MAC 0219/5742 – Introdução à Computação Concorrente, Paralela e Distribuída

Prof. Dr. Alfredo Goldman

MiniEP2 - Otimizando a *cache* `versão 1.0`

Monitores: Giuliano Belinassi e Matheus Tavares

1 Introdução

A memória *cache* é uma memória de rápido acesso, localizada dentro dos processadores. Sua velocidade de acesso em comparação à RAM chega a ser da ordem de centenas de vezes mais rápida por: estar fisicamente muito mais próxima da CPU; e ser implementada usando SRAM que, embora encareça a produção por *byte*, permite a construção de memórias mais rápidas[1]. Existem heurísticas de detecção automática do tamanho do *cache*, conforme discutido em [2], mas também é possível detectar o cache usando os recursos do SO (veja o comando `lscpu`).

2 Problema

Considere o seguinte código (multiplicação de matrizes $n \times n$):

```
for (i = 0; i < n; ++i)
    for (j = 0; j < n; ++j)
    {
        double sum = 0;
        for (k = 0; k < n; ++k)
            sum += A(i, k)*B(k, j);
        C(i, j) = sum;
    }
```

presente em `matrix_dgemm_0`, dentro do arquivo `matrix.c`. Seu objetivo é utilizar os conhecimentos recém adquiridos a respeito da *cache* para otimizar o código acima.

2.1 Primeira otimização

Você deverá implementar na função `matrix_dgemm_1` uma versão mais rápida (e ainda correta) do código acima, apenas usando as noções de localidade de acesso à memória *cache* vistas em aula.

Dica: Lembre-se que a ordem que se itera sobre uma matriz (coluna depois linha ou linha depois coluna) pode acabar invalidando *cache*. Assim, procure uma forma de garantir que a iteração sobre `A` e `B`, no código acima, possa melhor aproveitar o *cache*.

[1] `https://people.freebsd.org/~lstewart/articles/cpumemory.pdf`

[2] `https://stackoverflow.com/questions/2576762/measure-size-and-way-order-of-l1-and-l2-caches`

2.2 Usando blocagem

É possível melhorar ainda mais o acesso ao cache usando uma técnica chamada blocagem. Aqui as matrizes são particionadas em matrizes menores, e a multiplicação é feita em etapas. Por exemplo, podemos particionar as matrizes $A = \begin{bmatrix} A_{11} & A_{12} \end{bmatrix}$ e $B = \begin{bmatrix} B_{11} & B_{21} \end{bmatrix}^\intercal$ e assim fazer o produto:

$$C = AB = \begin{bmatrix} A_{11} & A_{12} \end{bmatrix} \begin{bmatrix} B_{11} \\ B_{21} \end{bmatrix} = \begin{bmatrix} A_{11}B_{11} + A_{12}B_{21} \end{bmatrix}$$

como ilustrado na Figura 1. Por fim, é possível realizar o particionamento de diversas maneiras[3], mas aqui você deve buscar um particionamento de forma que a multiplicação melhor use o cache.

Sua tarefa é implementar em `matrix_dgemm_2` uma versão ainda mais otimizada de sua `matrix_dgemm_1`, usando essa técnica.

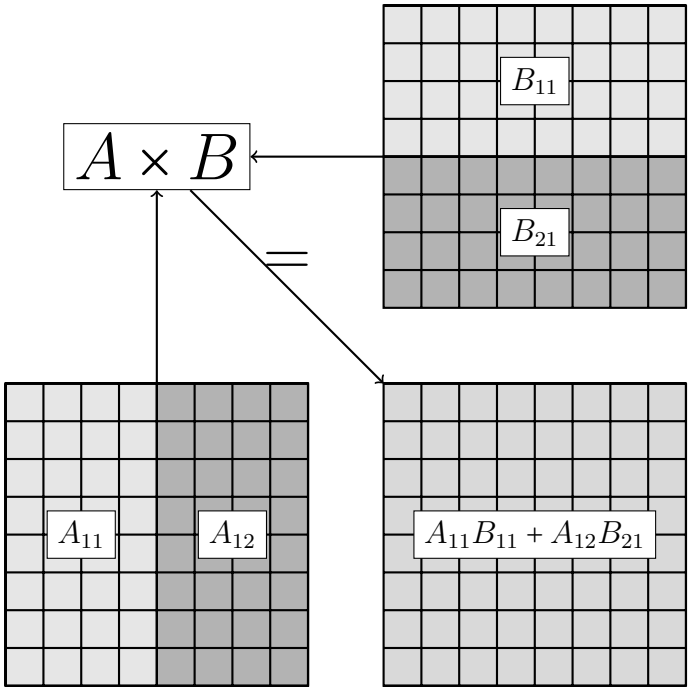

Figura 1: Ilustração de multiplicação de matrizes usando blocagem

2.3 Relatório

Você deverá elaborar um *relatório detalhado*, respondendo as seguintes questões:

1. Mostre, com embasamento estatístico, a variação de tempo entre `matrix_dgemm_1` e sua implementação de `matrix_dgemm_0`. Houve melhora no tempo de execução? Explique porque.

