ARQUITETURA DE COMPUTADORES

2022-2023

Laboratório 5 – Memória Cache

Este laboratório destina-se a consolidar conhecimentos lecionados sobre memórias cache, utilizando o simulador Ripes (https://github.com/mortbopet/Ripes).

Componentes do laboratório:

O laboratório tem duas componentes:

- Exercícios de preparação para o laboratório, que, não sendo avaliados, servem de guia para o trabalho a realizar durante a aula de laboratório. Estes exercícios estão estruturados em duas partes. Na Parte I o trabalho é realizado em Assembly; na parte II o trabalho é realizado em C, fazendo uso de um compilador para RISC-V.
- Trabalho de laboratório, que corresponde a um exercício surpresa e diferente em cada turno de laboratório, cuja complexidade é semelhante aos exercícios de preparação.

Simulador de Cache

Para além de simular diferentes modelos de arquiteturas de processadores, o Ripes integra também um simulador de memórias cache, permitindo estudar o comportamento deste elemento de memória perante o padrão de endereços (de instruções ou de dados) requeridos pela aplicação.

Para garantir uma perfeita compreensão dos recursos e do modo de funcionamento deste simulador de cache, recomenda-se a leitura da respetiva documentação, disponível na seguinte página web: https://github.com/mortbopet/Ripes/blob/master/docs/cache_sim.md

Em particular, o simulador de cache permite a definição de um conjunto alargado de parâmetros de configuração da mesma, nomeadamente:

- O número de vias de associatividade (ways)
- O número de linhas (em cada via)
- A dimensão da linha, o qual é indicada em função do tamanho do registo. Contudo, em ambas as versões (2.2.4 e 2.2.6-11) as designações usadas são diferentes da literatura, i.e.,
    - Ripes 2.2.4 – a dimensão da linha é indicado no campo *Blocks*, sendo 1 block = 32 bits para RV32 ou 1 block = 64 bits para RV64;
    - Ripes 2.2.6-11 – a dimensão da linha é indicada em words/line, sendo que neste contexto 1 word corresponde a 32 bits em RV32 e 64 bits em RV64;
- A política de substituição (apenas para caches associativas) – *Repl. Policy*
- A política de escrita (*Wr. hit*) e alocação de dados (*Wr. miss*) na cache no caso de um store

Permite também a contagem do número de Hits e de Misses, bem como o cálculo do Hit-Rate resultante, à medida que o programa vai sendo executado pelo processador (ver Figura 1).

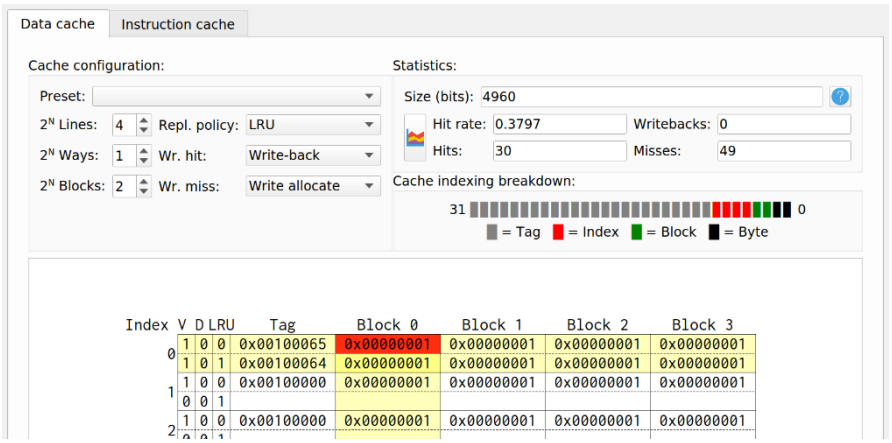


Figure 1 - Configuração e estatísticas de utilização da cache de dados.

