UNIVERSIDADE FEDERAL DE SANTA CATARINA
EDUARDO BISCHOFF GRASEL

Relatório Organização de Computadores

Florianópolis - SC
2024

Parte 1:

Começamos definindo no .text o array com um espaço de 16x16x4 posições para referenciar nossa matriz, logo abaixo definimos o número de linhas e colunas.

```
.data
        A:      .space 1024         # espaço reservado para a matriz A (exemplo: 16x16x4 inteiros)
        m:      .word 16            # número de linhas
        n:      .word 16            # número de colunas
```

No início do .text carregamos os valores da memória para registradores e também definimos um inicializador ‘i’ (index linha) e um contador.

```
.text
        la      $t0, A          # carregar o endereço base da matriz A em $t0
        lw      $t1, m          # carregar o número de linhas em $t1
        lw      $t2, n          # carregar o número de colunas em $t2

        li      $t3, 0          # inicializar i = 0
        li      $t8, 0          # inicializar contador = 0
```

Com isso já podemos entrar na lógica de criação da matriz, a qual se dará por um loop externo responsável por percorrer as linhas, criar o inicializador ‘j’ (index coluna) e um loop interno responsável por percorrer as colunas, as condições de saída de cada loop é: caso o número de loops tenha alcançado o total de linhas ou colunas.

Dentro do inner_loop, teremos a lógica de cálculo de endereços e preenchimento da matriz, o qual foi recomendado linha por linha logo abaixo, mas resumidamente ele calcula o offset da linha, somar com a coluna (j) e multiplicar por 4, que é o tamanho da palavra. Depois guardamos o contador no endereço, incrementamos o contador, incrementamos a coluna, se for o último loop incrementamos as linhas, e seguimos esse padrão até acabar as iterações do loop externo.

```
outer_loop:
        bge     $t3, $t1, end_outer_loop   # se i >= m (linhas), sair do loop externo

        li      $t4, 0          # inicializar j = 0

inner_loop:
        bge     $t4, $t2, end_inner_loop   # se j >= n, sair do loop interno

        mul     $t5, $t3, $t2           # t5 = i * n
        add     $t5, $t5, $t4           # t5 = i * n + j
        sll     $t5, $t5, 2             # t5 = (i * n + j) * 4 (multiplicar por 4, tamanho de palavra)
        add     $t6, $t0, $t5           # t6 = endereço de A[i][j]

        sw      $t8, 0($t6)             # armazenar contador em A[i][j]
        addi    $t8, $t8, 1             # contador++

        addi    $t4, $t4, 1             # j++

        j       inner_loop              # repetir o loop interno

end_inner_loop:
        addi    $t3, $t3, 1             # i++
        j       outer_loop              # repetir o loop externo

end_outer_loop:
        # fim do programa
        li      $v0, 10                 # código para sair do programa
        syscall
```

Parte 2:

Como a parte dois é praticamente igual, mudando apenas a lógica de criação da matriz, onde ao invés de ser criada com um loop externo preenchendo as linhas e um loop interno preenchendo as colunas, será ao contrário, portanto apresentarei aqui apenas a parte do código que foi modificada de modo a inverter a lógica a fim de percorrer externamente as colunas e internamente as linhas.

```
outer_loop:
        bge     $t3, $t2, end_outer_loop   # se j >= n, sair do loop externo

        li      $t4, 0        # inicializar i = 0 (linha)

inner_loop:
        bge     $t4, $t1, end_inner_loop   # se i >= m, sair do loop interno

        mul     $t5, $t4, $t2             # t5 = i * n
        add     $t5, $t5, $t3             # t5 = i * n + j
        sll     $t5, $t5, 2               # t5 = (i * n + j) * 4 (multiplicar por 4, tamanho de palavra)
        add     $t6, $t0, $t5             # t6 = endereço de A[i][j]

        sw      $t8, 0($t6)               # armazenar contador em A[i][j]
        addi    $t8, $t8, 1               # contador++

        addi    $t4, $t4, 1               # i++

        j       inner_loop                # repetir o loop interno

end_inner_loop:
        addi    $t3, $t3, 1               # j++
        j       outer_loop                # repetir o loop externo

end_outer_loop:
    # fim do programa
        li      $v0, 10                   # código para sair do programa
        syscall
```

Como podemos ver apenas o loop sofreu alteração, alternado as anteriores linhas para colunas e as anteriores colunas para linhas.

Parte 3:

Abaixo teremos uma tabela mostrando os Cache Miss Count da parte 1 e parte 2, com o mapeamento associativo por conjuntos.

	8 blocos 4 words	8 blocos 8 words	16 blocos 8 words	16 blocos 16 words	16 blocos 32 words
Parte 1	65	33	33	17	9
Parte 2	257	257	257	17	9

Como podemos ver na parte 1, a qual usou um loop externo para as linhas, é muito superior em caches menores do que a parte 2, que usou um loop externo para colunas. Isso acontece por conta do conceito de localidade espacial da cache, a parte 1 foi executada de modo que os dados ficassem sequenciais na memória, assim explorando essa carácteristica, diferentemente da parte 2, a qual apresenta lacunas na memória, ou seja, os dados buscados não estão sequenciais, por isso

ela apresenta um desempenho horrível em cache menores, mas estabiliza em caches maiores, pois conseguimos trazer todos os dados para a cache.