Estudo de Arquiteturas de Memórias Cache Distribuídas para o Sistema MULTIPLUS

Alexandre Malheiros Meslin

TESE SUBMETIDA AO CORPO DOCENTE DA COORDENAÇÃO DOS PROGRAMAS DE PÓS-GRADUAÇÃO DE ENGENHARIA DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE MESTRE EM CIÊNCIAS EM ENGENHARIA DE SISTEMAS E COMPUTAÇÃO.

Aprovada por:

Prof. Ageu Cavalcanti Pacheco Junior, PhD
PRESIDENTE

Prof. Edil Severiano Tavares Fernandes, PhD

Prof. Júlio Salek Aude, PhD

Prof Adriano Joaquim de Oliveira Cruz, PhD

RIO DE JANEIRO, RJ - BRASIL

AGOSTO DE 1991

MESLIN, ALEXANDRE MALHEIROS

Estudo de Arquiteturas de Memórias *Cache* para o Sistema MULTIPLUS [Rio de Janeiro] 1991.

ix, 100 pp, 29,7 cm (COPPE/UFRJ, M.Sc., Engenharia de Sistemas e Computação, 1991)

Tese - Universidade Federal do Rio de Janeiro, COPPE.

1 - Memória *Cache* e Arquitetura de Computadores

I. COPPE/UFRJ II. Título (série)

Para minha mãe Sônia, minha avó Ruth e minha esposa Denise

AGRADECIMENTOS

Agradeço ao Professor Ageu Pacheco Cavalvanti Junior o apoio e orientação recebidos.

Ao professor Júlio Salek Aude pelas valiosas sugestões em toda a tese, e, principalmente, durante as simulações.

Ao NCE/UFRJ pela utilização dos seus laboratórios.

Ao CNPQ e FINEP pelo apoio financeiro ao Projeto MULTIPLUS que originou esta tese.

Resumo da Tese apresentada à COPPE/UFRJ como parte dos requisitos necessários para a obtenção do grau de Mestre em Ciências (M.Sc.)

ESTUDO DE ARQUITETURAS DE MEMÓRIAS *CACHE* DISTRIBUIDAS PARA O SISTEMA MULTIPLUS

Alexandre Malheiros Meslin

Abril, 1991

Orientador: Ageu Cavalcante Pacheco Júnior
Programa: Engenharia de Sistemas e Computação

RESUMO

Este trabalho apresenta as considerações iniciais, os problemas e as limitações de um projeto de memórias *cache* para um sistema do tipo multiprocessador conectado através de uma rede de interconexão, onde as terminações são barramentos paralelos com diversas *CPU*'s. São também apresentados um pequeno histórico de memórias *cache*, a análise de esquemas de manutenção de coerência e ainda algumas arquiteturas de *cache* possíveis de serem implementadas. A seguir, são discutidos esquemas de atualização da memória principal e de manutenção da coerência aplicáveis ao Sistema MULTIPLUS. Por fim, as diversas arquiteturas propostas são simuladas para uma efetiva comparação dos seus desempenhos.

Abstract of Thesis presented to COPPE/UFRJ as partial fulfillment of the requirements for the degree of Master of Science (M.Sc.)

STUDY OF ARCHITECTURES OF DISTRIBUTED CACHE MEMORIES FOR THE MULTIPLUS SYSTEM

Alexandre Malheiros Meslin

April, 1991

Chairman: Ageu Cavalcante Pacheco Júnior
Department: Engenharia de Sistemas e Computação

ABSTRACT

This work presents some initial considerations, problems and limitations of a multicache memory project for a multiprocessor system coupled by an interconnection network and parallel buses. A brief review of cache memories, an analysis of data coherence schemes and some possible cache architectures are also presented. Main memory update and coherence maintenance schemes applied to the MULTIPLUS system are discussed. Finally, some architectures are simulated for performance comparisons.

ÍNDICE

CAPÍTULO I

INTRODUÇÃO

Uma memória *cache* é uma unidade de armazenamento geralmente pequena e muito rápida situada entre o processador e a memória principal que contém cópias de certas posições específicas desta. Cada entrada na *cache* contém um dado e o seu endereço (ou parte dele) associado, além de um campo de estado deste dado armazenado (*tag*). O campo de endereço e de estado é chamado de rótulo (*tag*). Cada acesso à memória é mapeado em uma entrada da *cache*. A comparação do endereço armazenado na *cache* com o do acesso corrente determinará se existe um acerto (*hit*) ou falha (miss). Todas as memórias *caches* necessitam de um *bit* por bloco sinalizando se o dado presente é válido ou não. Geralmente, invalidações na memória *cache* ocorrem na troca de processos ou após a carga inicial do sistema. O endereço que chega à memória *cache* pode ser tanto o endereço virtual quanto o real, dependendo da forma de implementação.

Visando aumentar a capacidade de processamento dos computadores (ainda monoprocessados), observou-se que, mesmo com a evolução da tecnologia, as unidades de processamento (*PE - processing element*) necessitavam informações a uma taxa (ciclo de *CPU*) que crescia mais rapidamente do que a das memórias convencionais em fornecer estas informações (ciclo de memória), o que deixava a unidade de processamento muito tempo parada (*idle*). A memória do sistema ganhou, então,a conotação de gargalo de "von Newmann" [STON87]. Observou-se também que a maioria dos programas passava grande parte do seu tempo de execução em trechos de tamanhos reduzidos (quando comparados ao tamanho do programa principal)[1].

A primeira idéia de memória *cache* foi a da colocação de uma memória, entre a *CPU* e a memória principal, de menor capacidade de

---

[1]Princípio da localidade de referência

armazenamento, porém bem mais rápida (5 a 10 vezes) do que a principal. Esta memória forneceria as informações (código e dados) à unidade de processamento a uma taxa que minimizaria o seu tempo *idle.*

A introdução de outras unidades que poderiam acessar a memória principal e o compartilhamento do barramento fez com que o "gargalo", que antes consistia apenas do tempo de acesso à memória principal, passasse a incluir também a disputa pelo barramento entre a unidade de processamento e os controladores de entrada e saída. Uma nova justificativa para as memórias *caches* (que tenderiam a desaparecer, uma vez que o tempo de acesso à memória diminuia com a evolução da tecnologia) foi a redução da taxa de acessos ao barramento.