Programa a Executar

No presente trabalho será estudada a influência da cache no tempo médio de acesso aos dados presentes na hierarquia de memória ligada a um determinado processador. Em particular, irá ser considerada a implementação de uma operação genérica de multiplicação de matrizes:

$$C = \alpha AB + \beta C$$

Nesta equação, considera-se que **A**, **B** e **C** são matrizes quadradas com NxN elementos, enquanto ⟨ e ® são coeficientes escalares. Todos os operandos são números inteiros de 32-bits. Para o efeito, o seguinte programa (em C) foi traduzido para linguagem Assembly RISC-V, facultado através do ficheiro L5-MM16.s (de notar que a matriz B está já transposta, simplificando a execução do código):

```
for (i=0, checksum=0; i<N; i++){
      for (j=0; j<N; j++){
            for (k=0, sum=0; k<N; k++)
                  sum+= A[i][k]*B[j][k];
            aux = alpha*sum + beta*C[i][j];
            checksum += aux;
            C[i][j] = aux;
      }
}
```

PARTE I

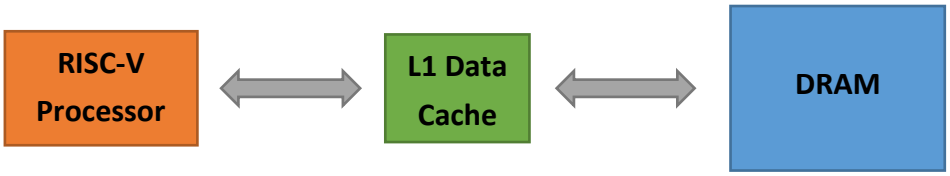
Exercício 1

1. Interprete o código fornecido no ficheiro L5-MM16.s e identifique-o com o programa em C e com a operação algébrica que se pretende realizar.
2. Considere que a execução do programa é realizada sobre um conjunto de matrizes quadradas (NxN) de dimensão fixa, com N=16. Cada elemento da matriz armazena um número inteiro de 32-bits.

3. Assuma que cada matriz é alocada em memória utilizando o padrão *row-major-order*, sendo o elemento A[0,0] armazenado no endereço `10000000h` e o elemento A[0,1] armazenado no endereço `10000004h`. Todas as matrizes são alocadas em espaços de memória adjacentes (i.e.: primeiro A, depois B, etc…).
4. Determine o valor dos endereços (em hexadecimal) dos seguintes elementos das matrizes utilizadas ao longo da execução do programa:

Elemento	Endereço (Hexadecimal)
A[0,15]	
A[1,0]	
A[7,11]	
B[7,11]	
C[7,11]	

5. Considere agora a utilização de um sistema de memória constituído por uma cache de dados (L1) e uma memória primária (DRAM), com as seguintes características:



L1 DATA CACHE	
Nº de vias:	1
Número de linhas:	256
Dimensão da linha(ver nota):	16B (4 palavras)
Política de substituição:	*Least Recently Used*
Política de escrita:	*Write-Back*
Política de alocação:	*Write-Allocate*
Tempo de acesso (t_{Hit}):	1 ns

DRAM	
Dimensão total:	1 GByte
Tempo de acesso:	100 ns

Nota: como o processador foi configurado em modo 32-bits (RV32) o simulador define a dimensão da linha em função de palavras de 4B, a que, na versão 2.2.4, denomina de bloco (note-se que a utilização do termo bloco é contrário à gíria mais usual, que utiliza bloco como sinónimo dos dados de uma linha).

6. Com base na especificação descrita na alínea anterior, e assumindo um barramento de endereços de 32-bits, determine o número de bits associados à etiqueta (*tag*), ao índice (*index*) e ao deslocamento (*offset*) de cada endereço invocado pelo processador. Justifique sucintamente.

Tag	Index	Offset

7. Determine o valor da etiqueta (*tag*), índice (*index*) e deslocamento (*offset*) dos endereços dos elementos das matrizes referidas na alínea 4:

Elemento	Endereço (Hexadecimal)	Tag	Index	Offset
A[0,15]				
A[1,0]				
A[7,11]				
B[7,11]				
C[7,11]				

Exercício 2

1. Configure o simulador Ripes de modo a que seja utilizado o modelo de processador de ciclo único. Para tal, deverá clicar sobre o símbolo do processador do canto superior esquerdo da janela e selecionar a opção *"Single Cycle Processor"*.
2. Abre a vista do simulador que lhe permita observar o funcionamento e a ocupação da memória cache. Execute alguns ciclos de relógio de modo a observar os critérios de preenchimento e de acesso referentes aos primeiros acessos à memória.
3. Execute o programa até ao fim, carregando na tecla >> do simulador
4. Observe e interprete o resultado, consultando as várias regiões da memória de dados (secção .data) de modo a identificar os endereços que guardam a matriz com o resultado da função.
5. Preencha a tabela seguinte:

Nº de Hits:		Nº total de acessos à memória:	
Nº de Misses:		Miss-Rate (total) [%]:	

6. Considerando o sistema de memória definido, determine:

Tempo médio de cada acesso aos dados [ns]:	
Tempo total de acesso aos dados [ms]:	

7. Compare os valores obtidos com aqueles que iria obter caso o sistema de memória não incorporasse a cache (i.e., todos os acessos seriam feitos diretamente à memória):

Tempo médio de cada acesso aos dados [ns]:	
Tempo total de acesso aos dados [ms]:	

Exercício 3

1. Varie agora os parâmetros da cache, aumentando o número de vias para 2, 4 e 8, mantendo a capacidade total da cache e a dimensão da linha. Justifique os resultados no miss-rate total em função do número de vias.

