CENTRO TECNOLÓGICO
Universidade Federal de Santa Catarina

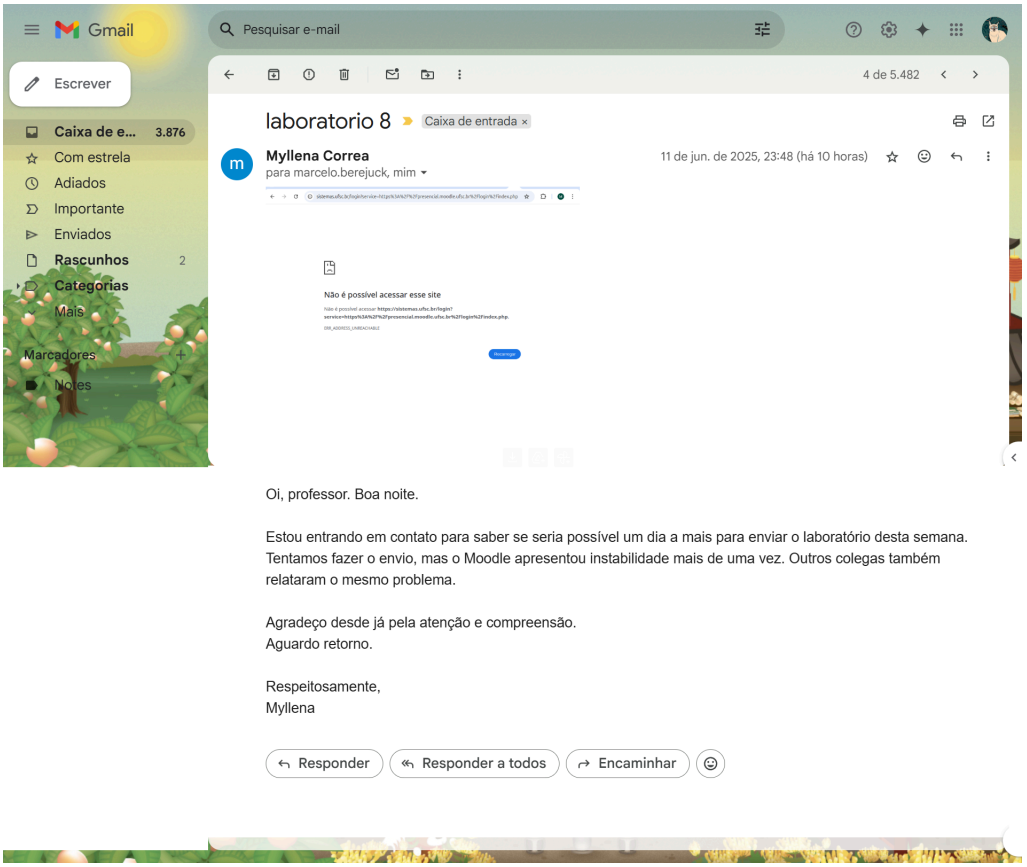
Laboratório 08

Cache Blocking

Alunas: Jéssica Regina dos Santos e Myllena Corrêa
Disciplina: Organização de Computadores I
Professor: Marcelo Daniel Berejuck
Entrega: 11/06/25

Anexo

Sobre o atraso na entrega deste laboratório:



Pedimos desculpas pelo inconveniente. Realizamos a entrega assim que possível (quando o Moodle voltou ao normal, no dia seguinte).

Introdução

Principais objetivos:

- Implementação e análise de algoritmos de soma de matrizes em linguagem MIPS, com ênfase na aplicação da técnica de *cache blocking* (por meio da comparação entre uma abordagem tradicional e outra otimizada com *blocking*).
- Buscamos compreender o impacto dessa técnica no desempenho de acesso à memória cache

Os experimentos foram realizados utilizando a ferramenta MARS, com apoio do Data Cache Simulator, permitindo observar e discutir a eficiência de diferentes configurações de tamanho de matrizes e blocos.

Questão 01

A seguir, explicaremos a estrutura do primeiro código implementado.

