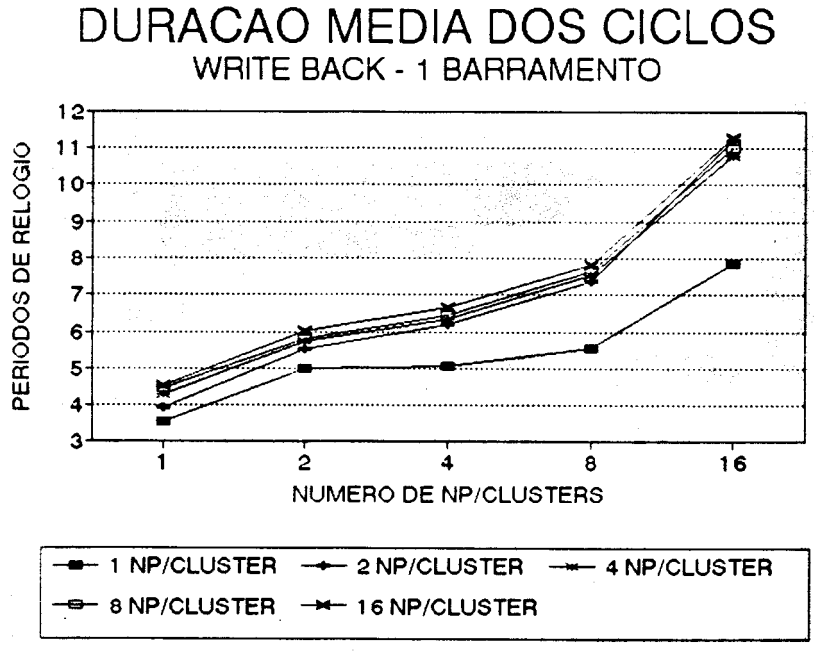


figura 7.b: Duração média dos ciclos em *write back* com 1 barramentos.

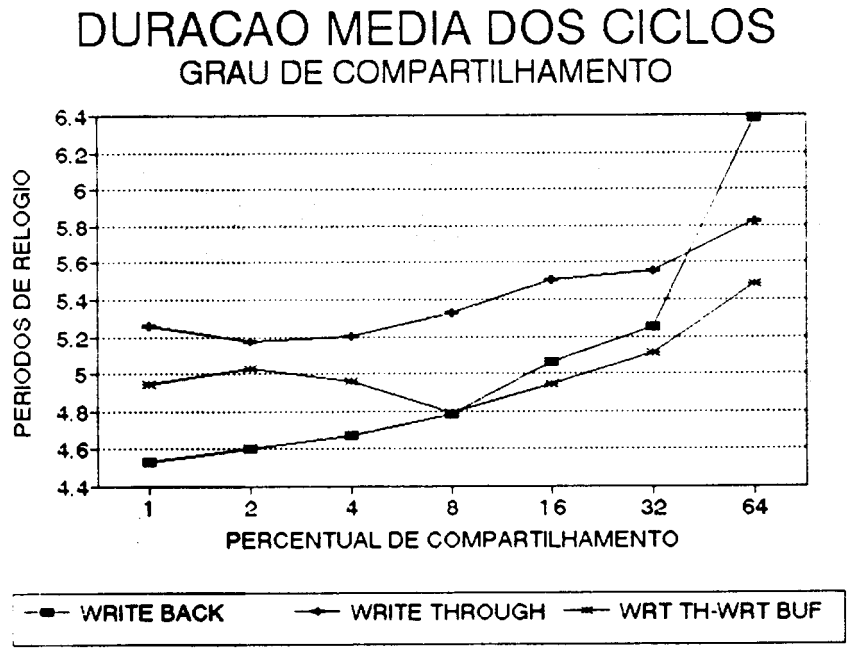


figura 8: Duração média dos ciclos X compartilhamento.

A figura 8 representa a simulação de uma configuração do MULTIPLUS com 4 *clusters* e 4 NP's por *cluster*. Observa-se que para valores pequenos para percentual de compartilhamento de dados, o esquema de *write back* é bastante superior ao esquema de *write through*. No entanto, quando este percentual atinge a faixa de 10%, o esquema de *write through* com *write buffer* passa a apresentar um desempenho superior. Constata-se ainda que o comportamento do sistema com política de *write through* é basicamente invariante com o percentual de compartilhamento. Já o desempenho do sistema utilizando *write back* cai rapidamente com o percentual de compartilhamento.

6 - Conclusões e Perspectivas Futuras

A política de atualização da memória principal do tipo *write back* mostrou que, em termos de desempenho do sistema, é superior à de *write through*. Entretanto, pôde-se notar que com o aumento do grau de compartilhamento de variáveis, esta diferença no desempenho diminui, em parte devido ao grande número de invalidações de *cache* seguidas de *copy back* para a memória principal.

Com a inclusão de um *buffer* de escrita (*write buffer*), possibilitando que escritas na memória principal sejam realizadas, em alguns casos, em apenas um ciclo de relógio, o sistema utizando a política do tipo

write through apresentou desempenho equivalente ao sistema baseado na política de *write back*.

Como continuação deste trabalho, pretende-se incluir as rajadas de E/S no estudo visando obter-se resultados de simulações mais próximos aos de uso normal do sistema. Em uma próxima versão, a contenção na rede de interconexão terá sua simulação analítica substituída por uma simulação funcional para melhorar a estimativa dos resultados.

7 - AGRADECIMENTOS

Os autores agradecem ao CNPq e à FINEP o apoio dado ao desenvolvimento deste projeto.

8 - REFERÊNCIAS

[AUDE90] J. S. AUDE et al, "MULTIPLUS: Um Multiprocessador de Alto Desempenho", Anais do 10º Congresso da SBC, Vitória, pp 93-105, jul/1990

[AZEV91] G. P. AZEVEDO et al, "MULPLIX: Um Sistema Operacional Tipo Unix para o Multiprocessador MULTIPLUS", Relatório Técnico RT 91-1 - Núcleo de Computação Eletrônica - Universidade Federal do Rio de Janeiro, jan/91

[BBN85] "Butterfly Parallel Processor Overview", BBN Laboratories Incorporated, versão 1, 17pp, 13/jun/85

[BRON90] G. BRONSTEIN et al, "Análise do Desempenho de Redes de Interconexão para Máquinas Paralelas", Anais do III Simpósio Brasileiro de Arquitetura de Computadores/Processamento Paralelo, pp 345-360, Rio de Janeiro - RJ, 7-9/nov/90

[CYPR90] "SPARC RISC USER'S GUIDE", Cypress Semiconductor Corporation, segunda edição, fev/1990

[FEIT90] R. Q. FEITOSA, "O Problema de Coerência de Memórias Cache Privadas

em Grandes Multiprocessadores para Aplicações Numéricas: Uma Nova Solução", Anais do 10º Congresso da SBC - Vitória, ES, pp 138-156, 22-27/jul/90

[GOOT83] A. GOTTLIEB e outros, "The NYU Ultracomputer - Designing an MIMD Shared Memory Parallel Computer", IEEE Transactions on Computers, vol. C-32(2), pp 175-189, fev/1983

[MICR90] "MOS DATA BOOK", Micron Technology Inc, 1990

[NAMJ88] M. NAMJOO et all, "CMOS Custom Implementation of the SPARC Architecture", 33th IEEE Computer Society International Conference, pp 18-20, California, 29/fev-04/mar/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, "Dinamic Decentralized Schemes for MIMD Parallel Processors", Proceeding 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, set/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 17(2), pp 243-256, abr/89