Simulação de Cache em Arquiteturas Configuráveis

JONES OLIVEIRA DE ALBUQUERQUE CLAUDIONOR COELHO JR.
MÁRCIA DE BARROS CORREIA

UFLA - Universidade Federal de Lavras
DCC - Departamento de Ciência da Computação
Cx Postal 37 - 37200-000. Lavras-MG
jones@ufla.br

UFMG - Universidade Federal de Minas Gerais
DCC - Departamento de Ciência da Computação
Cx Postal 702 - 30.123-970. Belo Horizonte - MG
coelho@dcc.ufmg.br

UFPE - Universidade Federal de Pernambuco
Cin - Centro de Informática
Caixa Postal 7851 - 50732-970. Recife-PE
mbc@cin.ufpe.br

Resumo. O uso de um simulador configurável permite reduzir o tempo de projeto de um computador. Neste artigo, é descrita uma série de experiências realizadas com o módulo de emulação da memória do SPARCsim. Em particular, observa-se o comportamento da *cache*.
Abstract: A configurable simulator allows to reduce the computer project time. This paper describes some experiments performed with the Memory Emulation of SPARCsim. For instance, we observe the cache behavior.

Palavras-Chave: arquitetura configurável, simulador SPARCsim, cache.

1 Introdução

Um dos fatores mais importantes para o sucesso de um processador é a redução do tempo de projeto, para que seja possível se tirar partido dos mais recentes avanços em tecnologia de semicondutores. Esta redução tem sido obtida através do uso de poderosas ferramentas de projeto e de simulação.

O sistema SunOS possui um Simulador de Arquitetura SPARC, SPARCsim, que é constituído de módulos configuráveis. Nestes módulos, o projetista pode realizar as modificações desejáveis e depois avaliar o comportamento da máquina na nova configuração.

2 SPARCsim

O Simulador de Arquitetura SPARC (SAS) fornece um ambiente de execução simulado formado por módulos configuráveis. Cada um dos módulos é composto por várias subrotinas responsáveis pela emulação do sistema de arquitetura a ser simulado. Os módulos configuráveis para simulação de arquiteturas são:

- O **Módulo de Emulação de Memória**, possui funções que emulando várias características físicas do sistema de memória, simulam as operações em *caches*, páginas físicas e virtuais de memória.
- O **Módulo de Emulação de Interrupção**, possui funções que emulam as características físicas da estrutura de interrupção, simulando geração de *clocks*, interrupções síncronas e interrupções de E/S.
- O **Módulo de Suporte do Simulador**, possui um *parser* para as opções disponíveis em linha de comando, e funções para simular *set* e *reset* do *hardware* emulado.

Este trabalho descreve uma série de experiências com o **Módulo de Emulação da Memória** visando o estudo comportamental da *cache*.

O Módulo de Emulação de Memória executa as seguintes tarefas:

- Busca a próxima instrução para simulação;

- Lê os dados resultantes da simulação de uma instrução *load* e/ou instrução de busca no programa simulado;

- Escreve os dados resultantes da simulação de uma instrução *store* no programa simulado.

Dependendo da arquitetura a ser simulada, o Módulo de Emulação de Memória pode testar a presença de dados na *cache* simulada, executar a tradução de endereços de memória, emular um dispositivo de I/O ou buscar uma palavra da memória física simulada.

O Módulo de Emulação de Interrupção fornece uma maneira para o sistema simulado gerar interrupções ao processador SPARC. Pode-se emular interrupção assíncrona de discos, portas seriais e outros dispositivos. Estas interrupções podem estar classificadas em níveis de prioridade.

Módulo de Suporte do Simulador fornece serviços auxiliares para o núcleo do simulador. Estas funções são chamadas para:

- Utilizar o *parser* para as opções disponíveis em linha de comando;

- Iniciar e re-iniciar as partes do simulador dependentes da arquitetura;

- Fornecer os estados das partes do simulador dependentes da arquitetura.

2.1 Configurando o Módulo de Memória do Simulador SAS

Os arquivos que constituem o simulador de arquitetura podem ser classificados em dois grupos: os configuráveis e os não configuráveis. Desta maneira, o usuário, para montar seu ambiente de simulação, deve criar *links* para os arquivos não configuráveis e cópias dos módulos configuráveis em seu diretório para poder alterá-los conforme sua necessidade. Todo o procedimento para tal configuração encontra-se em [Oli94].

Dentre as alterações realizadas para a análise de *cache*, as mais relevantes foram implementadas no arquivo **cache.c**.

