ANÁLISE E ALTERNATIVAS PARA CONTENÇÃO EM MULTIPROCESSAMENTO

M. C. A. Zabeu
CPqD - TELEBRÁS
Caixa Postal 1179, 13085 - Campinas - SP

RESUMO

O presente artigo visa avaliar as alternativas propostas para o problema da contenção existente em sistemas multiprocessados. A contenção se dá quando do acesso simultâneo à memória compartilhada; sobremaneira existe o fenômeno de "hot spots", áreas de memória com alta taxa de acesso, o que leva a uma degradação da performance do sistema como um todo, independente de sua arquitetura. Em especial é mostrada uma alternativa para implementação de ferramentas baseadas em memória compartilhada sem incorrer em "hot spots", viabilizada por recursos de hardware da placa U32, desenvolvida no CPqD. Propõem-se também modelos de utilização destas ferramentas tanto a nível de software básico quanto de aplicação, baseando-se na máquina em questão.

ABSTRACT

Memory contention is a problem in multiprocessing architetures, that leads to "hot spots" (highly accessing memory locations) which causes overall performance degradation. This paper analyses and presents some alternatives to this problem, focusing the U32 characteristics. We discuss both the hardware features and software models adaptable for this topic, in the P3 environment.

1. Contenção/hot spots

1. 1. Introdução

Com a grande ênfase dada hoje em arquiteturas voltadas para processamento paralelo, surgem assuntos de interesse específico no tocante ao controle de recursos em um ambiente composto de várias unidades computacionais. Independentemente do tipo de arquitetura utilizada, a possibilidade de utilizar memória partilhada é atrativa para funções de sincronização ou passagem de dados entre os elementos processadores. Classicamente são definidos dois tipos de sistemas no tocante ao compartilhamento de memória: sistemas fortemente acoplados, onde todos os processadores têm igual acesso à memória, sem necessariamente caminho para inter-comunicação exceto esta memória, e sistemas fracamente acoplados, onde geralmente não existe possibilidade de acesso de um elemento processador à memória de outro [1]. Hoje existem filosofias mistas onde se tem tanto uma quanto outra forma de trabalho.

Algumas das vantagens na existência de memória compartilhada mais ressaltadas são, do ponto de vista de programação, a não necessidade de partição de dados entre processadores (trazendo então eficiência e simplicidade no acesso aos dados) e não tão evidente, a facilidade de programação e transporte de ambientes uniprocessadores para multiprocessadores [1]. Do ponto de vista de implementação de ferramentas "básicas", a memória compartilhada é utilizada para mecanismos de sincronização, onde os diferentes processos espalhados no ambiente de multiprocessamento devem consultar valores determinados para estabelecer condições para seu funcionamento.

1. 2. Características da contenção

O acesso à memória compartilhada pode se dar através de diversos mecanismos, como através de um barramento comum, uma matriz de comutação, ou via mecanismos de controle via hardware [1] [4] [5]. Em todos os casos, entretanto, é mostrado que existem posições de memória "hot", cuja característica é o seu acesso mais frequente comparado às demais posições. Resultados de simulações mostram que, quando existe este efeito (doravante chamado de "hot spot"), tem-se:

- uma degradação do tráfego de dados do sistema como um todo (não somente no acesso aos "hot spots"), e
- é um efeito geral, que independe do tipo de arquitetura utilizada [3].

As variáveis do tipo "hot spot" são, na grande maioria, utilizadas para sincronização entre processos. É demonstrado que, apesar do acesso a estas locações ser pequeno frente a todo o acesso à memória, tem-se o problema de contenção e, como citado acima, uma degradação do sistema [2].

