MAC 0219/5742 - Introdução à Computação Paralela e Distribuída

Mini EP 6 - Otimizando o uso de *cache* - parte 2

Alfredo Goldman e Vitor Terra

1. Introdução

No mini EP anterior, foi feita uma otimização em um programa que multiplica duas matrizes quadradas, levando em consideração a arquitetura de memória. Este exercício é uma continuação do anterior, no qual será feita ainda mais uma otimização, utilizando a técnica de blocagem (*blocking*). Tal técnica consiste em particionar uma matriz em submatrizes (blocos) e realizar o produto entre as matrizes bloco a bloco, tal como se cada bloco fosse um elemento independente. Por exemplo, podemos particionar as matrizes A e B da seguinte forma para calcular o seu produto:

$$A = [A_{11} \quad A_{12}] \text{ e } B = [B_{11} \quad B_{21}]^{\top}$$

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

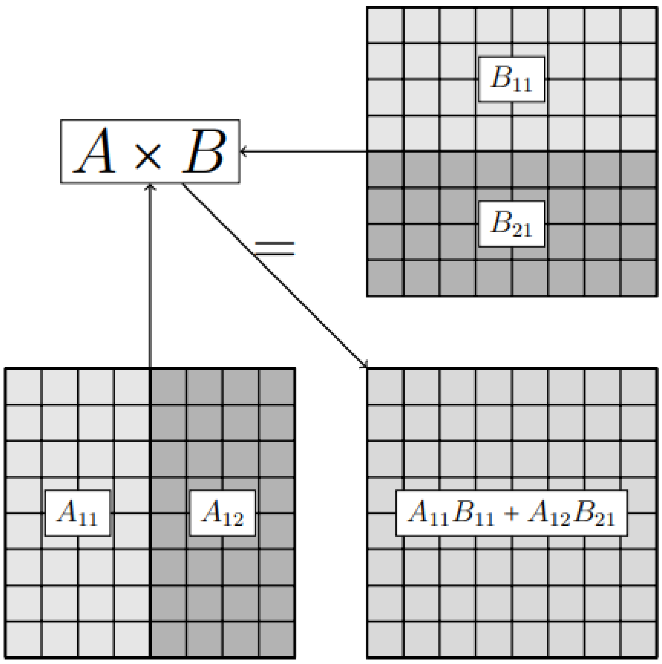

Figura 1: Ilustração de multiplicação de matrizes usando blocagem

Existem diversas formas de se realizar o particionamento[1], mas neste exercício o objetivo é fazê-lo de modo que seja feito bom uso do *cache* na multiplicação das matrizes. A tarefa deste mini EP é implementar a função `matrix_dgemm_2`: uma versão ainda mais otimizada de `matrix_dgemm_1`, usando a técnica de blocagem.

2. Código inicial

O programa base para a codificação do mini EP (arquivo `src_miniep6.zip`) é semelhante ao do exercício anterior, com algumas modificações para incluir a função `matrix_dgemm_2` (inicialmente vazia). Você deverá modificar apenas a função `matrix_dgemm_2` com a sua versão otimizada de `matrix_dgemm_1`, além de incluir a sua versão de `matrix_dgemm_1` enviada no mini EP anterior.

A seguir são reproduzidas as instruções para compilação e execução, que são as mesmas do mini EP anterior:

Para executar os testes, use 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, você pode diminuir o valor de `#define N` em `test.c` para que fique mais rápido (mantendo o N como potência de 2).

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

```
$ ./main --matrix-size <N> --algorithm <ALGO>
```

onde `<N>` é a dimensão das matrizes quadradas (N = número de linhas = número de colunas) e `<ALGO>` é o número da implementação a ser executada (0 = dgemm_0, 1 = dgemm_1, 2 = dgemm_2). Para compilá-lo, execute no terminal o comando

```
$ make
```

com seu *shell* aberto na pasta contendo o `Makefile`. O tempo de execução em segundos é mostrado na saída (`stdout`). Se desejar, você pode formatar a saída de forma conveniente para a análise dos dados, alterando a linha 118 do arquivo `main.c`. Para remover os arquivos binários gerados na compilação, use o comando

```
$ make clean
```

---

[1] [Block matrix - Wikipedia]

3. Entrega

Você deverá elaborar um relatório breve, incluindo os seguintes tópicos:

1. Explique como você usou a blocagem para melhorar a velocidade da multiplicação de matrizes. Para encontrar o tamanho adequado do bloco, meça o tempo médio de execução de `matrix_dgemm_2` variando as suas dimensões entre diferentes potências de 2.