Nº de vias:	1	2	4	8
Número de linhas:	256			
Dimensão da linha:	16B			
Miss-Rate (total) [%]:				

2. Considerando que o número de vias aumenta o tempo de acesso à cache, qual a configuração que escolheria?
3. Varie agora a dimensão do linha para 8B, 32B e 64B, mantendo a capacidade total da cache e o número de vias escolhido no ponto anterior. Justifique os resultados no miss-rate total em função da dimensão da linha.

Nº de vias escolhidas:				
Número de linhas:				
Dimensão da linha:	8B	16B	32B	64B
Miss-Rate (total) [%]:				

4. Considere a melhor configuração de cache obtida em 3.3. Altere agora a dimensão das matrizes de 16x16 para 64x64. Para tal, carregue o ficheiro L5-MM64.s (o qual define N=64). Repita a alínea 1 e comente os resultados obtidos.

Nº de vias:	1	2	4	8
Número de linhas:				
Dimensão da linha:				
Miss-Rate (total) [%]:				

Nota: para melhor compreender a solução desta alínea, sugere-se que repita os pontos 1-4 do exercício 1, tomando agora em consideração que N=64.

PARTE II

A parte II deste enunciado é realizada diretamente a partir do código C. Para tal deverá seguir as instruções apresentadas em:

https://github.com/mortbopet/Ripes/blob/master/docs/c_programming.md

De forma resumida,

1. Fazer download do compilador disponível em: https://github.com/sifive/freedom-tools/releases/tag/v2020.04.0-Toolchain.Only
2. Extraia os ficheiros
3. Configure o Ripes em Edit□Settings□Compiler. Use as seguintes configurações:

Compiler path: Caminho para o gcc, disponibilizado na subpasta "bin". Por exemplo em Windows será o ficheiro "<path base>"\bin\riscv64-unknown-elf-gcc.exe"

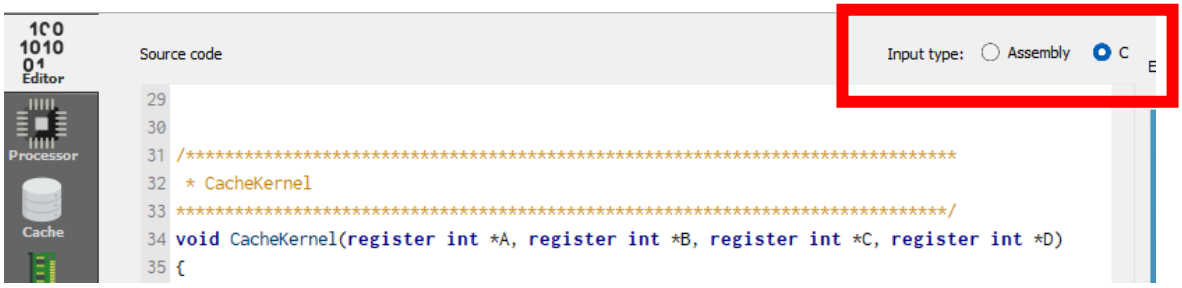
Compiler Arguments: -O0 -g

Linker Arguments: -static -lm

Caso tenha dificuldades em realizar este procedimento no seu computador pessoal, poderá utilizar os computadores da Sala de Computadores do DEEC (SCDEEC), na cave do pavilhão de eletricidade, dias úteis das 8h às 20h.

