Project Assignment I: All-Pairs Shortest Path Problem

David Ferreira up202006302

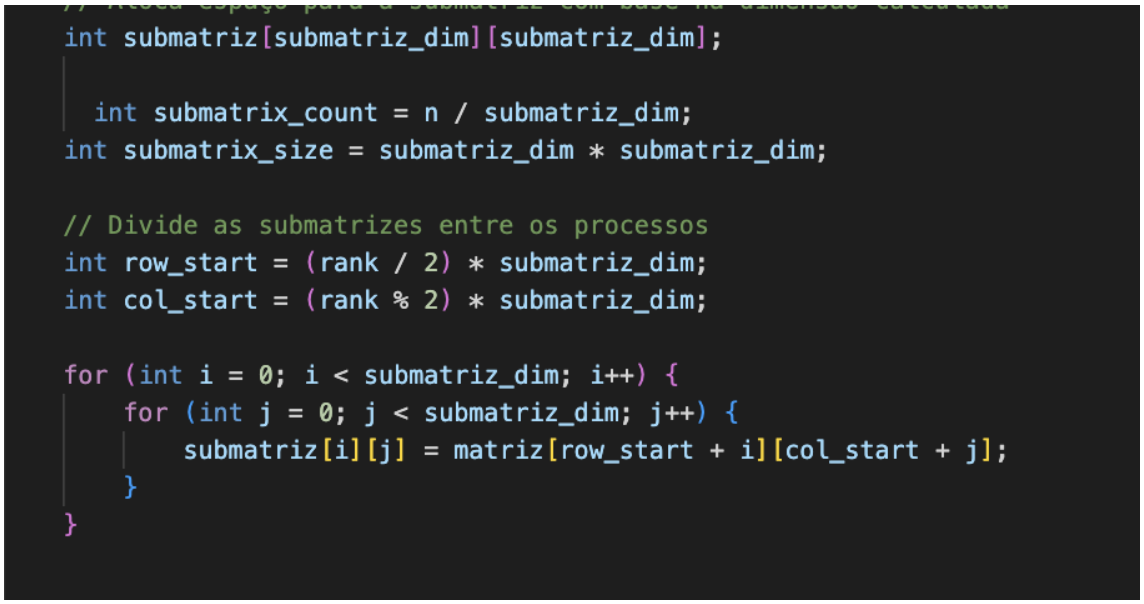
Este projeto foi feito no âmbito da cadeira de Computação Paralela da faculdade de ciências (FCUP). O objetivo deste trabalho passa por adquirir experiência com a computação paralela para distribuição memoria usando a Framework MPI.

No meu projeto tenho dois ficheiros c (divisão.c,main .c).O primeiro ficheiro divisão.c faz a divisão da matriz dada como input em submatrizes.Cada processo criado tem uma submatriz atribuida.O ficheiro main.c faz o calculo do caminho mais curto do grafo de um nó a outro. A matriz input contém o número de vértices do grafo e o custo das arestas.

No problema que foi proposto foi nos dado uma ideia de implementação para resolvê-lo, para isso, usávamos dois algoritmos. Um deles é o algoritmo fox em que divide a matriz em submatrizes por diferentes processos e faz a multiplicação entre elas. Este primeiro era para usar em complemento do segundo algoritmo que era o produto da distância que fazia o mínimo da soma das linhas e das colunas com o valor da posição em que estávamos a calcular.