Objetivo: soma de uma matriz com outra matriz transposta.

```
.eqv MAX 8 # Criando uma constante simbolica, nao fica em nenhum registrador
.eqv TOTAL_BYTES 256 # 8*8*4
```

A diretiva .eqv foi usada para criar constantes simbólicas. MAX define a dimensão das matrizes, usando (8x8) como exemplo e TOTAL_BYTES pré-calcula o espaço total em bytes necessário para uma matriz (8 * 8 * 4 = 256). Isto é feito porque a diretiva .space ou a inicialização de dados precisa de um valor fixo.

```
.data # Declaracao das variaveis

A: .space 256
B: .space 256
```

O segmento .data possui os rótulos A e B, que marcam o “espaço ocupado” por cada matriz.

```
18  .text
19  .globl main
20
21  main:
22          la $t0, A           # $t0 <- Endereço de A
23          la $t1, B           # $t1 <- Endereço de B
24          li $t2, MAX         # $t2 <- Valor de MAX (5)
25          li $t3, 0           # $t3 <- Contador i, inicializado com 0.
26
```

L22 e L23) Carregam os endereços base das matrizes A e B nos registradores $t0 e $t1.
L23) Carrega o valor da constante MAX em $t2 para ser usado nas condições dos laços e nos cálculos.
L25) O contador do laço externo, i, é inicializado em $t3.

```
27  # Laço Externo (i)
28  loop_externo:
29          # Se i ($t3) >= MAX, pula para o fim do programa.
30          bge $t3, $t2, fim_do_programa
31
32          # Inicializa o contador j. Isso deve ser feito a cada iteração de i.
33          li $t4, 0           # $t4 <- Contador j, inicializado com 0.
34
35  # Laço Interno (j)
36  loop_interno:
37          # Se j ($t4) >= MAX ($t2), pula para o fim do laço interno.
38          bge $t4, $t2, fim_loop_interno
39
40          # Calcular endereço de A[i,j] -> base(A) + (i * MAX + j) * 4
41          mul  $t5, $t3, $t2      # $t5 = i * MAX
42          add  $t5, $t5, $t4      # $t5 = i * MAX + j
43          sll  $t5, $t5, 2        # $t5 = (i * MAX + j) * 4
44          add  $t6, $t0, $t5      # $t6 = Endereço de A[i,j]
45
46          # Calcular o endereço de B[j,i] -> base(B) + (j * MAX + i) * 4
47          mul  $t7, $t4, $t2      # $t7 = j ($t4) * MAX ($t2)
48          add  $t7, $t7, $t3      # $t7 = (j * MAX) + i ($t3)
49          sll  $t7, $t7, 2        # $t7 = (j * MAX + i) * 4
50          add  $t8, $t1, $t7      # $t8 = Endereço de B[j,i]
51
52          # Realizar a operação com ponto flutuante (float)
53          l.s  $f0, 0($t6)        # Carrega o valor de A[i,j]
54          l.s  $f2, 0($t8)        # Carrega o valor de B[j,i]
55          add.s $f4, $f0, $f2     # Soma os dois valores
56          s.s  $f4, 0($t6)        # Armazena o resultado de volta em A[i,j]
57
58          addi $t4, $t4, 1        # Incrementa j (j++)
59          j    loop_interno       # volta para o início do laço interno
60
61  fim_loop_interno:
62          # Isso só acontece quando o laço interno (j) termina.
63          addi $t3, $t3, 1        # Incrementa i (i++)
64          j    loop_externo       # Pula de volta para o início do laço externo
65
```

L28) "loop_externo" controla o índice i em $t3. Então, a instrução bge verifica se i >= MAX. Caso seja verdade, encerra o programa.
L36) "loop_interno" controla o índice j em $t4, que é reiniciado a cada iteração do laço externo. Então, o bge verifica se j >= MAX para encerrar o laço interno.
Ao final de cada laço, o respectivo contador é incrementado (i++ ou j++) e um salto (j) retorna ao início do laço para a próxima iteração.

Para cada elemento, o código calcula seu offset (deslocamento) a partir do endereço base.