Com o aparecimento dos computadores multiprocessados, o uso de *caches* tornou-se indispensável. Sua importância maior não reside mais apenas no fato de tentar manter a unidade de processamento com taxa de ocupação total, e sim em possibilitar que, com uma taxa de acerto (*hit rate*) a mais próxima possível de 100%, um maior número de *PE*'s possam compartilhar um mesmo barramento. Modernamente, mesmo uma memória *cache* com tempo de acesso comparável ao da memória principal seria justificável, pois funcionaria como uma espécie de *buffer*, reduzindo o tráfego entre os processadores e a memória principal [KAPL73].

Para melhor situar o problema, seja por exemplo, um sistema multiprocessado, onde os processadores demandam um acesso à memória a cada ciclo, mas não possuem memória *cache*. Supondo um barramento cujo esquema de prioridade de atendimento de acessos seja fixo (pelo número da placa, por exemplo), este sistema comportará idealmente apenas um *PE*, no máximo dois[2], já que um terceiro jamais ganharia o barramento conforme mostrado na figura 1.1.

---

[2]sendo que a cada instante de tempo haverá sempre um *PE* parado (*idle*), não contribuindo para a melhoria do desempenho do sistema

BR*n*- pedido de acesso do processador *n*

BG - reconhecimento ao pedido de acesso de qualquer PE

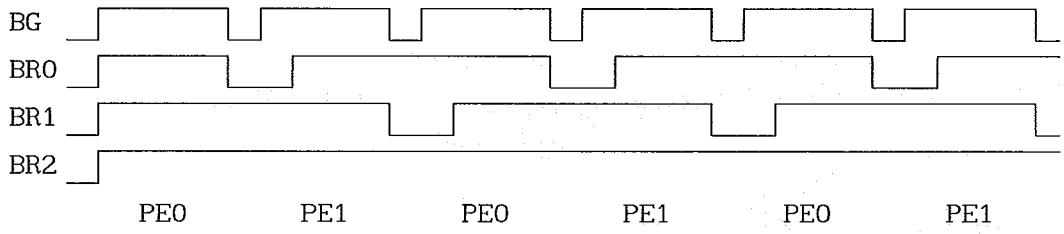


figura 1.1: diagrama de tempos

Este mesmo sistema, com *cache*, com *hit rate* de 95% (que atualmente pode ser considerada baixa) e uma relação entre tempo de acesso à *cache* e à memória principal de um fator de 10, comportaria perfeitamente 4 *PE*'s. O número máximo de *PE*'s em um mesmo barramento pode ser aproximadamente calculado por[3]:

$$N_{PE} \cong \frac{T_{CACHE} * P_{HIT}}{(1 - P_{HIT})\ T_{BUS}} + 1$$

onde: N_{PE} = número de processadores

P_{HIT} = taxa de acerto (*hit rate*)

T_{BUS} = tempo total de acesso ao barramento

T_{CACHE} = tempo total de acesso à *cache*

Para uma taxa de acerto igual para todas as *caches* dos *PE*'s, de P_{HIT} = 98% e um barramento com 100% de ocupação, um sistema com barramento é viável com até oito *PE*'s.

---

[3]desenvolvimento no apêndice D.

I.1 - Motivação

Este trabalho analisa diversas opções de arquitetura de memórias caches possíveis para o Sistema MULTIPLUS [AUDE90] e também suas implicações quanto ao tipo de protocolo de barramento e outros aspectos de arquitetura e sistema operacional. A escolha da arquitetura de cache pode influenciar de forma significativa o desempenho geral do sistema, principalmente considerando o esquema proposto de barramento duplo, como será visto na seção 3.

Sistemas de memória *cache* já foram discutidos em diversos trabalhos, sendo, portanto, a sua influência no desempenho de sistemas de computadores monoprocessados bem conhecidas.

A necessidade de uma simulação para o Sistema MULTIPLUS decorre de diversas inovações propostas para a arquitetura desta máquina, em particular, a co-existência de barramento e rede de interconexão ligando os diversos processadores e a sua arquitetura de memória com uma forte e bem definida hierarquia, onde cada processador possui uma memória local pertencente ao espaço de endereçamento global, além de permitir que variáveis compartilhadas sejam armazenadas em mais de uma *cache* privada.

Outro fator preponderante é a raridade de simulações de sistemas baseados em processadores de arquitetura RISC.

I.2 - Descrição da Arquitetura do MULTIPLUS

A necessidade de processar informações cada vez mais rápidamente, leva os projetistas de computadores a buscar novas soluções de arquitetura para os sistemas. O desenvolvimento da microeletrônica, que tem propiciado uma maior integração dos circuitos e, principalmente, o avanço da tecnologia CMOS, que possibilita a construção de transistores mais rápidos, tem fornecido componentes a baixo custo com velocidades superiores.

Observa-se que a demanda de processamento cresce a uma razão maior

do que a de aumento de velocidade dos computadores, assim como também é esperado um limite físico para o aumento da velocidade dos circuitos. Novas soluções foram investigadas de forma a tentar solucionar o problema. Primeiro houve a introdução de outros processadores para tarefas mais simples como as de E/S, depois vieram os computadores multiprocessados, onde vários processadores compartilham os mesmos recursos. Neste caso, o meio originalmente utilizado de acesso aos recursos era o próprio barramento paralelo já existente. Entretanto, o aumento do número de processadores acarreta um correspondente aumento do tráfego no barramento, não permitindo que este número seja muito grande. Assim, os projetistas procuraram outras formas de interconectar os processadores, as memórias e os periféricos. Soluções foram encontradas na especialização de barramentos e na inclusão de redes das mais variadas topologias.