2.2 Alterando o Arquivo cache.c

As principais modificações que podem ser realizadas com o arquivo cache.c são com respeito a interface, ou seja, melhorar as informações disponíveis para o usuário.

No arquivo cache.c, existe uma função responsável pela impressão de dados que se chama *printCache()*. Através dela pode-se ecoar na tela consideráveis informações sobre os valores disponíveis no programa, como *cache hits* e *cache miss*, tanto para leitura como para escrita.

Também podem ser modificados os tamanhos da cache e dos blocos, e dos parâmetros do SAS. As modificações podem ser feitas através das funções *cacheSetup()* e *cache_arg()*.

As alterações realizadas durante este trabalho consistiram em definir a *cache* com os valores abaixo e chamar o **make** para executar o SAS com os exemplos implementados.

- Tamanho de *cache* = 16, 32, 64, 128 e 256KB com blocos de tamanhos 8, 16, 32 e 64 bytes.

Além disto, foi criado o parâmetro *cachetype 4_660* como uma opção a mais, além das duas já disponíveis na versão padrão, *cachetype 4_260* e *cachetype 4_460*. Para isso foram acrescentadas as seguintes linhas de código ao arquivo **cache.c**:

Como variável global:

```
#define CACHE_660 2
```

Na função *cacheSetup()*:

```
case CACHE_660:
cache_sz = 32*1024;
cache_blksz = 16;
break;
```

E na função *cache_args()*:

```
else if (strcmp(s, "4_660") == 0)
                return(CACHE_660);
```

Com estas modificações, os programas foram compilados e executados no SAS para obtenção dos resultados apresentados neste trabalho.

2.3 Limitações do Simulador SAS

Tentou-se utilizar o SAS para analisar o comportamento da *cache* através de programas maiores e mais complexos do que os apresentados como exemplo, mas em todas as situações o resultado foi sempre o mesmo: *segmentation fault*.

Recompilando o SAS com opção de depuração, o programa foi seguido passo a passo, até chegar ao problema.

O mecanismo do SAS para carregar um programa depende do tamanho deste. Um programa com menos de $64K$ é automaticamente carregado num vetor que o SAS mantém para o código. Se for maior que $64K$, o

SAS tenta colocá-lo em um vetor mantido para dados e para alocação dinâmica. Quando tentávamos fazer isto, um erro na rotina de alocação fazia o programa entrar em *loop* infinito até gerar uma falha de proteção.

Depois de uma série de testes e algum tempo de depuração, encontrou-se a razão do problema:

```
while(index--)
*cp++ = pat;
```

Devido a alguns erros de cálculo, *index* pode assumir valores negativos, gerando um *loop* infinito. Uma maneira de se consertar este erro foi mudar o código para:

```
while((index--)>0)
*cp++ = pat;
```

Após este *patch*, SAS passou a aceitar programas de qualquer tamanho, mas sempre que se tentava executar programas que faziam muito uso de memória ($> 500K$) e em programas que usavam alocação dinâmica, ele disparava a seguinte mensagem:

TRAP AT ADDRESS <NUMERO> (<NUMERO>): DATA_ACCESS_EXC

A instrução correspondente à *trap* foi identificada como sendo a iniciacão do vetor de funções, ou seja, novamente impossibilitando a execução de programas mais complexos.

3 Sistema de Memória

A memória principal pode ser vista como um *buffer* entre o disco e o processador. Existe uma crescente necessidade de aumento de memória e redução do tempo de acesso devido ao aumento da diferença entre as velocidades do disco e do processador. Uma das soluções mais utilizadas para reduzir o tempo de acesso é o uso de sistemas de memória hierarquizados.

Em sistemas hierarquizados, a memória é organizada por níveis de diferentes tipos, de modo a otimizar custo e velocidade. Cada nível é implementado com uma tecnologia diferente, fornecendo diferentes velocidades e capacidades de armazenamento.

As memórias hierárquicas funcionam de modo que as informações são trazidas dos níveis mais lentos para os mais rápidos à medida que vão sendo requisitadas pelo processador. Para fazer face ao aumento da diferença entre as velocidades do processador e da memória principal, os computadores passaram a incluir uma memória de alta velocidade, chamada *cache*, entre a memória principal e o processador, que lhes permite, sem elevar muito o custo, alcançar uma largura de banda compatível com a demanda do processador.