Exercício 4

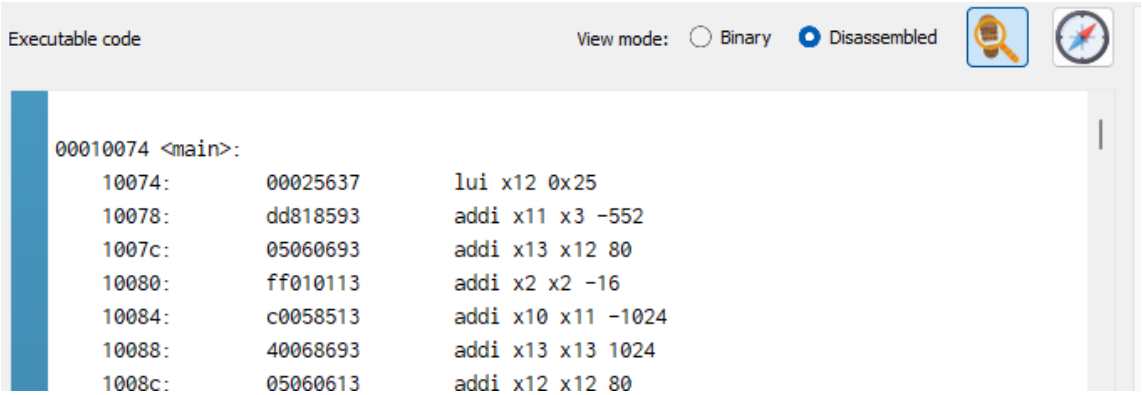
1. Faça download do ficheiro L5-C.c disponibilizado na página do Fenix, junto deste enunciado. Carregue o programa L5-C.c para o Ripes. Não se esqueça de mudar o input type para C.



2. Compile o programa pressionando o botão da barra de ferramentas com o símbolo de martelo:

De notar que neste caso o ponto de entrada do programa (primeira instrução do programa) não corresponde à função `main`, mas sim à função `_start`. Isto deve-se à necessidade de pré-inicializar determinadas posições de memória e/ou registos. Adicionalmente, neste caso, o compilador indica explicitamente no ficheiro compilado (binário) que a primeira instrução do programa se situa numa posição diferente de zero, pelo que quando o Ripes carrega o programa (faz *load* do binário), coloca PC com o endereço relativo à função `_start`.

3. Analise o binário resultante.



4. Identifique o endereço de cada uma das rotinas Assembly (i.e., da primeira instrução da rotina) do programa compilado. Preencha a tabela indicada em baixo.

Elemento	Endereço (Hexadecimal)
<_start>	
<main>	
<CacheKernel>	
<VerifyResult>	
<exit>	

5. Configure o processador para o modo RV32 em ciclo único e a cache com as seguintes configurações:

L1 DATA CACHE	
Nº de vias:	2
Número de linhas:	32
Dimensão da linha:	16B (4 palavras)
Política de substituição:	*Least Recently Used (LRU)*
Política de escrita:	*Write-Back*
Política de alocação:	*Write-Allocate*

6. Através da análise do código C, obtenha a estimativa para o miss-rate da função `CacheKernel`. Admita que a cache começa não inicializada.
7. Coloque um ponto de paragem na chamada à função `CacheKernel` (i.e., procure na função `main` a instrução `jal` que chama `CacheKernel`). Execute o programa em modo simulação rápida sem atualização do ecrã (deve usar o botão com o símbolo "»" na barra de ferramentas).
    a. Anote o número de hits e misses na cache à entrada da função.
    b. Qual a razão para os números apresentados serem diferentes de 0?

8. Faça reset à simulação, retire o ponto de paragem anterior, e coloque um novo ponto de paragem no final da função `CacheKernel` (i.e. no `jalr` que realiza o retorno da função).
    a. Anote o número de hits e misses na cache à saída da função.
    b. Determine a taxa de falhas na cache (*miss rate*) fazendo a diferença entre os valores observados em 8a e 7a, de forma a determinar o número total de hits e misses da rotina `CacheKernel`. Compare com a sua estimativa em 6.
9. Proponha alterações à rotina `CacheKernel` de forma a reduzir a taxa de falhas na cache (*miss rate*). Valide a correta execução do programa retirando todos os pontos de paragem e correndo a simulação em modo rápido. A função `VerifyResult` irá indicar se o resultado está correto.
10. Analise a função `CacheKernel` original (i.e., sem as alterações ao código propostas no ponto anterior). Mantendo a capacidade total da cache L1 de dados (L1-D), modifique a configuração desta cache de forma a minimizar a taxa de falhas. Justifique todas as alterações feitas.

    Nota: o Ripes não considera o uso de um write-buffer na implementação da política write-through, no write allocate. Assim, ao contrário do habitual, um write access conta como um hit ou miss, e um acesso.
11. Repita a alínea anterior considerando as alterações ao código propostas no ponto 9.