NOVA SCHOOL OF SCIENCE & TECHNOLOGY

# Arquitetura de Computadores 2022/23

## Ficha 8

***Tópicos***: *Hierarquia de memória: Caches.*

## Introdução

A matéria teórica ilustrada nesta aula prática corresponde às aulas teóricas sobre caches. A matéria encontra-se no capítulo 11 do livro *Dive into Systems* e também no capítulo 6 do livro *Computer Systems: A Programmer´s Perspective 2nd Ed.* Todo o código necessário para este trabalho é fornecido via CLIP. Para compilar basta, no *shell* do terminal, dar o comando `make`. Este usa as definições em `Makefile` para compilar os ficheiros necessários, e só esses.

## Caches de uma arquitetura real

Usando dois programas em C cujo código fonte está disponível no CLIP pretende-se que obtenha informações sobre a hierarquia de memória do processador do seu computador e das velocidades de acesso à memória que estão associadas a cada um dos níveis de cache e da memória central.

### Informação disponibilizada pelo comando *getconf -a*

Execute, no shell, o comando:

```
getconf -a | grep CACHE
```

e anote a informação fornecida.

## Percurso de uma matriz por linhas e por colunas

Considere a seção 11.5 do livro *Dive into Systems* sobre a análise da cache. Execute o programa de cálculo da média dos valores de uma matriz tal como sugerido nas seções 11.5.1 e 11.5.2.

## Memory mountain

Vamos agora usar o programa *mountain* discutido no livro *Computer Systems: A Programmer´s Perspective 2nd Ed* (ver sec. 6.6.1). O objetivo é avaliar a taxa de transferência CPU/memória (*memory throughput* ou *bandwidth*) do seu computador e avaliar os vários níveis de cache. Neste programa são medidas as transferências (MB/s) para vários padrões de acesso à memória, simulando diferentes níveis de localidade temporal e espacial. Para tal é medido o tempo que uma função demora a somar, repetidamente, todos os valores de um vetor de inteiros. Dois parâmetros ajustam o perfil dos acessos a memória: o tamanho do vetor e o intervalo entre acessos consecutivos (*stride*). Um vetor menor leva a mais acessos sempre na mesma zona, logo, melhor localidade temporal. Um menor *stride*, corresponde a acessos consecutivos mais próximos, logo, melhor localidade espacial; no limite, com *stride* de 1, temos acessos sequenciais.

Compile e execute o programa. Este irá reportar as várias taxas de transferência para os diferentes parâmetros de tamanho do vetor e de *stride*.

Note que o output de um programa pode ser redirigido do ecrã/terminal para um ficheiro acrescentando "`> nome_do_ficheiro.txt`" à linha de comando em que o programa é executado. Por exemplo:

```
mountain > resultados.txt
```



Campus'de'Caparica'
2829'516'CAPARICA'
Tel:'+351'212'948'536'
Fax:'+351'212'948'541'
di.secretariado@fct.unl.pt'
'
www.fct.unl.pt'

Este ficheiro de output pode depois ser importado para outro programa, como uma folha de cálculo, para facilitar a análise dos dados. Com uma folha de cálculo ou outro programa similar, gere um gráfico (p.e. 3D) que relacione estes valores. Deve obter um gráfico com o aspeto semelhante ao de uma montanha (daí o nome do programa). Veja o exemplo na capa do livro da bibliografia ou na secção 6.6.1.

Procure responder às perguntas através da análise do gráfico e compare com os valores obtidos de início:
- Quantos níveis de cache tem o seu computador?
- Faça uma estimativa da dimensão de cada nível.
- Qual é a dimensão de uma linha da caches (blocos)? Não se esqueça que é uma potência de 2.

## Simulação de cache de mapa direto

Veja agora a diretoria *cacheSim* que contém um simulador de cache de memória. Este usa uma organização em mapa direto e cada linha guarda um bloco de 32 bytes da memória. O simulador recebe como argumento um ficheiro com um traço (registo) dos endereços de memória acedidos durante a execução de um programa. O ficheiro com o traço dos endereços de memória acedidos é um ficheiro de texto com o seguinte formato (exemplo):

```
0041f7a0 R
31348900 W
00347b00 F
```

Em cada linha, o primeiro valor é um inteiro (hexadecimal) com o endereço de memória a que o CPU acede. A letra representa o tipo de operação realizada naquele endereço de memória: R (leitura), W (escrita) ou F (*fetch*, leitura de uma instrução). O programa fornecido simula as ações do *hardware* que permitem decidir se cada acesso terá sucesso na cache (hit) ou não (miss) e efetuar a substituição de linhas quando necessário. Deve sempre incrementar os contadores de números de acessos a memória e *cache miss*, de acordo com as operações realizadas na cache.

Em `cache.c` complete a função `simulateOneStep` que verifica se determinado endereço está ou não na cache, devolvendo `HIT` ou `MISS`, consoante o caso e atualiza a cache de acordo com o funcionamento das caches de mapa direto. Aquando de um *miss*, quer numa leitura (R ou F) quer numa escrita, o bloco de memória será sempre lido para a respetiva linha de cache. O tratamento das escritas deve ser *write through*, ou seja, serão sempre feitas imediatamente na memória.

Uma vez completada a implementação e compilado o programa, pode executar uma simulação indicando o ficheiro com o traço de execução. Exemplo:

```
cacheSim bzip.trace
```



Campus'de'Caparica'
2829Ø516'CAPARICA'
Tel:'+351'212'948'536'
Fax:'+351'212'948'541'
di.secretariado@fct.unl.pt'
'
www.fct.unl.pt'