[3] `https://en.wikipedia.org/wiki/Block_matrix`

2. Mostre, com embasamento estatístico, a variação de tempo entre `matrix_dgemm_2` e sua implementação de `matrix_dgemm_1`. Houve melhora no tempo de execução? Explique porque.

3. Como você usou a blocagem para melhorar a velocidade da multiplicação de matrizes?

3 *Software* fornecido

Foi fornecido no PACA um programa base onde o seu Mini-EP será codificado. Você deverá modificar apenas as funções `matrix_dgemm_1` e `matrix_dgemm_2` com a sua implementação, conforme descrito acima. O programa fornecido contém testes, na qual a sua implementação deverá passar. para executá-los, basta usar o comando:

```
$ make test
```

no terminal, com seu *shell* aberto na pasta contendo o `Makefile`. Se o teste com a `matrix_dgemm_0` demorar muito, vocês podem diminuir o valor de `#define N` no `test.c` para que fique mais rápido.

Para facilitar a coleta de amostras estatísticas, também é fornecido um binário `main` que recebe como entrada os seguintes argumentos

```
main <ARGS>
onde <ARGS> pode ser a combinacao de
--matrix-size <NUM>      Tamanho da matriz quadrada. (n)
--algorithm <NUM>        Numero da implementacao. (0 = dgemm_0,
                                                   1 = dgemm_1,
                                                   2 = dgemm_2)
```

Para compilá-lo, basta rodar

```
$ make
```

no terminal, com seu *shell* aberto na pasta contendo o `Makefile`.

4 Material de Apoio e Curiosidades

A eficácia da memória *cache* em algoritmos de multiplicação de matrizes é um assunto amplamente discutido por vários autores. O livro *Fundamentals of matrix computations*, de Watkins, apresenta de maneira superficial o efeito deste sobre os algoritmos de álgebra linear, preferindo focar nos aspectos teóricos da técnica de blocagem. Já *Compilers: Principles, Techniques, and Tools* discute de maneira mais concreta e detalhada o efeito da *cache* na multiplicação de matrizes, além de explicar o funcionamento da técnica de blocagem e discutir uma paralelização desse algoritmo. Há também um artigo bem completo sobre memórias em computadores com o título de *What every programmer*

should know about memory, de Ulrich Drepper, discutindo aspectos de implementação de memórias em *Hardware*, seus efeitos e modificando uma implementação de multiplicação de matrizes para ilustrar as diferenças.

Há também estudos mostrando algoritmos mais eficientes para a multiplicação de matrizes. Strassen impressionou o mundo mostrando uma maneira de multiplicar duas matrizes de tamanho $n \times n$ em $O(n^{2.807})$, o que é um ganho sobre o algoritmo $O(n^3)$ usado convencionalmente[4]. Posteriormente, Coppersmith conseguiu reduzir este valor para $O(n^{2.375})$[5].

5 Entrega

Deverá ser entregue um pacote no sistema PACA com uma pasta com o nome e o sobrenome do estudante que o submeteu no seguinte formato: `nome_sobrenome.zip`. Essa pasta deve ser comprimida em formato `ZIP` e deve conter dois itens:

- Sua modificação de `matrix.c`
- Um relatório em `.txt` ou `.pdf` com o nome dos integrantes, e uma breve explicação sobre a sua solução, desafios encontrados, e os tópicos pedidos na seção 2.3. Imagens também podem ser inseridas no arquivo. Relatórios em `.doc`, `.docx` ou `odt` **não** serão aceitos.

Em caso de dúvidas, use o fórum de discussão do Paca. A data de entrega deste Exercício Programa é até às **16:00h do dia 2 de Maio**.

Boa Sorte!

[4] `https://pt.wikipedia.org/wiki/Algoritmo_de_Strassen`

[5] `https://pt.wikipedia.org/wiki/Algoritmo_de_Coppersmith-Winograd`