Divisão.c

No ficheiro C divisão começamos por calcular o tamanho que as submatrizes vão ter e se é possível fazer essa mesma divisão das matrizes. O número de processos utilizados tem de ser igual a q*q. O valor de q terá de ser um divisor do tamanho da matriz.



```
int submatriz[submatriz_dim][submatriz_dim];

  int submatrix_count = n / submatriz_dim;
int submatrix_size = submatriz_dim * submatriz_dim;

// Divide as submatrizes entre os processos
int row_start = (rank / 2) * submatriz_dim;
int col_start = (rank % 2) * submatriz_dim;

for (int i = 0; i < submatriz_dim; i++) {
    for (int j = 0; j < submatriz_dim; j++) {
        submatriz[i][j] = matriz[row_start + i][col_start + j];
    }
}
```

Main.c

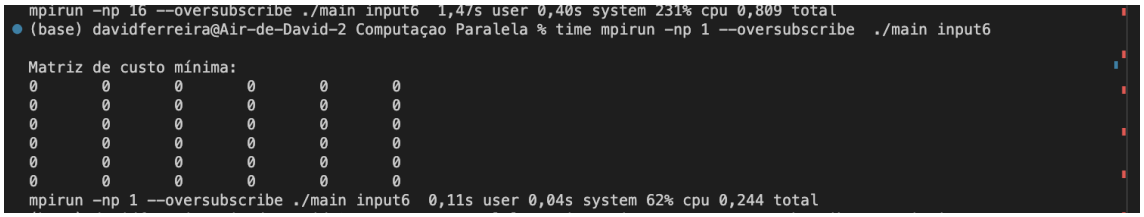
O main.c deveria ser utilizado o algoritmo fox para calcular o custo mínimo do grafo de um no a outro, mas não o consegui a utilizar devido a muitas dificuldades em entender como iria conseguir usar a linha e colunas que das submatrizes que pretendíamos para calcular certo valor. A primeira abordagem que tentei foi ter dois arrays row e column.O arrray column ia ter os valores da soma de todas as colunas e o array row ia ter a soma de todas as linhas e quando tivéssemos esses valores calculados usamos os comandos MPI_Recv e MPI_Send para passar essa informação entre processos. Para calcular por exemplo na matriz exemplo 6*6 fazíamos a divisão em submatrizes de 3*3 em 4 processos e para calcular o valor da posição [0][0] da matriz do custo mínimo fazíamos a soma da linha 0 do

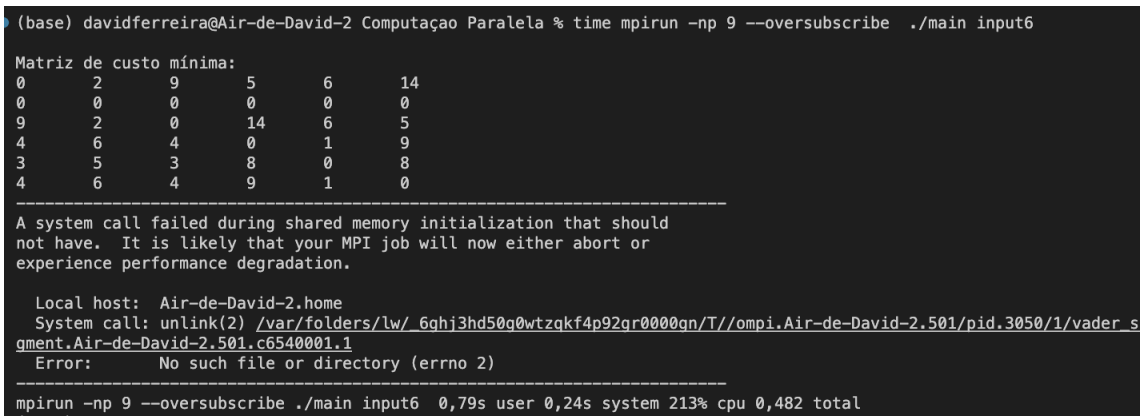
processo 0 com a soma da coluna 0 do processo 0 mais soma da linha 0 do processo 1 que tínhamos acesso graças ao nosso array row mais a soma da coluna 0 do processo 2 que tínhamos acesso graças ao array column se esta soma fosse menor ao valor atual da submatriz do processo ,o valor da submatriz passava a ser o valor da soma da submatriz do processo 0 senão deixava estar o valor original .Esta abordagem não deu certo pois o resultado correto nunca foi atingido.

