Architecture et calcul parallèle

TP N° 3

M2 Bio-informatiques

BOUZIDI Abdeldjalil 201504000052

ZERROUKHI Djillali 201400745000

Solution de l’exercice N°7 :

**Usage des types dérivés de l’équation de la chaleur**

**(Laplacien)**

Pour compiler et exécuter :

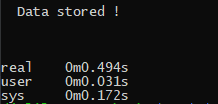


1. **Version séquentielle :**

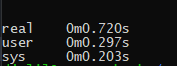
**Code :**

|  |
| --- |
| for (int epoch = 0; epoch < 10000; epoch++) {        printf("\n>>>>>>>>>>>>>  EPOCH :: %d <<<<<<<<<<<<<\n", epoch);         for (i = 1; i < length-1; i++) {            for (j = 1; j < length-1; j++) {                MatRes[i][j] = (int)((MatRes[i][j-1] + MatRes[i][j+1] + MatRes[i-1][j] + MatRes[i+1][j]) / 4);            }        } } |

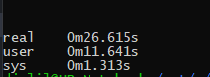
* Résultat de compilation & exécution du code (1000 itérations / 10\*10) :



* Résultat de compilation & exécution du code (1000 itérations / 100\*100) :



* Résultat de compilation & exécution du code (1000 itérations / 1000\*1000) :



1. **Version parallèle :**
   1. Dans le cas ou le nombre de processus est inférieur à la taille de la matrice, on ne peut pas utiliser ‘Scatter’ puisque chaque processus doit calculer un certain nombre de lignes et puisque ‘Scatter’ ne permet pas le chevauchement des lignes et donc ce n’est pas possible.
   2. L’indice de début de calcule ainsi que le nombre de lignes chaque processus sont calculés par le processus ‘Master’ (Rank = 0).

**Code :**

|  |
| --- |
| for (int epoch = 0*; epoch < 50; epoch++) {*        if (rank == 0) {            printf("\n>>>>>>>>>>>>>  EPOCH :: %d <<<<<<<<<<<<<\n", epoch)*;*        }        MPI\_Scatter(MatRes, length, MPI\_DOUBLE, old\_row, length, MPI\_DOUBLE, 0, MPI\_COMM\_WORLD)*;*         MPI\_Send(old\_row, length, MPI\_DOUBLE, next, 1, MPI\_COMM\_WORLD)*;*        MPI\_Send(old\_row, length, MPI\_DOUBLE, prev, 1, MPI\_COMM\_WORLD)*;*         MPI\_Recv(old\_row\_prev, length, MPI\_DOUBLE, prev, 1, MPI\_COMM\_WORLD, &status)*;*        MPI\_Recv(old\_row\_next, length, MPI\_DOUBLE, next, 1, MPI\_COMM\_WORLD, &status)*;*         for (i = 1*; i < length-1; i++) {*                new\_row[i] = (old\_row[i-1] + old\_row[i+1] + old\_row\_prev[i] + old\_row\_next[i]) / 4*;*        }        new\_row[0] = old\_row[0]*;*        new\_row[length-1] = old\_row[length-1]*;*      MPI\_Gather(new\_row, length, MPI\_DOUBLE, MatRes, length, MPI\_DOUBLE, 0, MPI\_COMM\_WORLD)*;*     //printf (" My Rank : %d , I've send One Row \n ", rank)*;*     MPI\_Barrier(MPI\_COMM\_WORLD)*;* } |

* Résultat de compilation & exécution du code (100 itérations / 10\*10) :



* Résultat de compilation & exécution du code (1000 itérations / 100\*100) :



* Résultat de compilation & exécution du code (1000 itérations / 500\*500) :



1. **Version parallèle V2 (NB\_proc < NB\_lignes) :**

* Résultat de compilation & exécution du code (1000 itérations / 10\*10 / 4 process) :



* Résultat de compilation & exécution du code (1000 itérations / 20\*20 / 4 process) :



**Code (structure indexed\_row) :**

|  |
| --- |
| typedef struct {            int ind*;*            double row[length]*;*     }indexed\_row*;*      MPI\_Datatype index*;*     int blocs[2] = {1, length}*;*     MPI\_Datatype types[2]*;*     MPI\_Aint moves[2];     moves[0] = offsetof(indexed\_row, ind)*;*     moves[1] = offsetof(indexed\_row, row)*;*     types[0] = MPI\_INT*;*     types[1] = MPI\_DOUBLE*;*     MPI\_Type\_struct(2, blocs, moves, types, &index)*;*     MPI\_Type\_commit(&index)*;* |

**Code (Send / Receive) :**

|  |
| --- |
| for (int epoch = 0; epoch < 50; epoch++) {        if (rank == 0) {            printf("\n>>>>>>>>>>>>>  EPOCH :: %d <<<<<<<<<<<<<\n", epoch);            i = 0;            k = 0;            while (i < numtasks) {                val = chunk[i];                indexed\_row data[val];                for (j = 0; j < chunk[i]; j++) {                    data[j].ind = k;                    for (int l = 0; l < length; l++) {                            data[j].row[l] = MatRes[k][l];                    }                    k++;                }                MPI\_Bsend(&data, chunk[i], index, i, 1, MPI\_COMM\_WORLD);                i++;            }            i = 0;            while (i < numtasks){                val = chunk[i];                indexed\_row data[val];                MPI\_Recv(&data, chunk[i], index, i, 1, MPI\_COMM\_WORLD, &status);                for (j = 0; j < chunk[i]; j++) {                    k = data[j].ind;                    for (int l = 0; l < length; l++) {                            MatRes[k][l] = data[j].row[l];                    }                }                i++;            }         }        else {        indexed\_row receved[chunk[rank]], sended[chunk[rank]];        MPI\_Recv(&receved, chunk[rank], index, 0, 1, MPI\_COMM\_WORLD, &status);        for (j = 0; j < chunk[rank]; j++) {            k = receved[j].ind;            sended[j].ind = receved[j].ind;            if(k !=0 && k != numtasks-1){                for (int l = 1; l < length-1; l++) {                    new\_row[i] = (receved[j].row[i-1] + receved[j].row[i+1] + MatRes[k-1][i] + MatRes[k+1][i]) / 4;                    sended[j].row[l] = new\_row[i];                }            }            else {                for (int l = 0; l < length; l++) {                    sended[j].row[l] = receved[j].row[l];                }            }        }        MPI\_Bsend(&sended, chunk[rank], index, 0, 1, MPI\_COMM\_WORLD); } |