1. 3. Arquiteturas clássicas e o problema da contenção

A arquitetura mais simples para multiprocessamento é a que utiliza um barramento comum, graças ao seu baixo custo e como uma evolução normal de arquitetura de minicomputadores. A contenção é bastante visível neste tipo de arquitetura, uma vez que o próprio barramento é um recurso compartilhado que vai ser disputado a cada elemento processador acrescentado a ele.
Extensão para o barramento comum é o uso de uma matriz de conexão. O uso de vários barramentos unidos por uma matriz suaviza o problema da contenção, pois, apesar de utilizar as mesmas técnicas de arbitragem dos esquemas a barramento comum, os conflitos tendem a ser exceção e não regra. Entretanto, a complexidade destas arquiteturas é bastante maior, e, embora a adição de novas unidades processadoras a um sistema não leve a maior contenção (pois também contribuem com seu próprio caminho entre processador/memória), levam a um aumento de custo da ordem quadrática [1].
Outra alternativa para arquiteturas multiprocessamento que evitam a contenção é o uso de redes multi estágio no formato de árvore, redes omega, cubos binários ou variações destas. Algumas técnicas, como a de combinação dos acessos a "hot spots", foram propostas; entretanto o custo do hardware associado é proibitivo [3]. Nestes esquemas, a idéia é incorporar um hardware específico nas redes de conexão entre processadores para, quando do acesso à memória, combiná-los em estágios de forma que apenas um acesso seja efetuado e então propagado para os diversos solicitantes [2]. É também possível implementação deste esquema de combinação em software. Neste caso, o controle de combinação dos acessos, que deveria ser armazenado em "buffers" hardware associados a cada elemento de comutação, fica em um módulo de memória partilhada. Além da diminuição de custo, este esquema contribui também para um ajuste mais fino do sistema, uma vez que a quantidade possível de acessos a um nó da árvore (refletido no tamanho do "buffer" associado à "chave comutadora") é modificado facilmente.

2. A placa U32 e a contenção

2. 1. Características da placa

A placa U32, desenvolvida no CPqD, é baseada em um microprocessador 80386, tendo como processador associado (basicamente para fins de comunicação) um T800 (transputer), além de interfaces seriais RS232, contadores, e demais hardware de suporte. Conta com uma memória "cache" de 128K bytes, e contém memória interna até 16M bytes, com detecção de erros duplos e correção de erros simples.[10]
A placa U32 é conectada a outras placas U32 e às demais placas da família PP (placas periféricas) através de um barramento proprietário de 32 bits de largura (referido doravante como barramento global).
Características importantes da placa U32 são o acesso memória interna das demais U32 e a posibilidade de escrever contemporaneamente nas memórias de todas as U32 ligadas no barramento global (broadcast) [10]. Estes acessos são feitos utilizando faixas especiais de endereços. A faixa de endereço utilizada para o broadcast (0FCxxxxxxH) é reconhecida por todas as placas U32.
Diversas arquiteturas podem ser implementadas com a placa U32. Algumas propostas, em um estudo feito para sua predescessora, a placa UPN, podem ser encontradas em [6].

2. 2. Mecânismos de acesso à memória na U32

Na placa U32 existem de 1 a 16 Mbytes de memória interna à placa. Esta variação é possível através da troca de integrados nos dois bancos de memória da placa. A memória interna de uma placa é visível pelas demais existentes no mesmo barramento. Esta característica exclui a necessidade de uma memória adicional para uso como área de armazenamento compartilhado. O acesso à esta memória pelos elementos da

placa U32 é feito através de um barramento interno. Quando o acesso se dá por outra placa, utiliza-se primeiramente o barramento global. O acesso a dados da memória interna nem sempre utiliza este barramento. Isto acontece quando o dado já se encontra armazenado na memória "cache" do processador que solicita o acesso. Existe uma memória "cache" de 64/128 Kbytes associada a cada processador da placa U32 (386 e T800). Ela contém somente os dados referentes a acessos à memória interna a placa. O uso da "cache" justifica-se por dois fatores:

- adequação da velocidade de acesso dos processadores à da memória;
- minimizar o uso do barramento interno à placa para acesso à memória de forma a reduzir a contenção no uso do mesmo;