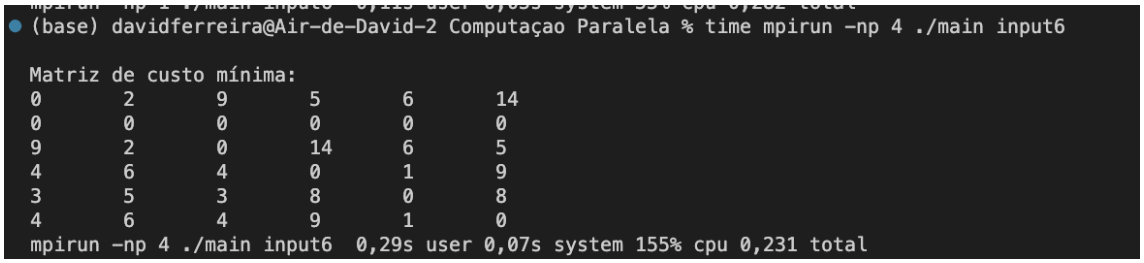
Devido às dificuldades que passei para atingir o resultado correto tive de abordar o problema de uma forma diferente.
Usei o algoritmo floyd warshall para realizar o problema proposto sabendo que este algoritmo é pouco eficiente com grafo muito grandes.Este algoritmo realiza uma série de iterações. Em cada iteração, ele considera um vértice intermediário como candidato a fazer parte do caminho entre dois outros vértices. Os vértices intermediários são considerados um por um.Para cada par de vértices (i, j) e para cada vértice intermediário k, o algoritmo verifica se o caminho de i para j passando por k é mais curto do que o caminho direto de i para j. Se for, o valor na matriz de distância é atualizado para refletir o caminho mais curto.

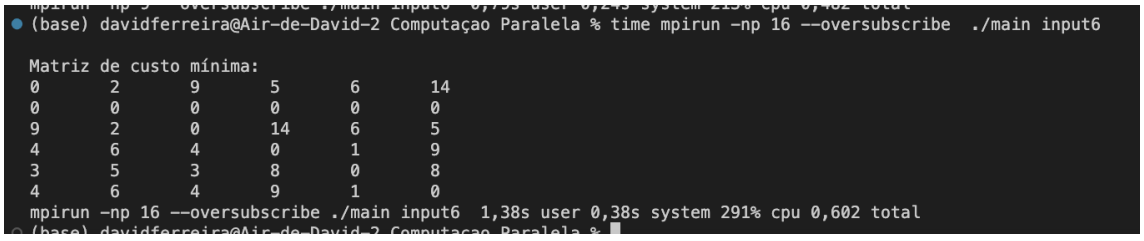
```
if(rank!=0) {

    MPI_Recv(&submatrix[0][0], n * n, MPI_INT, 0, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);

}


for (int k = 0; k < n; k++) {
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            if (submatrix[i][k] + submatrix[k][j] < submatrix[i][j]) {
                submatrix[i][j] = submatrix[i][k] + submatrix[k][j];
            }
        }
    }
}



if (rank != 0) {
    MPI_Send(&submatrix[0][0], n * n, MPI_INT, 0, 0, MPI_COMM_WORLD);


} else {

    for (int source = 1; source < size; source++) {
        MPI_Recv(&submatrix[0][0], n * n, MPI_INT, source, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);


    }


    printf("Matriz de custo minima:\n");
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            if (submatrix[i][j] == INF) {
                printf("0\t");
            } else {
                printf("%d\t", submatrix[i][j]);
            }
        }
        printf("\n");
    }
}
```

Performance Evaluation









Dificuldades:

Penso que a minha maior dificuldade foi entender como iria usar Min-plus matrix multiplication, conhecido como “ Distance Product”com o

algoritmo fox nunca conseguir perceber a partir da divisão de matrizes fazia este calculo usando as funcionalidades da framework MPI