L40) A[i,j]: Implementa a fórmula **endereço = base + (i * MAX + j) * 4.**
L46) B[j,i]: Implementa a fórmula da transposta, **endereço = base + (j * MAX + i) * 4.**
As instruções mul, add e sll, são usadas para executar esses cálculos e o endereço final de cada elemento é armazenado em $t6 e $t8.

L52) Com os endereços corretos em mãos, este bloco executa a soma.
l.s: carrega os valores float da memória (dos endereços em $t6 e $t8) para os registradores do coprocessador de ponto flutuante, $f0 e $f2.
add.s soma os dois valores e armazena o resultado em $f4.
s.s salva o resultado de volta na memória, no endereço de A[i,j] que está em $t6.

```
65
66  fim_do_programa:
67          li   $v0, 10
68          syscall
```

Por fim, é finalizado o programa. A instrução li carrega o código 10 (Exit) no registrador $v0 e syscall faz uma chamada ao sistema operacional para encerrar a execução.

Questão 2

Nessa questão, implementamos novamente uma soma de uma matriz com uma matriz transposta, mas, desta vez, utilizando a técnica de *cache blocking*.

```
.eqv MAX 8
.eqv BLOCK_SIZE 4
```

MAX define a dimensão da matriz (8x8).
BLOCK_SIZE: Define o tamanho do sub-bloco que será processado por vez (4x4). A lógica do programa assume que MAX é um múltiplo de BLOCK_SIZE.

```
.data

A: .space 256
B: .space 256
```

No segmento .data, os espaços para as matrizes A e B são declaradas novamente.

```
24      .float 57, 58, 59, 60, 61, 62, 63, 64
25
26  .text
27  .globl main
28
29  main:
30
31      la   $t0, A             # Endereço base de A
32      la   $t1, B             # Endereço base de B
33
34      li   $s0, MAX           # Carrega MAX em um registrador salvo
35      li   $s1, BLOCK_SIZE    # Carrega BLOCK_SIZE em um registrador salvo
36
37      li   $s2, 0             # i = 0
38
```

L31 a L37) O programa começa carregando valores essenciais em registradores.Carrega os endereços de memória iniciais das matrizes A e B nos registradores $t0 e $t1. Carrega as constantes MAX e BLOCK_SIZE nos registradores salvos $s0 e $s1. O primeiro contador de laço, i ($s2), é inicializado com 0.

```
38
39  # --- Laço 1 (for i = 0; i < MAX; i += BLOCK_SIZE) ---
40  loop_i:
41      bge  $s2, $s0, end_program
42      li   $s3, 0             # j = 0
43
44  # --- Laço 2 (for j = 0; j < MAX; j += BLOCK_SIZE) ---
45  loop_j:
46      bge  $s3, $s0, end_loop_j
47
48      # Calcula o limite do laço 'ii' -> i + BLOCK_SIZE
49      add  $t9, $s2, $s1      # $t9 = i + BLOCK_SIZE
50      move $s4, $s2           # ii = i
51
52  # --- Laço 3 (for ii = i; ii < i + BLOCK_SIZE; ii++) ---
53  loop_ii:
54      bge  $s4, $t9, end_loop_ii
55
56      # Calcula o limite do laço 'jj' -> j + BLOCK_SIZE
57      add  $t9, $s3, $s1      # $t9 = j + BLOCK_SIZE (reutilizando $t9)
58      move $s5, $s3           # jj = j
59
60  # --- Laço 4 (for jj = j; jj < j + BLOCK_SIZE; jj++) ---
61  loop_jj:
62      bge  $s5, $t9, end_loop_jj
63
64      # --- Corpo do Laço: A[ii,jj] = A[ii,jj] + B[jj,ii] ---
65      # Endereço de A[ii,jj] = base(A) + (ii * MAX + jj) * 4
66      mul  $t5, $s4, $s0      # $t5 = ii * MAX
67      add  $t5, $t5, $s5      # $t5 = ii * MAX + jj
68      sll  $t5, $t5, 2        # offset em bytes
69      add  $t6, $t0, $t5      # Endereço final de A[ii,jj]
70
71      # Endereço de B[jj,ii] = base(B) + (jj * MAX + ii) * 4
72      mul  $t7, $s5, $s0      # $t7 = jj * MAX
73      add  $t7, $t7, $s4      # $t7 = jj * MAX + ii
74      sll  $t7, $t7, 2        # offset em bytes
75      add  $t8, $t1, $t7      # Endereço final de B[jj,ii]
76
77      # Operação com floats
78      l.s  $f0, 0($t6)        # Carrega A[ii,jj]
79      l.s  $f2, 0($t8)        # Carrega B[jj,ii]
80      add.s $f4, $f0, $f2     # Soma
81      s.s  $f4, 0($t6)        # Salva o resultado em A[ii,jj]
82
```

```
82
83      addi $s5, $s5, 1        # jj++
84      j    loop_jj
85
86  end_loop_jj:
87      addi $s4, $s4, 1        # ii++
88      j    loop_ii
89
90  end_loop_ii:
91      add  $s3, $s3, $s1      # j += BLOCK_SIZE
92      j    loop_j
93
94  end_loop_j:
95      add  $s2, $s2, $s1      # i += BLOCK_SIZE
96      j    loop_i
97
```