Note-se que, sem o uso de "caches", apenas o processador iAPX386 utilizaria toda a memória constantemente, inviabilizando a operação do outro processador. O esquema de memória "cache" utilizado é do tipo "write-through". A consistência dos dados é feita através de um mecanismo em hardware em que, se é feita alguma escrita em uma locação cujo conteúdo já está na "cache", invalida-se o conteúdo da mesma [5], [10].
A memória "cache" traz ainda, do ponto de vista da contenção, uma outra vantagem: uma vez lido um determinado dado, enquanto não for alterado ele pode continuar a ser lido sem que se utilize o barramento interno à placa.
Isto libera o acesso externo àquela posição, não havendo portanto contenção.
O acesso à memória de uma placa por outra é efetuado usando o barramento global e o barramento interno às placas. Os acessos externos devem ser usados preferencialmente para comunicação de dados. O código será sempre executado na memória da placa de maneira a minimizar os acessos externos que existiriam (no caso contrário) devido aos ciclos de busca de código [7]. Os acessos externos não têm "cache" associada, já que o aumento de complexidade de hardware não justificaria a coerência para manutenção de dados que devem ser usados apenas em situações isoladas, como na transferência de um bloco de dados.
Outra modalidade de acesso à memória na U32 é a chamada "broadcast". Existe para este caso uma faixa de endereçamento de até 16 Mbytes que nos ciclos de escrita é reconhecida por todas as placas em um barramento. O acesso a esta área também é do tipo externo, com a característica que uma modificação em um dado situado nesta faixa de endereços é efetuada na memória de todas as placas, em controle exclusivo de hardware. Caso seja feita uma escrita na área de "broadcast" e esta for no conjunto que reside nas "caches" das placas, então o conteúdo destas é invalidado, para manutenção da coerência dos dados. Para aumentar a velocidade das transferências em broadcast os ciclos operam em "lock" total, isto é, não podem ser interrompidos por outra U32. O "lock" inclue os ciclos de leitura.

2. 3. A contenção no ambiente PP/U32

A contenção no acesso à memória no ambiente aqui tratado ocorre em dois níveis: barramento global e barramento interno. Para acesso ao barramento global competem os seguintes elementos de todos as U32: o processador 386, o processador T800, o DMA (se presente). A solicitação do barramento interno pode ser da parte do barramento global (isto é, do dono corrente do barramento global) ou dos próprios elementos internos à placa (386, T800, DMA).
Existe ainda um outro nível de contenção possível, na utilização das vias de comunicação do T800, mas isto não será motivo de análise no momento.
Para o controle do acesso à memória existe um mecanismo de "lock" na U32. Este mecanismo pode ser gerado de várias formas: espontaneamente ou programado em algumas instruções do 386, forçado por parte do T800 (que não tem os mecanismos do 386) e automaticamente na utilização do acesso tipo "brodcast". Nos casos onde o mecanismo é devido à instrução do 386 a duração do "lock" é limitada automaticamente, nas operações com "lock" no T800 o mecanismo é controlado por software e, no caso de "broadcast" o "lock" permanece enquanto houver acessos ininterruptos na faixa de endereçamento de broadcast.

3. Aplicações usando memória compartilhada e a placa U32

3. 1. Utilizações a nível de software básico

As facilidades no acesso à memória presente na placa U32 permitem a implementação de mecanismos de sincronização e comunicação em ambiente multiproces-

sado por primitivas software. Um dos mecanismos mais básicos para sincronização é o semáforo. Um semáforo é, em sua definição mais simples, uma variável que contém informação sobre disponibilidade ao acesso de um recurso a ela ligado. Sobre um semáforo são executadas as funções de consulta para conseguir acesso ao recurso controlado e informe de liberação para uso do dito recurso. Estas operações são normalmente chamadas P e V. O semáforo pode ser contador ou binário (assumindo valores 0 e 1 apenas). A variável "semáforo" é trabalhada sob "lock", ou seja, em exclusão mútua. No caso da placa U32, uma variável semáforo localizada na área de "broadcast" terá sua leitura-modificação-escrita de forma indivisível utilizando o barramento global. Entretanto, a leitura para teste da mesma variável "semáforo" na faixa de endereçamento normal não requer a utilização do barramento global (é lida a cópia interna à placa) e em caso de múltiplos leituras sequer utilizará o barramento interno, já que seu valor, até modificação, é contido na "cache".

