Relatório EP1 MAC0219

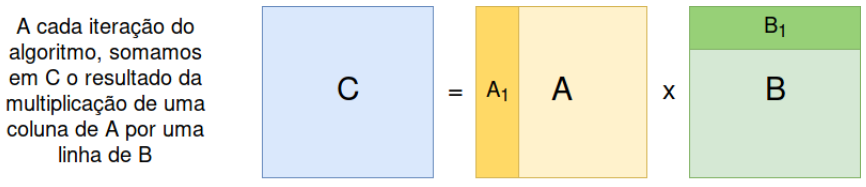
João Gabriel Basi Nº USP 9793801
Juliano Garcia de Oliveira Nº USP: 9277086

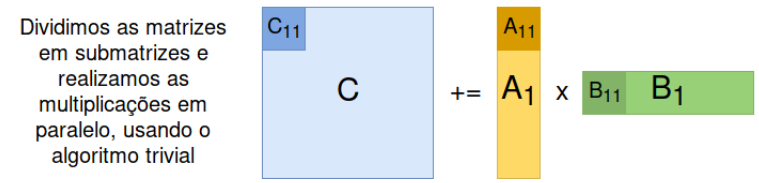
1 Pthreads

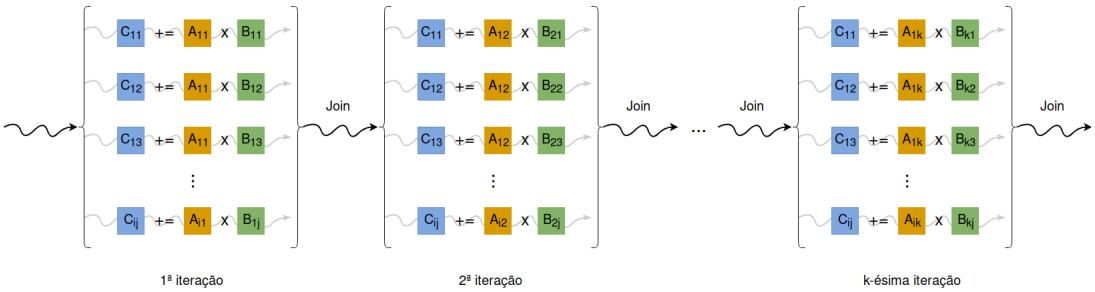
Nessa implementação o algoritmo divide a matriz A em submatrizes de tamanho NxP, a B em submatrizes de tamanho PxM e a C em submatrizes de tamanho NxM.

Os valores de N, M e P podem variar em uma unidade de submatriz para submatriz. Considerando que L é o tamanho do cache L1, os inteiros N, M e P são escolhidos de forma que as submatrizes tenham áreas parecidas, $P \leq 64$ e esteja o mais perto de 64 possível (escolhemos 64 pois foi o tamanho que teve a melhor performance), $M \cdot P \leq L$ e $N \cdot P \leq L$ e estejam o mais próximo de L possível.

Após dividir as matrizes, o algoritmo multiplica paralelamente a primeira coluna de submatrizes de A pela primeira linha de submatrizes de B, soma em C, dá join nas threads, multiplica paralelamente a segunda coluna de submatrizes de A pela segunda linha de submatrizes de B, soma em C, dá join nas threads e continua até a última coluna de A e última linha de B. Ao dar join nas threads a cada iteração, o algoritmo garante que não haja necessidade de criar zonas de exclusão mútua, já que duas threads diferentes não acessam a mesma posição de C ao mesmo tempo.







Para multiplicar as submatrizes utilizamos o algoritmo trivial de multiplicação de matriz, porém com a inversão nos loops internos para melhor aproveitamento do cache (a mesma otimização utilizada na implementação em OpenMP).

Nós tentamos usar as instruções intrinsics, mas não obtivemos sucesso. Por algum motivo, mesmo alinhando a memória, as operações davam segmentation fault, então desistimos, mas, após alguns testes, concluímos que utilizando intrinsics teriamos um ganho de 10% na velocidade.

2 OpenMP

A implementação no OpenMP é uma implementação mais simples do algoritmo de multiplicação de matrizes, se comparado com a implementação com Pthreads. No algoritmo de multiplicação implementado para o OpenMP, foi otimizada a ordem dos *loops* para aproveitamento do cache. O algoritmo padrão para multiplicação de matrizes é o seguinte (considerando que C é a matriz nula):

```
for (uint64_t i = 0; i < A.rows; i++)
        for (uint64_t j = 0; j < B.cols; j++)
            for (uint64_t k = 0; k < B.rows; k++)
                C[i][j] += A[i][k] * B[k][j];
```

Observando o algoritmo, verifica-se que o acesso aos elementos da matriz B são feitos na ordem das colunas, o que é muito ineficiente da forma como, na linguagem C, está implementado a alocação e armazenamento na memória. Uma técnica que é bastante utilizada para aproveitar a localidade do cache e diminuir esse problema é transpor a matriz B antes de fazer a multiplicação. Porém isso leva tempo $\mathcal{O}(n^2)$, e dependendo do tamanho da matriz, esse custo adicional poderia ser evitado.

Na nossa implementação, alteramos a ordem que é calculado o produto, fazendo com que o acesso seja mais rápido, e faça uso da localidade do cache, isto é, diminuindo a quantidade de *cache miss* na execução do código pois estamos percorrendo todos os valores nas "linhas", e não por colunas. Também calculamos explícitamente alguns índices antes para não ser calculado todo momento de acesso à memória (provavelmente isto deve ser otimizado pelo compilador, dependendo do nível de otimização). O código final (sem as cláusulas de OpenMP) é o seguinte:

```
for (uint64_t i = 0; i < A.rows; i++) {
             for (uint64_t k = 0; k < B.rows; k++) {
                double r = A[i*A.cols + k];
                for (uint64_t j = 0; j < B.cols; j++) {
                    C[i*C.cols + j] += r*B[k*B.cols + j];
                }
             }
        }
```

Assim como no caso da implementação com Pthreads, tentamos usar Intrinsics da Intel para melhorar o tempo, mas tivemos os mesmos problemas (erro de *segmentation fault* quando as instruções eram utilizadas). Para o OpenMP, não foi utilizado a multiplicação em blocos, tivemos alguns problemas em manter controle com os ponteiros da matriz explícitamente. Na implementação de Pthreads, para fazer as separações e multiplicações das matrizes é feito o cálculo usando apenas incrementos de ponteiros, sem calcular as posições i, j da matriz. Isso pode ser visto na função *matmul_pt_sub()* no arquivo *ptmatmul.c*.

Então, decidimos manter a versão do OpenMP que utiliza localidade do cache, ao invés de utilizar multiplicação por blocos das matrizes. Por esta razão, geralmente o tempo de execução da implementação usando Pthreads é mais rápido, nos testes que fizemos.

References

[M. Ananth, S. Vishwas and M. R. Anala] "Cache Friendly Strategies to Optimize Matrix Multiplication,"2017 IEEE 7th International Advance Computing Conference (IACC), Hyderabad, 2017, pp. 23-27.

[Kazushige Goto, Robert Van De Geijn] "Anatomy of High-Performance Matrix Multiplication", 2008