Em continuidade ao desenvolvimento dos componentes, duas correntes em relação à filosofia de arquitetura do processador central se estabeleceram:

CISC (*Complex Instruction Set Computer*): esta corrente aposta em processadores com instruções mais complexas e por este motivo, microprogramados, o que leva a ciclos de instruções mais lentos, mas que ao mesmo tempo necessitam de poucas dessas instruções para executarem as tarefas.

RISC (*Reduced Instruction Set Computer*): a outra corrente acredita que os computadores possam ser baseados em processadores com instruções muito simples que podem ser executadas em apenas um ciclo de máquina e não necessitam ser microprogramados. Por outro lado, precisam de muito mais instruções do que os processadores CISC para realizarem as mesmas tarefas, aumentando muito a importância do projeto de memória *cache* para reduzir o tráfego no barramento e diminuir o tempo de espera dos processadores.

O MULTIPLUS é um computador multiprocessado de alto desempenho com arquitetura modular que está sendo desenvolvido no Núcleo de Computação Eletrônica da Universidade Federal do Rio de Janeiro. Capaz de suportar até 2048 nós de processamento, o MULTIPLUS é baseado em microprocessadores RISC

de 32 *bits* de arquitetura SPARC.

Em uma mesma placa (figura 1.2), denominada Nó de Processamento (NP) estão o processador, o co-processador de ponto flutuante, 64 K*bytes* de memória *cache* para dado e 64 K*bytes* de memória *cache* para instruções, até 32 M*bytes* de memória do tipo RAM e interface para barramentos externos caracterizados a seguir.

Até oito NP's podem ser interligados por dois barramentos de 64 *bits* de largura para dado com 36 *bits* de endereço formando um *cluster*. Os barramentos são especializados em transferência de instruções e transferência de dados, possuindo uma vazão máxima (de pico) de 320 M*bytes* por segundo [CYPR90b].

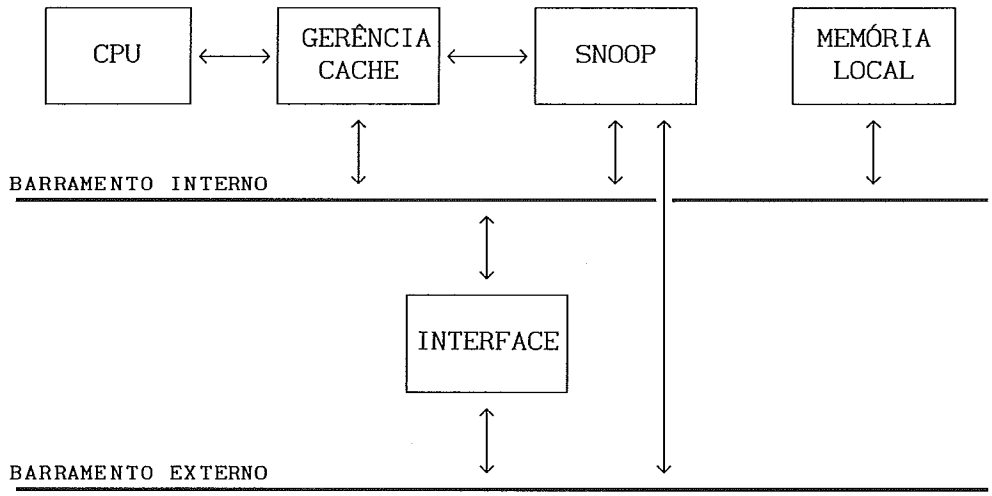


figura 1.2: diagrama em blocos do NP do **MULTIPLUS**

Os barramentos conectam os NP's às interfaces da rede de interconexão multiestágio do tipo N-cubo invertido (RI) [CRUZ90]. A RI pode suportar até 256 *clusters*.

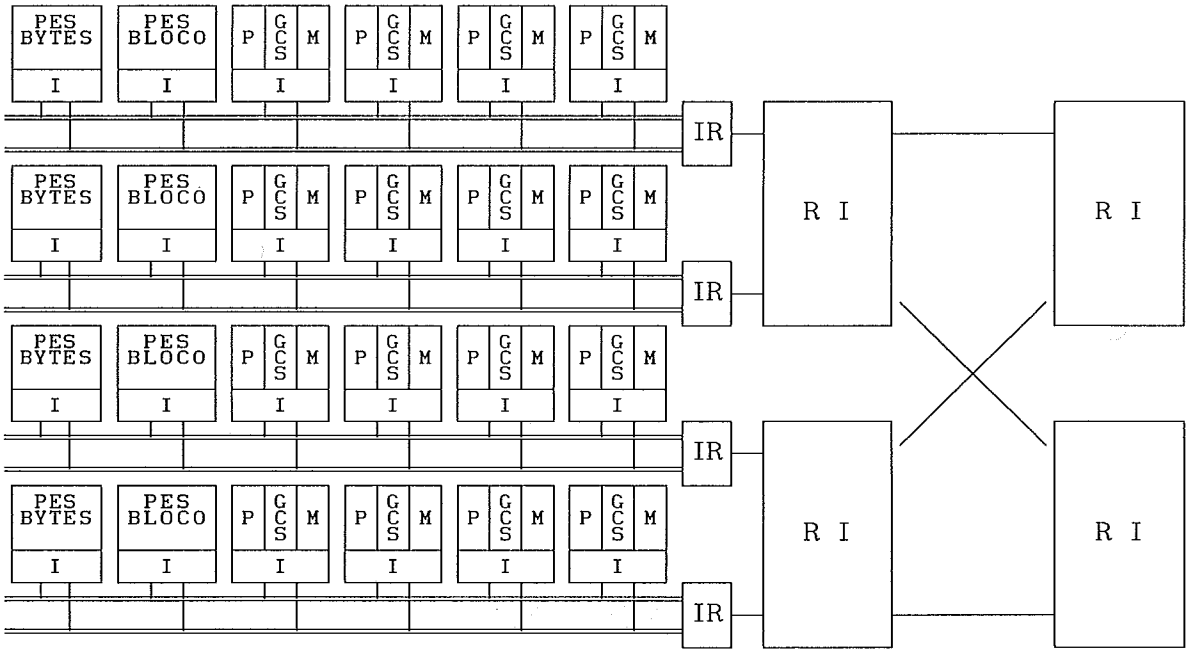
A memória local de cada NP pertence ao espaço de endereçamento global do sistema, podendo ser acessada por qualquer outro NP. Para o *software*, a memória do sistema é composta por um único grande bloco de até 32 G*bytes*.