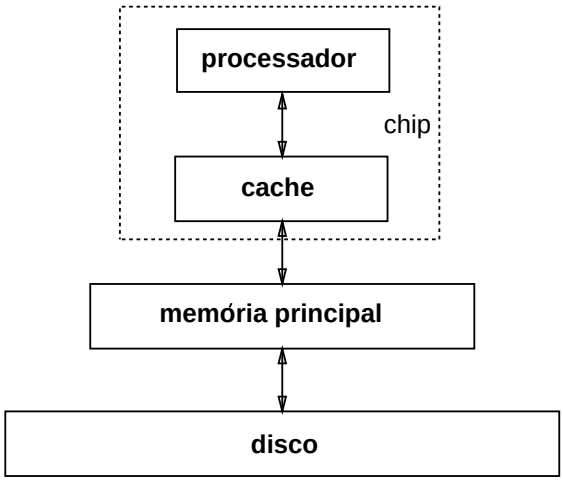

Figura 1: Memória Hierárquica

Para ilustrar o funcionamento de uma memória hierárquica, consideremos um sistema de memória com três níveis: *cache*, memória principal e disco. Em um dado instante, todas as informações precisam estar no disco, algumas na memória principal e umas poucas na *cache*. A CPU busca um objeto na memória principal quando não o encontra na *cache*, e no disco quando não o encontra na memória principal.

3.1 Cache

A memória *cache* funciona como um *buffer* de alta velocidade inserido entre o processador e a memória principal, contendo a parte da memória principal em uso no momento.

Diferentemente de uma falta de página, uma falta de informação na *cache* não provoca chaveamento de processo e deve ser atendida imediatamente para não comprometer o desempenho do sistema. Com o desempenho limitado pelo acesso à *cache*, os computadores recentes têm incorporado diferentes mecanismos de aceleração associados à conversão de endereço e tratamento de faltas.

Uma questão importante em relação à cache é a sua capacidade de armazenamento. Como pode-se observar em [Cor95], existe, na tecnologia de semicondutores, uma incompatibilidade intrínseca entre o aumento de velocidade e a capacidade de armazenamento. O princípio de memória hierárquica, adotado para vencer este impasse, tem sido aplicado com êxito aos projetos de *cache*.

Caches com múltiplos níveis permitem um grande aumento de capacidade com pequenas reduções no

tempo de acesso. Essa solução é particularmente interessante no caso dos microprocessadores onde o primeiro nível de *cache* está implementado no próprio *chip* e, portanto, com grande limitação em capacidade. Como o tempo de acesso do segundo nível é menos crítico, sua capacidade pode ser mais alta.

3.2 Parâmetros Avaliados

A organização da informação dentro da *cache* envolve três parâmetros básicos: tamanho da *cache*, tamanho do bloco e o tipo de mapeamento de endereço utilizado. Estes três parâmetros interferem diretamente no tempo de acesso à cache que por sua vez está fortemente relacionado com o tempo de ciclo do processador, e são, portanto, fundamentais na construção de sistemas de alto desempenho.

Na definição do tempo médio de acessos à *cache*, devem ser considerados a taxa de sucesso, o tempo de acesso no caso da busca ter sucesso e a perda de tempo causada por um insucesso. O aumento do tamanho da *cache* ou do tamanho do bloco reduzem a taxa de insucesso, contudo aumentam o tempo de acesso.

Avaliamos a variação dos números de sucessos e de insucessos para diferentes tamanhos de *cache* e de bloco, utilizando programas simples, mas que se adequam ao nosso propósito.

4 Exemplos e Resultados

Para a obtenção dos dados apresentados nas tabelas foram utilizados três tipos de programas:

- Exemplo recursivo - *fibonacci*
- Exemplo com utilização de um tipo abstrato de dados, vetor - *Acesso Aleatório*
- Exemplo com ponteiros - *lista encadeada com um elemento*

Tentamos um exemplo com ponteiros, mas este gerou *trap*, ora de *overflow* de janela, ora de endereço, pois como compila-se os programas com a opção *-Bstatic* [Oli94], a alocação de memória não se torna disponível dentro do ambiente de simulação.

4.1 Exemplos *.c

Os exemplos, a seguir, foram utilizados variando-se os parâmetros do SAS, tamanho da *cache* e tamanho do bloco. Nesta seção, estão as tabelas com os resultados dos testes e modificações da cache para os programas-exemplo.

Os dados a seguir foram conseguidos utilizando o SPARCsim (SPARC Architectural Simulator) numa **SPARC station SLC** e realizando algumas modificações na função *printCache()* do arquivo cache.c.