Outro mecanismo que pode ser implementado a nível software para sincronização é a chamada "barreira".[1] Uma "barreira" é usada para reunir um conjunto de processos em um determinado estado para depois permitir a continuação dos mesmos. A implementação de uma "barreira" pode ser feita via uma variável que conta quantos processos devem alcançá-la e uma "fila dos processos suspensos" ao aguardo de permissão de continuação. Os processos decrementam a barreira quan do chegam no mesmo e continuam unicamente quando o valor da mesma chega a 0.

A expressão "fila de processos suspensos" pode indicar tanto uma fila efetiva gerenciada pelo sistema operacional quanto um breve "loop" software (ou "busy wait" [2]) executado por cada processo que chega na barreira. O segundo caso é mais rápido e é necessário em aplicações fortemente acopladas e com uso exclusivo do processador. É a execução destas repetidas leituras da barreira que cria os chamados "hot spot".

Valem para este caso as mesmas considerações sobre escrita e leitura feitas para as variáveis "semáforos". Ou seja, utilizando a faixa de endereçamento normal, a leitura da variável que indica a sincronização não implica em contenção. Somente a atualização da variável, que será feita na faixa de broadcast, contribue para a contenção.

[1] Obs: há conflito de definição sobre este termo; refira-se a [5] e [11]. A definição a ser utilizada aqui é explicada a seguir.

3. 2. Utilização a nível da aplicação

Do ponto de vista da aplicação, tratando-se aqui preferencialmente de linguagens, o uso de memória comum é bastante vasto. Por um lado, como herança dos modelos de programação clássicos [1], e por outro, como rapidez frente aos mecanismos de troca de mensagens para comunicação entre processos. No uso direto dos recursos de memória compartilhada perde-se a segurança que existe quando se utiliza a máquina através de primitivas fornecidas pelo sistema de software básico (por exemplo, o sistema operacional). No caso da utilização direta é encargo da própria aplicação controlar a alocação dos recursos e os acessos aos mesmos. O ganho mais significativo nesta utilização direta é o aumento de performance. Outro fator de ganho é a flexibilidade e facilidade de programação, existentes em sistemas que se utilizam de memória compartilhada, e sua proximidade de sistemas mono-processados tornam várias aplicações portáveis [1].

Devem ser respeitadas no uso pela aplicação as características dos mecanismos de acesso à memória (acesso internos, externos, na área de "brodcast"), de modo a utilizar da maneira mais coerente as facilidades oferecidas pelo hardware. Ou seja, cuidados devem existir na localização de código, de dados compartilhados frequentemente acessados e na criação de controles como os delineados no item 3.1. No caso de existência de um ambiente misto do ponto de vista da utilização da memória (software básico + aplicação), ambas implementações não se devem interferir. A aplicação deve se valer de primitivas do sistema operacional para delimitar a área de memória de broadcast compartilhada em que pode agir e, caso deseje maior segurança em suas atividades, utilizar também meios do sistema operacional que permitam sua operação em memória comum.

4. Utilização das facilidades da U32 em relação à memória

4. 1. Utilização no suporte do controle de multiprocessamento

As áreas de memória com as facilidades da placa U32 podem ser usadas para manutenção de controle de distribuição de carga entre vários processadores. Neste caso, uma parte da área de "broadcast" seria alocada para conter tabelas que mapeariam a localização física de processos software executados em paralelo com sua identificação lógica. Uma vez que a área de "broad-

cast" é de leitura comum e sem contenção em sucessivas leituras, pode-se eficientemente montar um esquema onde um processo responsável pela comunicação entre unidades processadoras se informa da situação do sistema como um todo e pode, por exemplo, realizar um balanceamento de carga.
A proposta de implementação é a existência de primitivas para manipulação destas tabelas, utilizando parte da área de "broadcast". Estas primitivas auxiliariam a operação de, por exemplo, intercomunicação entre vários elementos processadores (utilizados ou não via barramento comum), sendo suporte às funções descritas em [8].