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 1.3 apresenta a arquitetura total do Sistema **MULTIPLUS** com 4 NP's por *cluster* e 4 *clusters*.

O esquema de manutenção de coerência de *cache* no MULTIPLUS é discutido na seção 4. Nas seções 2 e 3 serão apresentados alguns conceitos básicos de *cache* utilizados neste trabalho.



Legenda:
- P - Processador
- C - Cache
- G - Gerência
- S - Snoop
- M - Memória
- I - Interface
- RI - Rede de Interconexão
- IR - Interface de Rede
- ═ - Barramento Externo

figura 1.3: diagrama em blocos do Sistema **MULTIPLUS**.

CAPÍTULO II

REVISÃO DE MEMÓRIA "CACHE"

Dada a importância assumida pela utilização de memórias *cache* em sistemas multiprocessados, é lícito hoje afirmar que o desempenho de tais sistemas está intimamente ligado ao bom projeto do sistema de memória *cache*. Idealmente, o projeto deve considerar o tipo de aplicação à qual a máquina se destina para que possa ser identificada a sua *workload* (carga de trabalho) típica. Por outro lado, não é solução uma *cache* de tamanho muito grande que eleva demais o custo do projeto além de aumentar a sua complexidade ao ponto de torná-lo comercialmente inviável.

Variações mínimas da taxa de acerto da *cache* podem provocar uma diminuição muito grande no desempenho do sistema. Seja por exemplo, um sistema cujo tempo de acesso à memória principal é 10 vezes maior do que o de acesso à *cache*. Considerando-se apenas ciclos de leitura, o tempo médio de ciclo de *CPU* pode ser expresso pela fórmula:

$$t_{eff} = h * t_{cache} + (1 - h) * t_{main}$$

onde: t_{eff} = tempo eficaz médio de ciclo de *CPU*,
h = taxa de acerto na *cache*,
t_{cache} = tempo médio de acesso à *cache*,
t_{main} = tempo médio de acesso à memória principal.

Assim, para uma variação da taxa de acerto de 99% para 98% (variação relativa menor do que 1%), o tempo médio de ciclo aumenta em quase 10%.

Existem dois tipos de configurações básicas de memória *cache* para sistemas multiprocessados: a *cache* compartilhada (*shared cache*) e a *cache* privada (*private cache*) [DUBO82]. A *cache* compartilhada pode ser acessada por mais de um processador (fig. 2.1.a). É geralmente um *buffer* de alta

velocidade de uma determinada parte da memória. A memória *cache* privada é dedicada a um único processador e armazena blocos de diversas unidades de memória (fig. 2.1.b).

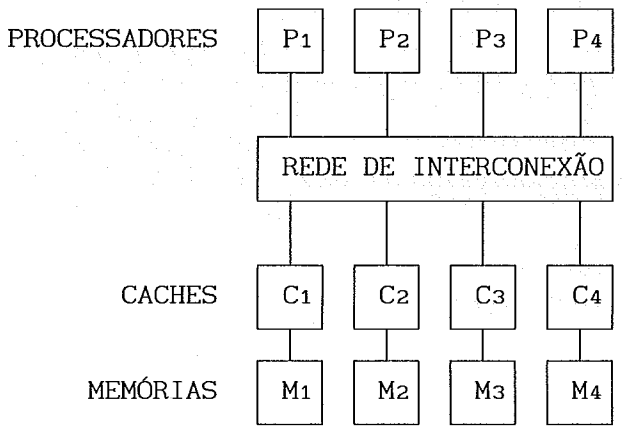


figura 2.1.a: sistema com *cache* compartilhada

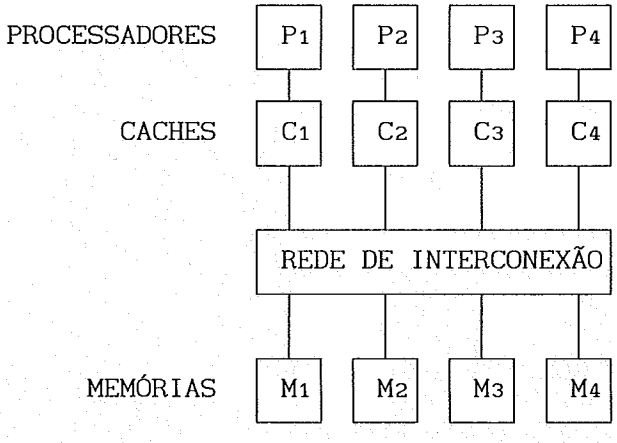


figura 2.1.b: sistema com *cache* privada

A desvantagem da *cache* compartilhada em relação à privada é que a primeira não reduz o tráfego nas vias de comunicação entre os processadores e os módulos de memória [KAPL73]. A *cache* privada é a opção mais utilizada em sistemas computacionais, sendo que o sistema Alliant FX/8 [TEST86] utiliza *cache* compartilhada.

II.1 - Alternativas de Arquiteturas "Cache"

Neste item serão apresentadas algumas características e parâmetros de arquiteturas de controladores e memórias *cache*.

Dependendo do esquema de gerência de memória[1] utilizado, são possíveis duas alternativas de arquitetura de *cache* em relação ao endereço: *cache* de endereço virtual e *cache* de endereço físico.

