Guião Laboratorial 2

Hierarquia de Memória – Localidade dos dados

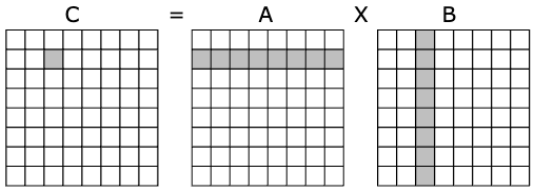
Objetivos:
- introdução à localidade espacial e temporal dos dados em memória
- exploração de técnicas para melhorar a localidade dos dados

Introdução
A hierarquia de memória é eficaz na redução dos tempos de acesso aos dados quando os padrões de acesso (tanto a dados como instruções) exibem algum tipo de localidade. A localidade espacial caracteriza-se por acessos consecutivos à memória endereçarem células de memória contíguas. A localidade temporal implica múltiplos acessos aos mesmos endereços de memória num curto intervalo de tempo.

Este guião está dividido em exercícios a serem realizados durante a sessão laboratorial (**Exercício 1-3**) e um exercício extra a ser submetido na plataforma *BlackBoard* (**TPC 1**). Os ficheiros necessários encontram-se na diretoria `/share/cpar/PL02_Codigo.zip` do cluster SeARCH, e deverão ser copiados para a *home*.

Exercício 1 – Análise da localidade dos dados
Em C as matrizes são armazenadas em memória em *row major order*, isto é, elementos consecutivos da mesma linha são armazenados em posições contíguas de memória. A multiplicação de matrizes, no código já disponibilizado, utiliza o seguinte algoritmo para calcular $C = A * B$, sendo A, B e C matrizes quadradas com N linhas (e N colunas):

```
/* Cij =0 */
para i=0 até N-1
      para j=0 até N-1
           para k= 0 até N-1
                C[i][j] += A[i][k] * B[k][j]
```



a) Indique quais as matrizes cujos acessos exibem localidade espacial e quais as que não exibem.
b) Estime a quantidade de L1 *cache misses* (versão `gemm1`) e preencha a respetiva coluna na Tabela 1, assumindo que as matrizes não cabem em *cache*.
c) Será possível aumentar os acessos que exibem localidade espacial trocando a ordem dos ciclos i, j, k?

Exercício 2 – Medição do impacto da localidade espacial dos dados
A transposta da matriz M, designada por M^T, obtém-se trocando as linhas de M pelas suas colunas. Por exemplo:

$$M = \begin{bmatrix} 3 & 6 \\ 8 & 0 \\ 1 & 12 \end{bmatrix}; M^T = \begin{bmatrix} 3 & 8 & 1 \\ 6 & 0 & 12 \end{bmatrix}$$

Se modificarmos o nosso caso de estudo para calcular $C = A * B^T$, então podemos usar o algoritmo acima modificando apenas a ordem com que B é visitada. Vamos percorrer as linhas de B (que corresponderão às colunas de B^T) passando a exibir localidade espacial no acesso a B.

Altere os ficheiros disponibilizados:

1. Em `gemm.c` copie o código de `gemm1` para `gemm2`, e altere o cálculo de *cij* de forma a que percorra B^T e não B, isto é, aceda consecutivamente a todos os elementos da mesma linha de B (`b[j][k]`), em vez de todos os elementos da mesma coluna (`b[k][j]`).
   Note que para matrizes com n colunas em *row major order*, temos `m[l][c] == m[l*n+c]`.

2. Altere o parte inicial do ficheiro `main.c` para medir também os seguintes contadores:
   - `PAPI_L1_DCM` – conta as *cache misses* na *cache* de dados nível 1;
   - `PAPI_L2_DCM` – conta as *cache misses* na *cache* de dados nível 2;

   Nota: basta alterar as linhas 20 e 21 que indicam os eventos a medir:
   ```
   #define NUM_EVENTS 2
   int Events[NUM_EVENTS] = { PAPI_TOT_CYC, PAPI_TOT_INS };
   ```

a) Estime a quantidade de L1 *cache misses* (versão `gemm2`) e preencha a respetiva coluna na Tabela 1.

b) Meça os eventos e preencha a Tabela 1 para n=256 e n=512 (sugestão: use uma tabela Excel para calcular o CPI). Verifica alguma melhoria de desempenho com a versão transposta? Como a justificaria?

N		#CC	CPI	#I	L1_DCM *estimados*	L1_DCM	L2_DCM
256	`gemm1()`						
	`gemm2()`						
512	`gemm1()`						
	`gemm2()`						

Tabela 1 - Localidade espacial

Exercício 3 – Medição do impacto da localidade temporal dos dados

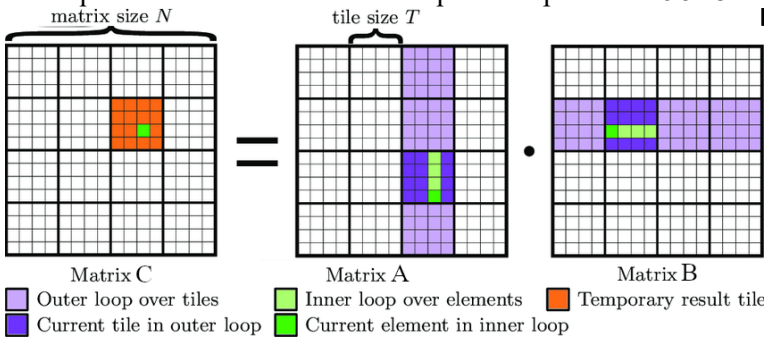
Execute a versão `gemm1()` (`srun --partition=cpar ./gemm 32 1`) da multiplicação de matrizes e preencha a Tabela 2. Como aumentam as várias grandezas *(#CC, CPI, #I, misses/#I*) em função do tamanho da matriz? Explique porquê (nota: verifique as características da *cache* com o comando `srun --partition=cpar papi_mem_info`).

n	#CC	CPI	#I	L1_DCM	L2_DCM	L1_DCM/#I	L2_DCM/#I
32							
64							
128							
256							
512							

Tabela 2 - Localidade temporal em função do tamanho da matriz

TPC 1 – Otimização da localidade dos dados em memória

Complete a função `gemm3` no ficheiro `gemm.c` para que implemente o cálculo da multiplicação de matrizes com recurso a *tiling*. Garanta que o tamanho da matriz é sempre múltiplo de `BLOCKSIZE`.



Preencha a Tabela 3 com n = 512. Para `gemm3()` altere a definição de `BLOCKSIZE` e recompile o programa (`make`).

	BLOCKSIZE	#CC	#I	CPI	L1_DCM
`gemm1()`	---				
`gemm3()`	8				
	16				
	32				
	64				

Tabela 3 - Localidade temporal

Comente o impacto do tamanho do bloco nas várias grandezas (#CC, #I, e CPI) e na quantidade de *misses* à cache. Pode assinalar com um círculo os valores da tabela relevantes para a justificação. Respostas que não sejam manuscritas serão ignoradas.

R: _____

_____

_____

_____

A submissão do TPC implica a impressão desta folha, preenchimento da tabela e comentários, e a sua digitalização e submissão na plataforma *BlackBoard* até ao dia **25 de outubro**.