4. 2. Utilização como primitivas básicas de sincronização

A utilização dos mecanismos de controle hardware da memória para sincronização segue o proposto no item 3.1. Assim, o software básico (sistema operacional) forneceria as primitivas para semáforo, barreira e delimitação da área comum entre a utilizável pela aplicação e a de uso exclusivo do sistema operacional.
A implementação das primitivas poderia ser como segue nos itens abaixo.

4. 2. 1. Semáforo

A descrição que segue supõe que a suspensão e a reativação dos processos é feita pelo sistema operacional.

a - Função inicia_semáforo

- forma de chamada:

  S: = inicia_semáforo (N)

- operação:

  aloca um byte na área de broadcast como variável semáforo

  indica em tabela de uso interno que tal variável está sendo utilizada como semáforo

  inicia a variável com número de esperas possíveis sobre o recurso ao qual vai ser associado (N).

  Devolve ao solicitante a referência na tabela interna de controle da variável "semáforo" (S), ou erro se não houver mais recursos para a alocação de semáforos.

b - Função P(S)

- forma de chamada:

  R: = P(S);

- operação:

  Se a variável tem valor maior que 0

  decrementa de "um" o valor da variável referenciada por "A"

  Caso contrário

  suspende processo corrente

  fim;

c - Função V(S)

- forma de chamada:

  R: = V(S);

- operação:

  Incrementa de "um" o valor da variável semáforo "S".

  Se S>0

  SE houver processos suspensos

  coloca-os para execução;

  fim;

  fim;

  Se não há a variável que referencia o semáforo "S"

  R: = erro;

  fim;

d - Função termina semáforo

- forma de chamada:

  R: = termina(S);

  Devolve a locação "S" ao sistema operacional

  Retira-se sua indicação de uso como semáforo.

4. 2. 2. Barreira

Esta descrição da barreira supõe uma implementação sem intervenção do sistema operacional para suspensão e reativação dos processos.

a - Função Inicia_barreira

- forma de chamada:

  B: = Inicia_barreira (N);

- operação:

  Aloca uma variável para contagem e inicia com N (quantidade de acessas para liberar a barreira)

  B: = índice na tabela de controle;

b - Função chegada_na_barreira

- forma de chamada:

  R: = chegada_na_barreira (B);

- operação:

  decrementa variável de contagem usando locação na faixa de broadcast

  lê variável no endereço normal

  WHILE (variável diferente de 0)

  lê variável no endereço normal

  ENDWHILE

  FIM;

c - Reinicia barreira

- forma de chamada:

  R: = reinicia_barreira(B, N)

- operação:

  Coloca na variável da barreira o valor N (broadcast)

d - Libera barreira

- forma de chamada:

  R:= libera_barreira(B)

- operação:

  Coloca 0 na variável da barreira (broadcast)

  Libera a variável para outras utilizações.

4. 3. Utilização como apoio à aplicação

Para os fins da operação é necessário que se possa particionar a área de interesse para intercomunicação (a área de "broadcast") entre o sistema operacional (ou software básico de controle) e a própria aplicação.
Para isso, a nível de inicialização do sistema deve-se montar uma estrutura onde se tenha uma parte desta área como de uso exclusivo dos mecanismos de software básico, como descrito no item anterior. A nível de operação normal, à operação seriam fornecidas função para alocação de área interna ao seu escopo de ação de alocação, informação em geral sobre esta área, escrita e leitura sobre esta área.
A validade do uso destas áreas deve ser avaliada pela aplicação conforme o melhor paradigma para seu uso. Ver sobre isso [11].

4. 3. 1. Função aloca_área(N)

- forma de chamada:

  R: = aloca_área(N);

- operação:

  Obtém um bloco de tamanho N bytes na área de "broadcast" pertencente à aplicação:

  Marca em tabela interna do sistema operacional o uso desta área e qual o processo proprietário.