No *cache* virtual, o endereço que o controlador de *cache* recebe é o endereço virtual gerado pelo processador (figura 2.2). Neste caso, não existe o atraso correspondente ao cálculo de conversão do endereço virtual em real pela gerência de memória, possibilitando ao controlador de *cache* respostas mais rápidas. Por outro lado, a cada troca de contexto, quando as tabelas de gerência de memória são alteradas, há a necessidade de invalidar a *cache* toda, uma vez que o novo processo poderá gerar endereços virtuais que estejam presentes na *cache* mas não correspondam ao mesmo endereço físico.

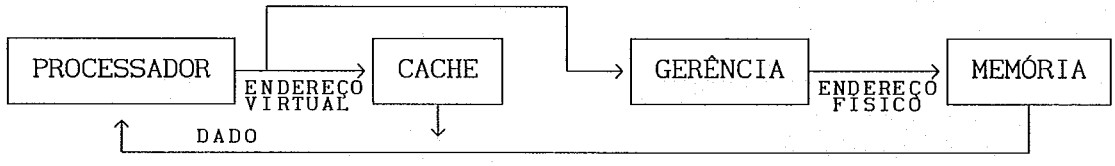


figura 2.2: diagrama em blocos de um sistema com *cache* de endereco virtual.

Alguns controladores de *cache* virtual (figura 2.3) armazenam também o endereço físico juntamente com os dados para evitar a necessidade de invalidar a *cache* a cada troca de processo. Para este tipo de controlador, é interessante que a gerência processe o endereço virtual em paralelo com a busca na *cache* e forneça o endereço físico em tempo menor do que o ciclo de *cache* para comparação com o endereço armazenado.

---

[1] entende-se como gerência de memória, neste caso, como sendo a unidade que, de alguma forma, remapeia os endereços gerados pelo processador, independentemente do método utilizado.

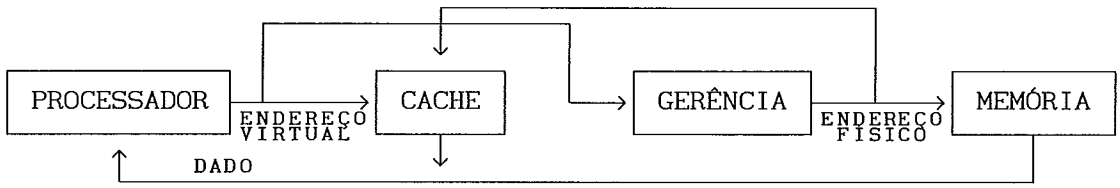


figura 2.3: diagrama em blocos de um sistema *cache* de endereço virtual e armazenamento de endereço físico

Cache Físico: a memória *cache* de endereço físico ou simplesmente *cache* físico, recebe o endereço já processado pela gerência de memória (figura 2.4).

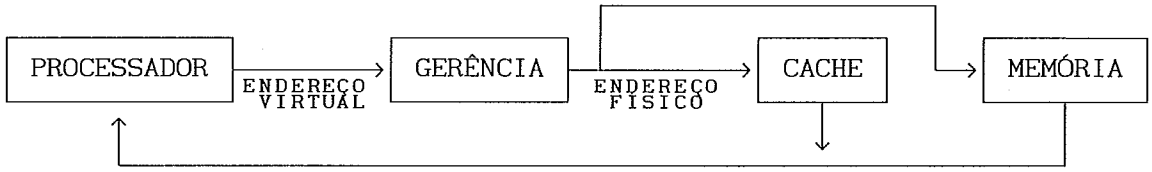


figura 2.4: diagrama em blocos de um sistema de *cache* de endereço físico

Para minimizar os efeitos da gerência entre a *cache* e o processador, geralmente aproveita-se o fato da gerência de memória não utilizar os *bits* de menor ordem da palavra de endereço e utiliza-se estes para endereçar a *cache*. Em paralelo, a gerência de memória converte o endereço virtual em real em um tempo compatível com o de acesso à *tag* de endereço da *cache*, ficando ambos os endereços disponíveis, quase que simultaneamente, para comparação pelo controlador de *cache* [HENN86]. O principal problema deste esquema está na limitação do número de entradas na *cache* pelo tamanho da página de memória virtual porque somente os *bits* não mapeados podem ser utilizados para endereçar a *cache* (figura 2.5).

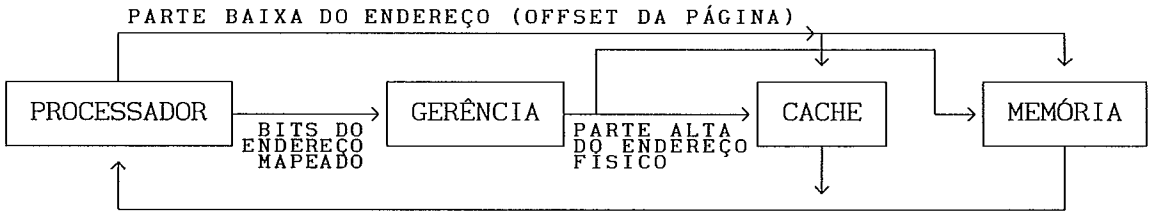


figura 2.5: diagrama em blocos de um sistema de *cache* de endereço físico e busca na *cache* em paralelo

A figura 2.6 fornece um exemplo de como a operação é realizada. Supondo um processador de 32 *bits* de endereço, páginas de 64K *bytes* e uma

cache com 2K entradas.