Acima, estão os dois laços mais externos, responsáveis por iterar sobre os blocos da matriz.
L40) "loop_i" itera sobre as linhas dos blocos. O contador i ($s2) avança em passos de BLOCK_SIZE.
L45) "loop_j" itera sobre as colunas dos blocos. O contador j ($s3) também avança em passos de BLOCK_SIZE. Juntos, eles selecionam o bloco (i, j) que será processado pelos laços internos.

L53) "loop_ii" itera sobre as linhas dentro do bloco. O contador ii ($s4) vai de i até i + BLOCK_SIZE - 1.
L61) "loop_jj" itera sobre as colunas dentro do bloco. O contador jj ($s5) vai de j até j + BLOCK_SIZE - 1. O limite de cada laço é pré-calculado e armazenado temporariamente em $t9.

O programa então calcula o endereço exato na memória para A[ii,jj] e B[jj,ii] aplicando a fórmula **endereço = base + (linha * MAX + coluna) * 4**, que é implementada usando as instruções:

mul: Multiplica linha * MAX.
add: Adiciona + coluna.
sll: Multiplica o resultado por 4 (tamanho de um float) de forma eficiente.
add: Soma o deslocamento (offset) ao endereço base da matriz.

L62) Por fim, são carregados os valores float de A[ii,jj] e B[jj,ii] da memória para os registradores de ponto flutuante $f0 e $f2.

add.s: Soma os dois valores e armazena o resultado em $f4.
s.s: Salva o resultado de $f4 de volta na posição de memória de A[ii,jj].

```
98  end_program:
99      li   $v0, 10
100     syscall
```

Quando TODOS os laços terminam, o programa chega a este ponto.
A syscall de código 10 em $v0 é chamada para encerrar a execução.

Questão 3

Para esta questão, foram feitos diversos testes com tamanhos de cache, blocos de cache e tamanho de matrizes diferentes. Abaixo, temos a tabela de resultados para comparação:

Tamanho da Matriz	Tamanho da Cache	Tamanho do Bloco (Cache Blocking)	Taxa de Acertos
2x2	4x4	N/A	83%
2x2	4x4	1	93%
2x2	4x4	2	95%
4x4	8x4	N/A	83%
4x4	8x4	2	92%
4x4	16x8	N/A	92%
4x4	16x8	2	96%
8x8	16x8	N/A	95%
8x8	16x8	4	97%

Interpretação da tabela:

A simulação comparativa comprovou que a técnica de *cache blocking* é superior ao algoritmo normal, resultando em maiores taxas de acertos na cache.

Isso ocorre pois, como estudamos, a técnica otimiza a localidade temporal, processando a matriz em blocos pequenos que cabem na cache e maximizando o reuso de dados.

O método “normal” acessa a memória de forma dispersa, causando muitos cache misses, um problema que piora com matrizes maiores.

Como esperado, os experimentos validaram o *cache blocking* como uma estratégia de otimização essencial para o processamento eficiente de grandes volumes de dados.