Un exemple de bibliothèque de passage de messages : MPI

- MPI_Init (&argc, &argv)
- MPI_Comm_size(MPI_COMM_WORLD, &p) ; /* nombre de processus */
- MPI_Send(message, liste-param..., &status)
- MPI_Recv(message, liste-param..., &status)
- MPI_Finalize()

Modes de communication, synchrone, asynchrone bufferisé, synchrone

http://www.daugerresearch.com/vault/parallelknock.shtml

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

int main(int argc, char *argv[ ]) {
        int myrank, num_procs, err ;
err = MPI_Init (&argc, &argv) ;
MPI_Comm_rank(MPI_COMM_WORLD, &myrank);
MPI_Comm_size(MPI_COMM_WORLD,&num_procs) ;
if (myrank == 0 {
        printf ("  proc %d de %d : Hello World du maitre \n" , myrank,
num_procs) ;
        } else   {
         printf (" proc %d de %d : Hello World de l 'esclave \n ", myrank,
num_procs) ;
                }
```

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include "mat.h"
#include <mpi.h>
#include "chrono.h"

#define MAITRE if (myid==0)
#define ESCLAVES if (myid!=0)

void initialisation(double *T,int n)
{
  double nn = n*n;
  int i;
  for(i=0 ; i<nn ; i++)
    T[i] = (double) rand() / (double) RAND_MAX;
}

double trace(double* T,int n)
{
  int i;
  double sommeTermesDiagonaux = 0;
  for(i=0 ; i<n ; i++)
    sommeTermesDiagonaux += T[i*(n+1)];
  return sommeTermesDiagonaux;
}

int main(int argc,char *argv[])
{
  clock_t dateDebut,dateFin;
  int n,i;
  double *A;
  double *B;
  double *C;
  double *DEC;
  int namelen;
  int nb_procs,myid;
  char processor_name[MPI_MAX_PROCESSOR_NAME];
  MPI_Status *stat;


  MPI_Init(&argc,&argv);

  MPI_Comm_size(MPI_COMM_WORLD, &nb_procs);
  MPI_Comm_rank(MPI_COMM_WORLD, &myid);
  MPI_Get_processor_name(processor_name, &namelen);
  n = atoi(argv[1]);
```

```
MAITRE
    {
      A = (double*) calloc(n*n,sizeof(double));
      B = (double*) calloc(n*n,sizeof(double));
      C = (double*) calloc(n*n,sizeof(double));

      printf("\n\n\t-------------- Multiplication de matrice %dx%d sur %d
noeuds --------------\n",
           n,
           n,
           nb_procs);

      initialisation(A,n);
      initialisation(B,n);

      dateDebut = clock();
      chrono("Global");
      chrono("Distribution B");
      MPI_Bcast(B,n*n,MPI_DOUBLE,0,MPI_COMM_WORLD);
      chrono("Distribution B");

      chrono("Envois des parties de A");
      for (i=1; i<nb_procs; i++)
      {
        DEC=A+i*(n*n/nb_procs);
        MPI_Send(DEC,
               n*n/nb_procs,
               MPI_DOUBLE
               ,i
               ,0
               ,MPI_COMM_WORLD);
      }
   chrono("Envois des parties de A");

   chrono("Calcul d'une partie");
   produitMatrice(A,B,C,n/nb_procs,n);
   chrono("Calcul d'une partie");

   chrono("Reception des resultats");
      for (i=1; i<nb_procs; i++)
      MPI_Recv(C+i*(n*n/nb_procs),
             n*n/nb_procs,
             MPI_DOUBLE,
             i,
             0,
             MPI_COMM_WORLD
             ,stat);
   chrono("Reception des resultats");

      chrono("Global");
      dateFin = clock();

      /*    printf("Execution en %5.2f secondes - Trace(C) = %.4f\n",
           (double) (dateFin-dateDebut)/(double) CLOCKS_PER_SEC,trace(C,n));
      */
      chrono("BILAN");
    }
```

```
  ESCLAVES
    {
      A = (double*) calloc(n*n/nb_procs,sizeof(double));
      B = (double*) calloc(n*n,sizeof(double));
      C = (double*) calloc(n*n/nb_procs,sizeof(double));

      MPI_Bcast(B,n*n,MPI_DOUBLE,0,MPI_COMM_WORLD);

      MPI_Recv(A,n*n/nb_procs,MPI_DOUBLE,0,0,MPI_COMM_WORLD,stat);

      produitMatrice(A,B,C,n/nb_procs,n);

      MPI_Send(C,n*n/nb_procs,MPI_DOUBLE,0,0,MPI_COMM_WORLD);
    }

  free(A);
  free(B);
  free(C);

  MPI_Finalize();

  return EXIT_SUCCESS;
}
```