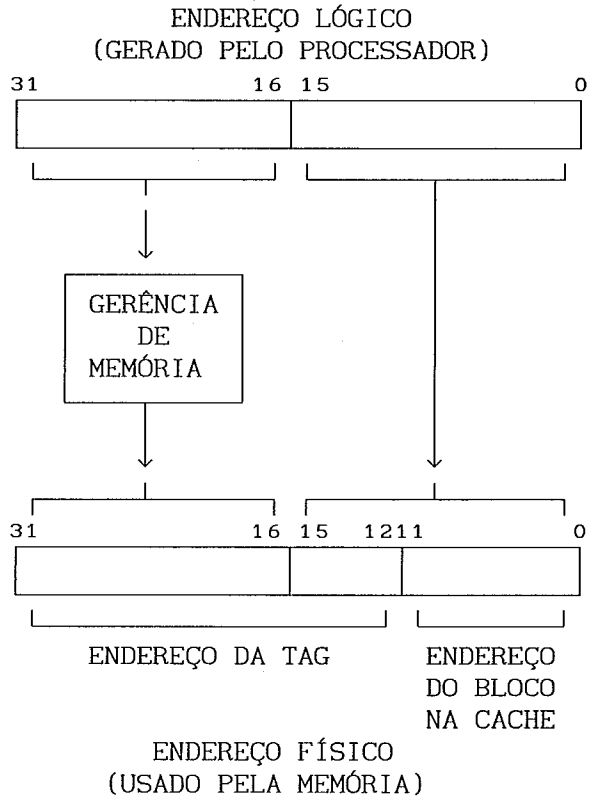


figura 2.6: formação de endereço físico e acesso à *cache* em paralelo

Uma memória *cache* pode ser vista como uma matriz de três dimensões que são as palavras, os blocos, e os conjuntos associativos (figura 2.7). Uma *cache* é formada por **s** conjunto associativos, cada um contendo **b** blocos de **p** palavras

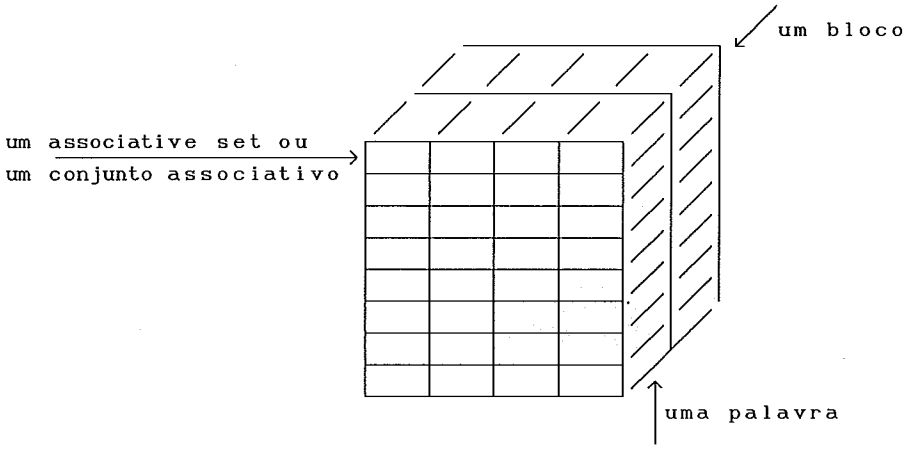


figura 2.7: representação de uma memória *cache* com:
- 64 palavras,
- bloco de 2 palavras,
- 8 conjuntos associativos de 4 blocos cada.

Um conjunto associativo (*associative set*) consiste de blocos que possuem em comum a mesma parte baixa do endereço do dado armazenado e podem mapear os mesmos blocos congruentes da memória principal. Quando é necessário ler um novo bloco da memória principal para a *cache*, um algoritmo de reposição deve selecionar qual bloco, dentro do conjunto, que deverá ser escolhido para ceder sua posição. Em geral, são algoritmos do tipo LRU, FIFO ou de base aleatória.

No exemplo da figura 2.7, os endereços dos conjuntos associativos seriam os três *bits* menos significativos da palavra de endereço, descontando-se os *bits* utilizados para selecionar o elemento acessado dentro do bloco[2], ou seja, os $\log_2 n_s$, onde n_s é o número de conjuntos associativos. O gráfico da figura 2.8 ilustra a relação entre o grau de associatividade e a taxa de falha na *cache*. Os blocos congruentes a um mesmo conjunto diferem entre si pelo rótulo associado (parte alta do endereço).

---

[2]Para um processador de 32 *bits* (4 *bytes*) e um bloco de *cache* de 256 *bits* (32 *bytes*), é necessário descontar 5 *bits* para seleção do elemento endereçado dentro do bloco.

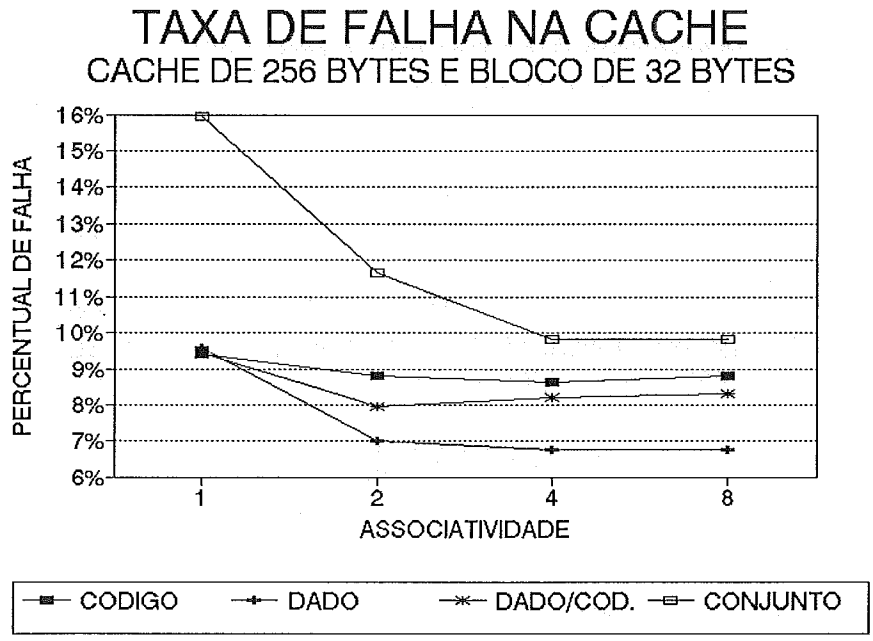


figura 2.8: grau de associatividade X taxa de falha na *cache* [3]

Definição de Bloco: é a unidade de informação nas operações de leitura/escrita entre o controlador de *cache* e a memória principal. Geralmente possui número de *bits* múltiplo da largura do barramento de dados, e normalmente é composto por "potência de 2" palavras do processador. Alguns autores preferem dar ao bloco o nome de linha, devido à forma de organização dos dados na memória *cache*.

