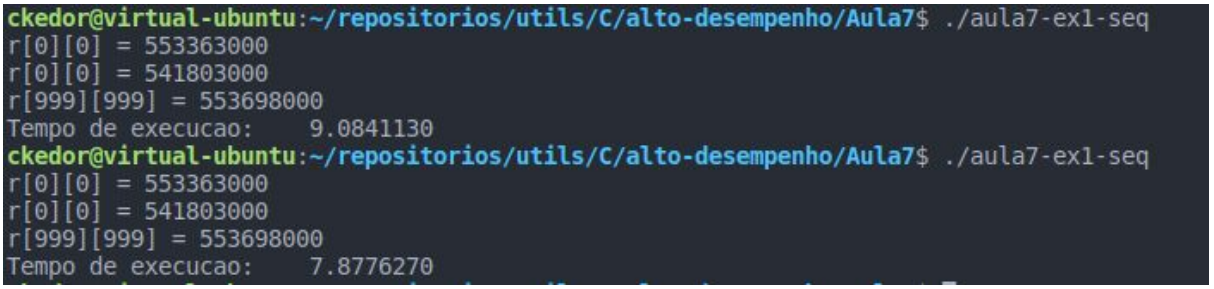
Relatório Aula 6 - Grupo 11

João Paulo Lins
Christian Braga Kedor

1. Inicialmente executou-se o exercício 1 de maneira sequencial, chegando a um tempo de execução médio de 8.5 segundos.



Para otimizar o código, criou-se duas seções paralelas utilizando os comandos <#pragma omp parallel sections> e <#pragma omp section> para criar duas seções paralelas, cada uma realizando uma das multiplicações desejadas.

Cada uma delas também foi paralelizada com o comando <#pragma omp parallel for num_threads(2) private (i,j,k)>, o que divide o processo em 2 threads distintas.

```
#pragma omp parallel sections
{
    #pragma omp section
    {
        #pragma omp parallel for num_threads(2) private(i,j,k)
        for(i = 0; i < SIZE; i++) {
            for (k = 0; k < SIZE; k++)
                for (j = 0; j < SIZE; j++)
                    ab[i][j] = ab[i][j] + a[i][k] * b[k][j];
        }

        printf("FIM secao 1 - THREAD: %d\n",omp_get_thread_num());
        fflush(stdout);
    }

    #pragma omp section
    {
        #pragma omp parallel for num_threads(2) private(i,j,k)
        for(i = 0; i < SIZE; i++) {
            for (k = 0; k < SIZE; k++)
                for (j = 0; j < SIZE; j++)
                    cd[i][j] = cd[i][j] + c[i][k] * d[k][j];
        }

        printf("FIM secao 2 - THREAD: %d\n",omp_get_thread_num());
        fflush(stdout);
    }

}
```

Por fim, a soma das matrizes resultantes foi otimizada analogamente à paralelização interna das multiplicações, como visto abaixo.

```
#pragma omp parallel for num_threads(2) private(i,j)
for(i = 0; i < SIZE; i++)
    for(j = 0; j < SIZE; j++)
        r[i][j] = ab[i][j] + cd[i][j];
```

Dessa forma, a execução do código passou a ser mais eficiente, chegando a um tempo médio de processamento em torno de 2 segundos, ou seja, 4x mais rápido.

```
ckedor@virtual-ubuntu:~/repositorios/utils/C/alto-desempenho/Aula7$ ./exercicio1
INICIO
FIM secao 2 - THREAD: 1
FIM secao 1 - THREAD: 0
FIM
r[0][0] = 553363000
r[0][0] = 541803000
r[999][999] = 553698000
Tempo de execucao:    2.0204430
ckedor@virtual-ubuntu:~/repositorios/utils/C/alto-desempenho/Aula7$ ./exercicio1
INICIO
FIM secao 1 - THREAD: 0
FIM secao 2 - THREAD: 1
FIM
r[0][0] = 553363000
r[0][0] = 541803000
r[999][999] = 553698000
Tempo de execucao:    1.7975950
```

2 - Exercício 2

2.1 - Não é possível usar a paralelização através de laços porque existem duas dependências entre as matrizes utilizadas, de forma que por esse meio não existiria um controle sobre o uso do conteúdo dessas matrizes. Por exemplo, a matriz B poderia estar pegando valores errados da matriz A pois não se sabe em que momento do laço ela está executando.

2.2 - Como a matriz B depende de A e a matriz C depende de B, o problema pode ser resolvido com o uso de dois semáforos, com a estratégia de paralelização *pipeline*.

Seção 1

```
#pragma omp section
{
    int i,j;
    A[0]=15;
    for (i=1;i<size;i++) {
        for(j=0;j<size;j++){
            A[i]=(A[i-1]+i*3+15+j)%1000;
        }
        sem_post(&sem);
    }
}
```

Seção 2

```
#pragma omp section
{
    int i,j;
    B[0] = 1;
    for(i = 1; i < size; i++){
        sem_wait(&sem);
        for(j=0;j<size;j++){
            B[i] = (B[i-1]*3 + A[i-1]+j)%1000;
        }
        sem_post(&sem2);
    }
}
```

Seção 3

```
#pragma omp section
{
    int i,j;
    C[0] = 1;
    for(i = 1; i < size; i++){
        sem_wait(&sem2);
        for(j=0;j<size;j++){
            C[i] = (C[i-1] + B[i-1] * 2+j)%1000;
        }
    }
}
```

A forma apresentada garante que os valores da matriz B serão calculados apenas após o cálculo do valor correspondente na matriz A. De forma análoga, o mesmo acontece para os valores da matriz C.

Resultados:

2a) Duas execuções da versão sequencial:

```
ckedor@virtual-ubuntu:~/repositorios/utils/C/alto-desempenho/Aula7$ ./aula7-ex2-seq
INICIO
C[0]=1  C[1]=2  C[19999]=668
FIM
Tempo de execucao:    3.6897350
ckedor@virtual-ubuntu:~/repositorios/utils/C/alto-desempenho/Aula7$ ./aula7-ex2-seq
INICIO
C[0]=1  C[1]=2  C[19999]=668
FIM
Tempo de execucao:    3.4996640
```

2b) Duas execuções da versão paralela:

```
ckedor@virtual-ubuntu:~/repositorios/utils/C/alto-desempenho/Aula7$ ./exercicio2
C[0]=1  C[1]=2  C[19999]=668
FIM
Tempo de execucao:    1.3515100
ckedor@virtual-ubuntu:~/repositorios/utils/C/alto-desempenho/Aula7$ ./exercicio2
C[0]=1  C[1]=2  C[19999]=668
FIM
Tempo de execucao:    1.3093410
```

Nota-se que a execução paralela trás valores corretos e o tempo cai para mais da metade.