```
/* ParallÃ©lisation avec MPI */
/* Damien CrÃ©milleux - Lauriane Holy */

#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#include <mpi.h>

#define N 13
#define M 15
#define MAX 100
#define SEUIL 0.1
#define NB_ITER 100

int main(int argc, char * argv[])
{
  int i, j;
  int rank, size;

  MPI_Status stat;
  MPI_Init(&argc, &argv);
  MPI_Comm_rank(MPI_COMM_WORLD, &rank);
  MPI_Comm_size(MPI_COMM_WORLD, &size);

  /* MPI_Datatype type_Colonne; */

  if(size<=1) {
    printf("Ce programme ne fonctionne qu\'avec au moins 2 processus\n");
    exit(-1);
  }

  if((N+2)%size){
    printf("Ce programme ne fonctionne qu\'avec un nombre de  processus divisant N\n");
    exit(-1);
  }

  int borneN;
  borneN =(N+2/size);
  if(rank == 0)
    printf("borneN : %d\n", borneN);

  double delta, deltaT;
  delta = 0;

  int nb_iter;
  nb_iter = 0;

  /* Premier processus */
  if(rank == 0) {

    /* initialisation de la matrice globale */
    double TG[N+2][M+2];
    for(i = 0 ; i<borneN; i++)
      { for(j=0; j<M+2; j++)
          {
            if(i==0 || j ==0 || j == M+1)
              TG[i][j]= MAX;
            else
              TG[i][j]= 0;
          }
      }

    /* initialisation de la partie spÃ©cifique */
    double T[borneN+1][M+2], T1[borneN+1][M+2];
    for(i = 0 ; i<borneN; i++)
      {
        for(j=0; j<M+2; j++)
          {
            if(i==0 || j ==0 || j == M+1)
              T[i][j]= MAX;
            else
              T[i][j]= 0;
          }
      }

    /* On affiche la matrice initiale */
    for(i = 1; i < size-1; i++) {
      MPI_Recv(&TG[borneN*i], (M+2)*borneN, MPI_DOUBLE, i+1, MPI_ANY_TAG, MPI_CO
MM_WORLD, &stat);
    }
    MPI_Recv(&TG[borneN*(size-1)], (M+2)*borneN, MPI_DOUBLE, i+1, MPI_ANY_TAG, M
PI_COMM_WORLD, &stat);


    for(i = 0 ; i<N+2; i++)
      {
        for(j = 0; j<M+2; j++)
          {
            printf("%6.2f ", TG[i][j]);
          }
        printf("\n");
      }

    /* Propagation */
    do {
      MPI_Send(&T1[borneN-1],M+2, MPI_DOUBLE, rank+1, 0, MPI_COMM_WORLD);

      for(i = 1 ; i<borneN; i++)
        {
          for(j = 1; j<M+1; j++)
            {
              T1[i][j] = (double) (T[i][j+1] + T[i][j-1] + T[i+1][j] + T[i-1][j]
 + T[i][j]) / 5;
              delta = delta +  fabs(T1[i][j]-T[i][j]);
            }
        }

      /* recopie du tableau */
      for(i = 0 ; i<borneN; i++)
        {
          for(j=0; j<M+2; j++)
            {
              if(i==0 || i== N+1 || j ==0 || j == M+1)
                T[i][j]= MAX;
              else
                T[i][j] = T1[i][j];
            }
        }

      MPI_Recv(&T[borneN], M+2, MPI_DOUBLE, rank+1, MPI_ANY_TAG, MPI_COMM_WORLD,
 &stat);

      nb_iter++;
      /* On vÃ©rifie la valeur de delta toutes les NB_ITER itÃ©rations */
      if(nb_iter%NB_ITER == 0) {
        MPI_Allreduce(&delta, &deltaT, 1, MPI_DOUBLE, MPI_SUM, MPI_COMM_WORLD);
        printf("deltaT : %d\n", deltaT);
      }
    } while(deltaT >= SEUIL);
  }

  /* Processus intermÃ©diaire */
  if((rank!= 0) && (rank != size-1)){

    /* initialisation */
    double T[borneN+1][M+2], T1[borneN+1][M+2];
    for(i = 0 ; i<borneN; i++)
      {
```

```
        for(j=0; j<M+2; j++)
          {
            if(j ==0 || j == M+1)
              T[i][j]= MAX;
            else
              T[i][j]= 0;
          }
      }

    MPI_Send(&T[0], borneN*(M+2), MPI_DOUBLE, 0, 0, MPI_COMM_WORLD);

    do{
      /* on envoie aux voisins les valeurs */
      MPI_Send(&T1[borneN-1],M+2, MPI_DOUBLE, rank+1, 0, MPI_COMM_WORLD);
      MPI_Send(&T1[1],M+2, MPI_DOUBLE, rank-1, 0, MPI_COMM_WORLD);
      for(i = 1 ; i<borneN; i++)
        {
          for(j=1; j<M+1; j++)
            {
              T1[i][j] = (double) (T[i][j+1] + T[i][j-1] + T[i+1][j] + T[i-1][j]
 + T[i][j]) / 5;
              delta = delta +  fabs(T1[i][j]-T[i][j]);
            }
        }

      /* recopie du tableau */
      for(i = 0 ; i<borneN; i++)
        {