O tamanho do bloco é um importante parâmetro para o desempenho da *cache*, visto que estudos sobre taxas de acerto de programas monoprocessados têm mostrado que, para uma *cache* de tamanho fixo, a taxa de falhas inicialmente diminui com o aumento do tamanho do bloco [EGGE89] porque os programas tendem a referenciar instruções e dados na vizinhança dos recentemente referenciados [KAPL73]. A medida que o tamanho do bloco cresce e se aproxima do tamanho da *cache*, a taxa de falhas começa crescer. Tal

---

[3] os gráficos 2.8 e 2.9 foram obtidos através de um pequeno simulador do tipo *trace driven* executando um programa que simula uma aplicação tipicamente científica.

fenômeno é explicado pelo fato de que a medida que a distância de um certo objeto pertinente ao bloco aumenta com relação ao objeto originariamente referenciado, sua chance de vir a ser referenciado diminui (*memory pollution* [SMIT87]). O gráfico 2.9 ilustra a relação entre o tamanho do bloco da *cache* e a sua taxa de falha.

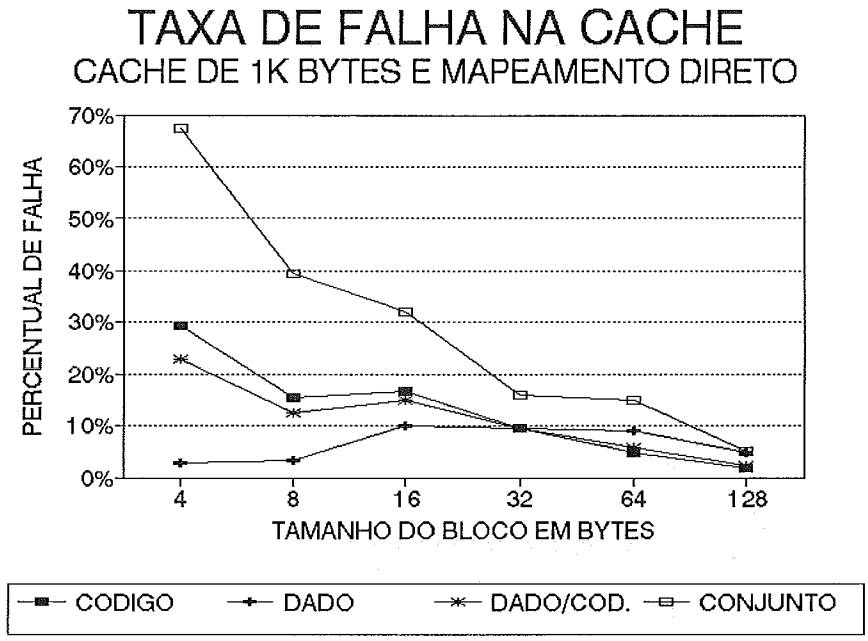


figura 2.9: tamanho do bloco X taxa de falha

Os comportamentos da demanda de dados de instruções de um programa não são idênticos, o que leva a uma necessidade de políticas diferentes para a gerência de *cache* [SMIT83].

Enquanto que a demanda de instruções de um programa é essencialmente sequêncial, com repetições destas sequências através de *loops*, dados são, em sua maioria, acessados por uma complexa lei de formação ou de forma puramente sequêncial quando se trata de vetores.

Sendo o código de um programa de apenas leitura, as invalidações na *cache* somente ocorrerão ao término de um programa, o que torna a coerência por *software* fácil de ser implementada. Em sistemas em que a *cache* seja acessada através de endereços virtuais, há necessidade de invalidação também quando houver troca de contexto, favorecendo o controle por *software*

da coerência dos dados armazenados em *cache*.

Neste mesmo esquema estão os dados apenas de leitura (*R/O*) ou utilizados por somente um processador. A partir desta separação, problema de manutenção de coerência se resume aos dados compartilhados por mais de um processador e que sejam de leitura e/ou escrita.

Weber [WEBE89] e Agarwal [AGAR88] dividem os objetos em diversos conjuntos dependendo do tipo de invalidação por eles provocados:

1 - código e dados de apenas leitura,
2 - objetos migratórios,
3 - objetos de sincronização,
4 - objetos muito lidos[4],
5 - objetos frequentemente lidos e escritos[5]

Uma outra divisão de dados que melhor se adapta às necessidades do sistema em discussão (figura 2.10) seria:

1 - código,
2 - dados de sincronização,
3 - dados locais,
4 - dados remotos no mesmo barramento,
5 - dados compartilhados através da rede de apenas leitura,
6 - dados compartilhados através da rede de leitura e escrita.

---

[4]objetos muito lidos são objetos que são lidos muitas vezes antes de serem escritos.

[5]objetos frequentemente lidos e escritos são objetos que são frequentemente lidos e frequentemente escritos simultaneamente.

