Solution labo # 7 en MPI

1 hello.c

```
int main( int argc, char *argv[] ) {
  int numProc;    // Numero d'idenfification du processus.
  int nbProcs;    // Nombre total de processus.

  MPI_Init( &argc, &argv );
  MPI_Comm_rank( MPI_COMM_WORLD, &numProc );
  MPI_Comm_size( MPI_COMM_WORLD, &nbProcs );

  if ( numProc != 0 ) {
    MPI_Recv( NULL, 0, MPI_BYTE, MPI_ANY_SOURCE, 0,
                                 MPI_COMM_WORLD, MPI_STATUS_IGNORE );
  }

  system( "uname --nodename" );
  printf( "\tnumProc = %2d, nbProcs = %2d\n\n", numProc, nbProcs ); fflush( stdo

  if ( numProc < nbProcs-1 ) {
    MPI_Send( NULL, 0, MPI_BYTE, numProc+1, 0, MPI_COMM_WORLD );
  }

  MPI_Finalize();
  if ( numProc == 0 ) {
    printf( "---------------------------------------\n" );
  }

  return( 0 );
}
```

2 somme-vecteurs.c

```
void sommeVecteurs( int A[], int B[], int C[], int n )
{
  int numProc;          // Numero du processeur.
  int nbProcs;          // Nombre de processeurs.

  MPI_Comm_rank( MPI_COMM_WORLD, &numProc );
  MPI_Comm_size( MPI_COMM_WORLD, &nbProcs );

  // On alloue de l'espace pour les copies (tranches) locales.
  int *monA, *monB, *monC;
  int monN = n / nbProcs;
  monA = (int *) malloc( monN * sizeof(int) );
  monB = (int *) malloc( monN * sizeof(int) );
  monC = (int *) malloc( monN * sizeof(int) );

  // On distribue les valeurs aux processeurs.
  MPI_Scatter( A, monN, MPI_INT, monA, monN, MPI_INT, 0, MPI_COMM_WORLD );
  MPI_Scatter( B, monN, MPI_INT, monB, monN, MPI_INT, 0, MPI_COMM_WORLD );

  // Chaque processeur fait la somme de sa tranche.
  for ( int i = 0; i < monN; i++ ) {
    monC[i] = monA[i] + monB[i];
  }

  // On combine les differentes tranches sur le processus maitre.
  MPI_Gather( monC, monN, MPI_INT, C, monN, MPI_INT, 0, MPI_COMM_WORLD );
}
```

3 min-max.c

```
void minMaxCentralise( int maVal, int minMax[] )
{
  int numProc, nbProcs;
  MPI_Comm_rank( MPI_COMM_WORLD, &numProc );
  MPI_Comm_size( MPI_COMM_WORLD, &nbProcs );

  minMax[0] = minMax[1] = maVal;
  if ( numProc == 0 ) {
    // Le processus maitre recoit les valeurs et les traite.
    for( int k = 1; k < nbProcs; k++ ) {
      int minMaxTmp[2];
      MPI_Recv( minMaxTmp, 2, MPI_INT, MPI_ANY_SOURCE, 0,
                            MPI_COMM_WORLD, MPI_STATUS_IGNORE );
      minMax[0] = MIN( minMax[0], minMaxTmp[0] );
      minMax[1] = MAX( minMax[1], minMaxTmp[1] );
    }
  } else {
    // Les processus autres que le maitre envoient leur valeur au maitre.
    MPI_Send( minMax, 2, MPI_INT, 0, 0, MPI_COMM_WORLD );
  }
  // Le maitre retourne aux autres processus
  MPI_Bcast( minMax, 2, MPI_INT, 0, MPI_COMM_WORLD );
}
```

```
void minMaxAllreduce( int val, int minMax[] )
{
  MPI_Allreduce( &val, &minMax[0], 1, MPI_INT, MPI_MIN, MPI_COMM_WORLD );
  MPI_Allreduce( &val, &minMax[1], 1, MPI_INT, MPI_MAX, MPI_COMM_WORLD );
}
```

```
void minMaxAnneau( int maVal, int minMax[] )
{
  // On definit les voisins de l'anneau virtuel.
  int numProc, nbProcs;
  MPI_Comm_rank( MPI_COMM_WORLD, &numProc );
  MPI_Comm_size( MPI_COMM_WORLD, &nbProcs );
  int gauche = (numProc + nbProcs - 1) % nbProcs;
  int droite = (numProc + 1) % nbProcs;

  // Valeur locale du min et max.
  minMax[0] = minMax[1] = maVal;

  // On effectue les communications pour trouver le min et le max.
  if ( numProc == 0 ) {
    // Premiere passe.
    MPI_Send( minMax, 2, MPI_INT, droite, 0, MPI_COMM_WORLD );
    MPI_Recv( minMax, 2, MPI_INT, gauche, 0, MPI_
                      COMM_WORLD, MPI_STATUS_IGNORE );

    // Deuxieme passe.
    MPI_Send( minMax, 2, MPI_INT, droite, 1, MPI_COMM_WORLD );
  } else {
    // Premiere passe.
    int minMaxTmp[2];
    MPI_Recv( minMaxTmp, 2, MPI_INT, gauche, 0,
                         MPI_COMM_WORLD, MPI_STATUS_IGNORE );

    minMax[0] = MIN( minMax[0], minMaxTmp[0] );
    minMax[1] = MAX( minMax[1], minMaxTmp[1] );
    MPI_Send( minMax, 2, MPI_INT, droite, 0, MPI_COMM_WORLD );

    // Deuxieme passe.
    MPI_Recv( minMax, 2, MPI_INT, gauche, 1,
                      MPI_COMM_WORLD, MPI_STATUS_IGNORE );
    if (numProc < nbProcs-1) {
      MPI_Send( minMax, 2, MPI_INT, droite, 1, MPI_COMM_WORLD );
    }
  }
}
```