Para facilitar a execução desses experimentos, você pode alterar o arquivo `main.c` para receber o tamanho do bloco como parâmetro. Note que também pode ser necessário fazer pequenas alterações nos arquivos `matrix.c` e `matrix.h`.

2. Mostre, com embasamento estatístico, a variação do tempo de execução entre as suas implementações de `matrix_dgemm_1` e `matrix_dgemm_2`, considerando as dimensões do bloco que levaram ao menor tempo médio de execução. Houve melhora no tempo de execução? Explique o porquê.

3. Execute os mesmos experimentos em outra máquina (por exemplo, a de um colega) e verifique se houve mudança no tamanho ideal do bloco e no tempo de execução de `matrix_dgemm_2`. Eventuais diferenças podem ocorrer devido a diferenças nas especificações das máquinas (por exemplo, tamanho das linhas de *cache*).

Entregue no e-Disciplinas uma pasta compactada com o seu nome e sobrenome no seguinte formato: `miniep6_nome_sobrenome.zip`. Essa pasta deve ser comprimida em formato .zip e deve conter dois itens:

- O arquivo `matrix.c` modificado, contendo a sua implementação da função `matrix_dgemm_2`, bem como a função `matrix_dgemm_1` do mini EP anterior;

- O relatório em .txt ou .pdf com o seu nome e uma breve explicação sobre a sua solução e desafios encontrados, bem como os tópicos pedidos anteriormente. 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 e-Disciplinas ou entre em contato diretamente com o monitor (vitortterra@ime.usp.br) ou o professor (gold@ime.usp.br).

4. Material de apoio e curiosidades

O uso adequado da memória *cache* em algoritmos de multiplicação de matrizes é um assunto amplamente discutido por vários autores. O livro F*undamentals of matrix computations*, de Watkins, apresenta de maneira superficial o efeito deste sobre os algoritmos de álgebra linear, focando nos aspectos teóricos da técnica de blocagem. Já *Compilers: Principles, Techniques, and Tools* (o "*Dragon Book*") 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 deste algoritmo.

Há também um artigo bem completo sobre memórias em computadores com o título de *What every programmer should know about memory*[2], 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.

As otimizações realizadas nos dois últimos mini EPs não alteram a complexidade assintótica do algoritmo utilizado, que é $O(n^3)$, sendo n o número de linhas e colunas das matrizes quadradas a serem multiplicadas. Em 1969, Strassen[3] apresentou um algoritmo que expressa a multiplicação de matrizes n x n em termos de 7 produtos de matrizes n/2 x n/2. Devido ao tempo $\Theta(n^2)$ utilizado para calcular as submatrizes e combinar os resultados dos subproblemas, o tempo gasto T(n) obedece à relação de recorrência $T(n) = 7T(n/2) + cn^2$, de modo que $T(n) \in O(n^{\log 7/\log 2}) \approx O(n^{2.807})$.

Desde então, foram descobertos algoritmos para multiplicação de matrizes cuja complexidade assintótica é cada vez menor. O resultado mais recente foi $O(n^{2.37188})$[4], publicado em outubro de 2022, por Du, Wuan e Zhou. No entanto, tais algoritmos são de interesse predominantemente teórico e não são utilizados na prática, pois possuem constantes multiplicativas muito altas, omitidas pelo uso da notação *Big O*.

Em outras palavras, tais algoritmos teriam menor tempo de execução apenas para matrizes densas muito maiores do que aquelas usadas na prática. No entanto, encontrar algoritmos de menor complexidade assintótica para o produto de matrizes segue sendo um tema de interesse, pois a existência de um algoritmo ótimo para multiplicação de matrizes ainda é um problema aberto em ciência da computação[5].

5. Agradecimentos

Aos monitores da disciplina em anos anteriores, Giuliano Belinassi e Matheus Tavares, pela elaboração do enunciado do mini EP no qual o presente exercício foi fortemente baseado.

[2] [What Every Programmer Should Know About Memory (freebsd.org)]
[3] [Strassen algorithm - Wikipedia]
[4] [[2210.10173] Faster Matrix Multiplication via Asymmetric Hashing (arxiv.org)]
[5] [Computational complexity of matrix multiplication - Wikipedia]