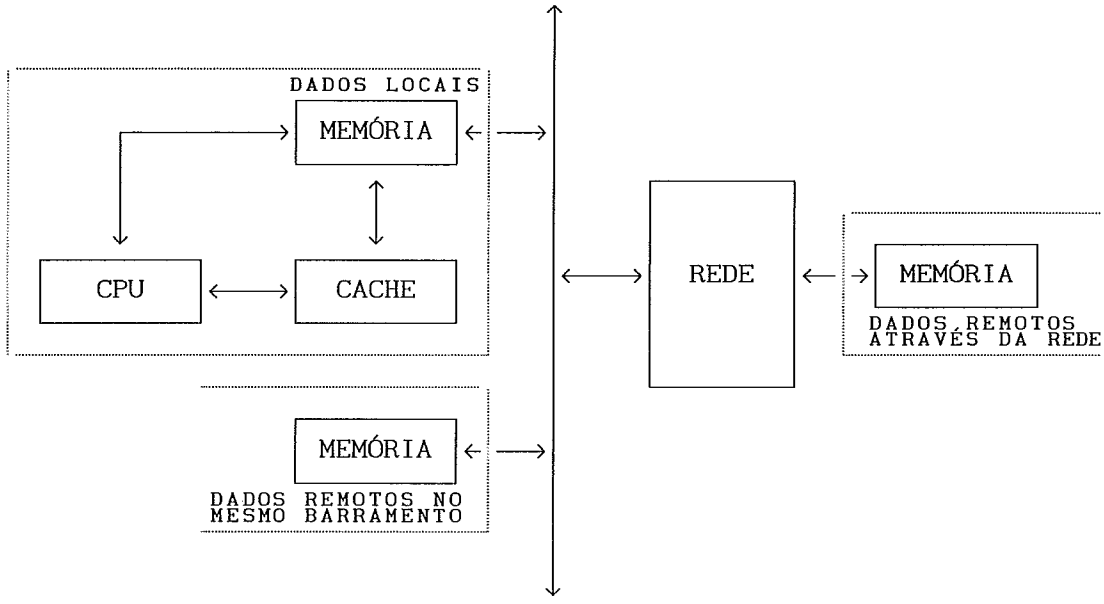


figura 2.10: distribuição dos dados em relação ao processador.

II.2 - Métodos de Atualização da Memória Principal

Esquemas de coerência de memória *cache* para sistemas multiprocessados interligados por um barramento comum a todas as unidades que podem acessar a memória principal já foram analisados por diversos autores ao longo do tempo [WILK65] [LEE69] [GOOD83]. A característica básica de um esquema de coerência consiste na monitoração do barramento (*snoop*) ou na troca de informações entre cada unidade que possua *cache* para detectar mudanças de conteúdo de dados que estejam mapeados em sua *cache* de forma a permitir que esta possa ser atualizada ou invalidada. Durante ciclos de leitura na memória principal, a monitoração é importante somente em alguns esquemas de *cache* que não atualizam a memória principal a cada ciclo de escrita da *CPU*. Neste caso, o valor contido em alguma *cache* será o atualizado, necessitando ser passado para a unidade que o requisitou.

Em sistemas *multicache* conectados por redes que não interliguem todas as unidades, não há um esquema de *snoop* implementável que não degrade consideravelmente o desempenho do sistema para monitorar a ação de unidades distantes (que não estão conectadas diretamente).

A seguir serão descritos alguns esquemas de atualização da memória principal comumente utilizados.

II.2.1 - "Write Trough"

O método *write through* destaca-se pela sua simplicidade e reduzido número de estados (dois - *Valid* e *Invalid*). Outra grande vantagem está no fato de que praticamente qualquer tipo de protocolo de barramento pode suportar esta política de controle de *cache*.

Neste esquema, toda escrita realizada pelo processador passa automaticamente para a memória principal. Em caso de *hit* (na *cache* privada local) na escrita, o dado da *cache* também é atualizado. Se em uma operação de leitura da *CPU* 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*. Variações deste esquema, que assumem que localidades de referência também são geradas por escritas do processador, podem alocar um bloco na *cache* também em ciclos de escrita realizando um ciclo de leitura de bloco seguido da escrita da palavra. O bloco alocado nestas condições poderá desalocar algum outro bloco alocado por demanda de leitura.

A figura 2.11 ilustra o diagrama de estados de um bloco genérico na memória *cache* para esquemas do tipo *write-through*.

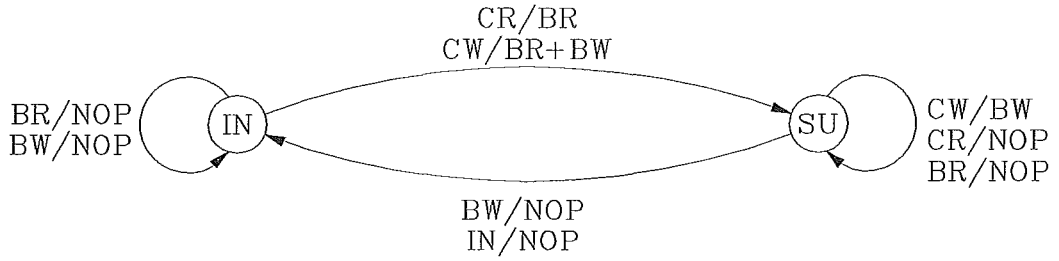


figura 2.11: esquema *write-through* - diagrama de estados de um bloco na *cache*

Ação/Reação
- BR - leitura de/no barramento
- BW - escrita de/no barramento
- CR - leitura executada pela CPU
- CW - escrita executada pela CPU
- IN - invalidação de entrada na *cache*
- NOP - nenhuma operação realizada ou necessária

Estado (dentro do círculo)
- IN - inválido (*INvalid*)
- SU - válido (*Shared Unmodified*)

Este método é o mais popular, sendo utilizado em sistemas como o Pégasus 32X do NCE-UFRJ. O método também é disponível em controladores de *cache* integrados fabricados comercialmente. Como exemplos, pode-se citar o Cypress CY7C604 e CY7C605 [CYPR89a], o Motorola 88200 [MOTO88], ambos para suas respectivas linhas de processadores RISC, e o Austek Microsystems A38152 [AUST87] para sistemas baseados em microprocessadores 80386 [INTE87] da Intel.