4.1.1 Fibonacci

```
int fib();

#include <stdio.h>

int main()
{
int x;
x = fib(15);
return x;
}

int fib(x)
int x;
{
if(x<1) return 0;
if(x==1) return 1;
if(x==2) return 1;
return fib(x-1) + fib(x-2);
}
```

Resultados:

Tamanho da cache	Tamanho do bloco	cache hit	cache miss
16K	8B	44877	190
256K	8B	44878	189
16K	16B	44945	122
256K	16B	44947	120
16K	32B	44994	73
256K	32B	44996	71
16K	64B	45024	43
256K	64B	45026	41

Pode-se notar como a variação do tamanho da *cache* quase não afeta o desempenho, em compensação, a variação do tamanho do bloco tem um efeito bem razoável no número de *misses*.

4.1.2 Acesso Aleatório

Em princípio, íamos testar este programa com um vetor de 1.000.000 de inteiros, mas houve o mesmo problema descrito na seção anterior. Durante a execução deste código no SAS, foram gerados *traps* de *overflow* de janela e de endereço.

```
int vet[100000];

int main()
```

```
{
int i, x, y;
for(i=0;i<100;i++) {
x = rand()%100000;
y = rand()%100000;
vet[x] = vet[y];
}
return vet[x];
}
```

Resultados:

Tamanho da cache	Tamanho do bloco	cache hit	cache miss
16K	8B	46080	373
32K	8B	46088	365
64K	8B	46090	363
128K	8B	46094	359
256K	8B	46094	359
16K	16B	46183	270
256K	16B	46199	254
16K	32B	46244	209
256K	32B	46264	189
16K	64B	46284	169
32K	64B	46296	157
64K	64B	46299	154
128K	64B	46304	149
256K	64B	46304	149

Neste caso, nota-se como o tamanho da *cache* exerce pequena influência no número de *cache misses*, especialmente, quando este é pequeno (16K ou 32K), mas quando o tamanho da *cache* passa de 128K passa a não ter influência alguma.

4.2 Resultados

Aumentando o tamanho dos blocos da *cache* constatamos que o número de *cache hits* também crescia. A justificativa reside no fato de, a cada ciclo, ser carregado na *cache* um bloco de tamanho especificado no tipo de *cache* em uso, assim, à medida que este tamanho aumenta, ocorre o crescimento referenciado, por ser a quantidade de dados colocada na *cache*, por ciclo, cada vez maior, explorando a localidade temporal dos dados.

Outro teste realizado foi avaliar o tamanho da *cache* em relação ao tamanho dos blocos. Em tal situação observamos que ao aumentarmos o tamanho do bloco, o número de cache miss decrescia. O que é natural, pois, com blocos maiores, a necessidade de informação externa ao bloco diminui. Objetivando obter resultados mais expressivos, as atividades foram direcionadas para a busca de um programa teste com maior complexidade, objetivando que este fosse sensível às alterações testadas. Infelizmente, não foi possível a comprovação desta conjectura devido às limitação do SAS em processar um programa maior.

5 Conclusões

O Simulador de Arquitetura SPARC é uma ferramenta importantíssima no desenvolvimento de projetos de arquiteturas. Como observamos, podemos avaliar características da arquitetura antes mesmo de qualquer implementação da mesma, apenas utilizando ferramentas de simulação adequadas.

É bem verdade que estas ferramentas, como o SPARCsim, possuem limitações. Porém, trabalhos de otimização e ampliação dos recursos disponíveis podem ser realizados. Desta forma, podemos constituir um conjunto de ferramentas poderosas, o suficiente, para os propósitos mais complexos.

6 Referências

[Cor94] Correia, Marcia B. Notas de Aula - Arquiteturas SPARC e RS6000.

[Cor95] Correia, Marcia B. Arquiteturas Superescalares. 1998. *Notas de Aula*.

[Goor89] van de Goor A.J. "COMPUTER ARCHITECTURE AND DESIGN ", *Addison Welsley* 1989.

[Hennessy90] Hennessy J. L. e Patterson D. A., "COMPUTER ARCHITECTURE A QUANTITATIVE APPROACH", *Morgan Kaufmann* , 1990.

[Hwang87] Hwang K. e Briggs F. A., "COMPUTER ARCHITECTURE AND PARALLEL PROCESSING", *McGraw Hill International Editions*, 1987.

[Oli94] Oliveira, Adriano L. I. Melo, Jeane C. B. Salgado, Liliane B. Monografia: Sistema de Computação 1. Departamento de Informática - UFPE. 1996.

[Stone90]Stone90 Stone, H.S., "HIGH-PERFORMANCE COMPUTER ARCHITECTURE", *Addison-Wesley Publishing Company*, 9, 1990.

[Sun90] SUN microsystems. 2.1 SPARCsim Configuration Guide. Manual técnico.1990.

[Uni90] UNIX *on-line manual* do sas. 1990.