          for(j=0; j<M+2; j++)
            {
              if(i==0 || i== N+1 || j ==0 || j == M+1)
                T[i][j]= MAX;
              else
                T[i][j] = T1[i][j];
            }
        }

      MPI_Recv(&T[borneN], M+2, MPI_DOUBLE, rank+1, MPI_ANY_TAG, MPI_COMM_WORLD,
 &stat);
      MPI_Recv(&T[0], M+2, MPI_DOUBLE, rank-1, MPI_ANY_TAG, MPI_COMM_WORLD, &sta
t);

      nb_iter++;
      /* On vÃ©rifie la valeur de delta toutes les NB_ITER itÃ©rations */
      if(nb_iter%NB_ITER == 0) {
        MPI_Allreduce(&delta, &deltaT, 1, MPI_DOUBLE, MPI_SUM, MPI_COMM_WORLD);
      }
    } while(deltaT >= SEUIL);


  }

  /* Dernier processus */
  if(rank == size-1) {

    /* initialisation */
    double T[borneN+1][M+2], T1[borneN+1][M+2];
    for(i = 0 ; i<borneN+1; i++)
      {
        for(j=0; j<M+2; j++)
          {
            if(i== borneN || j ==0 || j == M+1)
              T[i][j]= MAX;
            else
              T[i][j]= 0;
          }
      }

    MPI_Send(&T[0], borneN*(M+2), MPI_DOUBLE, 0, 0, MPI_COMM_WORLD);
```



```
    do{
      MPI_Send(&T1[1],M+2, MPI_DOUBLE, rank-1, 0, MPI_COMM_WORLD);
      for(i = 1 ; i<borneN; i++)
        {
          for(j=1; j<M+1; j++)
            {
              T1[i][j] = (double) (T[i][j+1] + T[i][j-1] + T[i+1][j] + T[i-1][j]
 + T[i][j]) / 5;
              delta = delta +  fabs(T1[i][j]-T[i][j]);
            }
        }

      /* recopie du tableau */
      for(i = 0 ; i<borneN; i++)
        {
          for(j=0; j<M+2; j++)
            {
              if(i==0 || i== N+1 || j ==0 || j == M+1)
                T[i][j]= MAX;
              else
                T[i][j] = T1[i][j];
            }
        }

      MPI_Recv(&T[0], M+2, MPI_DOUBLE, rank-1, MPI_ANY_TAG, MPI_COMM_WORLD, &sta
t);

      nb_iter++;
      /* On vÃ©rifie la valeur de delta toutes les NB_ITER itÃ©rations */
      if(nb_iter%NB_ITER == 0) {
        MPI_Allreduce(&delta, &deltaT, 1, MPI_DOUBLE, MPI_SUM, MPI_COMM_WORLD);
      }
    } while(deltaT >= SEUIL);
  }

  MPI_Finalize();
  return 0;
}
```

```c
#include <stdio.h>
#include <mpi.h>
int main(int argc, char *argv[])
  {
    int rank, size;

    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);

    MPI_Comm_size(MPI_COMM_WORLD,&size);

    printf("Hello wold from node %d of %d nodes\n", rank, size);
    MPI_Finalize();
    printf("Fini\n");
    return(0);
  }
```

```c
#include <stdlib.h>
#include <stdio.h>
#include <mpi.h>

int main (int argc, char *argv[])
{



  int i;      /* numero du processus courant */
  int p;      /* nombre total de processus */

  int recu; /* valeur recue par le processus courant */
  MPI_Status stat; /* pour l'initialisation */

  MPI_Init(&argc, &argv);
  MPI_Comm_rank(MPI_COMM_WORLD, &i); // Rang du processus
  MPI_Comm_size(MPI_COMM_WORLD, &p); // Nombre de processus

  if (p<=1) {
    printf("ce programme ne fonctionne qu\"avec au moins 2 processus\n");
    exit(-1);
  }

  printf("je suis le pss %d qui demarre parmi %d processus \n",i,p);
   if(i==0) {
     int n = 2;/* la valeur envoyée par le 1er processus */
     printf("je suis le pss 0 j'envoie au pss  %d la valeur  %d\n",i+1,n);
    MPI_Send(&n, 1, MPI_INT, i+1, 0, MPI_COMM_WORLD);
    printf("je suis le pss 0 j'attend du processus %d\n",p-1);
    MPI_Recv(&recu, 1,MPI_INT, p-1, MPI_ANY_TAG, MPI_COMM_WORLD, &stat);
    printf("je suis le pss 0 je recoit de %d la valeur %d\n",i+1,recu);
    }

   else if (i<p-1) {
     printf("je suis le pss %d j'attend du processus %d\n",i,i-1);

    MPI_Recv(&recu, 1,MPI_INT, i-1, MPI_ANY_TAG, MPI_COMM_WORLD, &stat);
    recu = recu *2;
    MPI_Send(&recu, 1, MPI_INT, i+1, 0, MPI_COMM_WORLD);
    }
    else
     {
       printf("je suis le pss %d j'attend du processus %d\n",i,i-1);
    MPI_Recv(&recu, 1,MPI_INT, i-1, MPI_ANY_TAG, MPI_COMM_WORLD, &stat);
    recu = recu *2;
    MPI_Send(&recu, 1, MPI_INT, 0, 0, MPI_COMM_WORLD);
    }


  MPI_Finalize();
  return (0);
}
```