  Se alocação com sucesso

R: = referência à localização;

Caso contrário

R: = erro;

Fim;

- OBS:

É possível que toda a área disponível seja alocada de uma só vez, e a gerência se dê exclusivamente pela aplicação.

4. 3. 2. Função dealoca_área

- forma de chamada:

R: = Dealoca_área(área)

- operação:

Devolve área para futura utilização

Retira controles da tabela associada

Responde erro se não há a área especificada (onde "área" deve ser resposta da função descrita em 4.3.1).

4. 3. 3. Função informe_área

- forma de chamada:

R: = Informe_área (tipo, área);

- operação:

Se tipo = total

R tem a quantidade total de área para operação

Caso contrário

R tem o tamanho da área especificada

Fim;

4. 3. 4. Função escreve_em_área

- forma de chamada:

R: = Escreve_em_área (número_bytes, apontador_dados, área);

- operação

Se área existe e tamanho_área>= número_bytes

escreve os dados apontados por "apontador_dados"

na "área" selecionada

R:= operação com sucesso

Caso contrário

R:= falha na escrita

fim

4. 3. 5. Função lê_em_área

- forma de chamada:

R: = lê_em_área (número_bytes,apontador_destino,'area)

- *operação:*

Se área existe e número bytes <=tamanho_área

lê "número bytes" da área especificada

coloca o conteúdo lido em "apontador_destin

Caso contrário

R: = falha na leitura

fim

5. Conclusão

O presente artigo teve como objetivo reunir brevemente os conceitos e problemas ligados à operação em memória compartilhada, principalmente na manutenção de elementos de controle. Com base nesta análise e utilizando os recursos da placa U32 desenvolvida no CPqD Telebrás, busca-se delinear alternativas para implementação de mecanismos de controle de multiprocessamento sob dois pontos principais: o do software básico (sistema operacional) e o da aplicação, inclusive com propostas concretas de utilização.

As ferramentas para utilização dos recursos de memória compartilhada são utilizáveis em um escopo de barramento único. Assim, em uma arquitetura onde se tenha também processamento paralelo ligado por vias outras [7], o software básico deve fornecer primitivas que

permitam também intercomunicação sem utilização de memória compartilhada.
Do ponto de vista da aplicação deve-se entretanto optar por uma visão o mais confortável possível para o uso destas facilidades aqui descritas, juntamente à interconexão não via barramento. Discussões sobre esta abordagem estão em [9] e [11].
A presente discussão deve contudo ser vista no contexto do ambiente "P3" ("Processador Preferencial Paralelo") e portanto integrado às referências [6], [7], [8] e [9].

Referências

[1] "Contention is no obstacle to shared memory multiprocessing"
R.Rettberg e R Thomas
CACM, dezembro 86, vol 26, n 12

[2] "Distributing hot-spot addressing in large scale multiprocessors"
P.Yew, N.Tzeng
IEEE Transaction on Computers, abril 87, vol C-36, n-4

[3] "Hot Spot" Contention and Combining in Multstage Interconnection Networks"
G.Pfister e A.Norton
IEEE Transactions on Computers, outubro 85, vol no.10

[4] "Parallel Supercomputing today and the Cedar approach"
D. Kuck et Al
Science, fevereiro 86

[5] "Syncronization, Coherence and Event ordering in Multiprocessor"
M.Dubois, C.Schenrich,F.Briggs
Computer, fevereiro 88

[6] "Sistema hardware para Processamento Paralelo"
E.Cavalli, M.Zabeu - I Simpósio brasileiro de arquitetura de computadores - Gramado - RS 1987

[7] "Sistema de processamento paralelo - P3"
Relatório interno ao projeto PP-CPqD Telebrás

[8] "Utilização do transputer no Sistema P3". Relatório interno ao projeto PP-CPqD Telebrás

[9] "Adaptação das linguagens Fortran e C para processamento processamento concorrente e paralelo".
Relatório interno ao projeto PP-CPqD - Telebrás.

[10] Especificação PP-U32 - versão preliminar- documentação interno Telebrás

[11] "Programming for Parallelism",
A.Karp, Computer, Março 87