Mésocentre de calcul
Université de Franche-Comté

Kamel MAZOUZI

8 décembre 2009

Présentation Utilisateurs

UNIVERSITÉ DE FRANCHE-COMTÉ

Plan

- Introduction au parallélisme
- OpenMP/Mémoire partagée
  - Exemples
- MPI/Mémoire distribuée
  - Exemples

Introduction

- Challenge
  - Problèmes fondamentales
  - Applications scientifiques
- Besoins importants
  - En puissance de calcul
  - En mémoire

Calcul haute performance = algorithmes efficaces + parallélisme

Parallélisme ?

- Programme séquentiel
  - un seul flot d'exécution
  - une seule instruction exécutée à la fois
  - un processeur
- Programme parallèle
  - Plusieurs flots d'exécution
  - Plusieurs instructions exécutées simultanées
  - Plusieurs processeurs

Faire coopérer plusieurs processeurs pour réaliser un calcul

Applications parallèles

- Plusieurs instances d'une application séquentielle
- Parallélisme à mémoire partagée
- Parallélisme à mémoire distribuée

Lancement de plusieurs tâches de calculs

- Besoin :
  - Exécuter une seule application avec des paramètres différents

- Solution :
  - ✗ lancer plusieurs scripts shell (SGE)
  - ✔ utiliser les tableaux de tâches
    - ➔ un seul job SGE à contrôler

Exemple

```
#!/bin/bash
#$ -t 1-10
echo hello from $SGE_TASK_ID
```

```
#!/bin/bash
#$ -t 1-10000

INPUT  = input.$SGE_TASK_ID
OUTPUT = output.$SGE_TASK_ID
./program $INPUT $OUTPUT
```

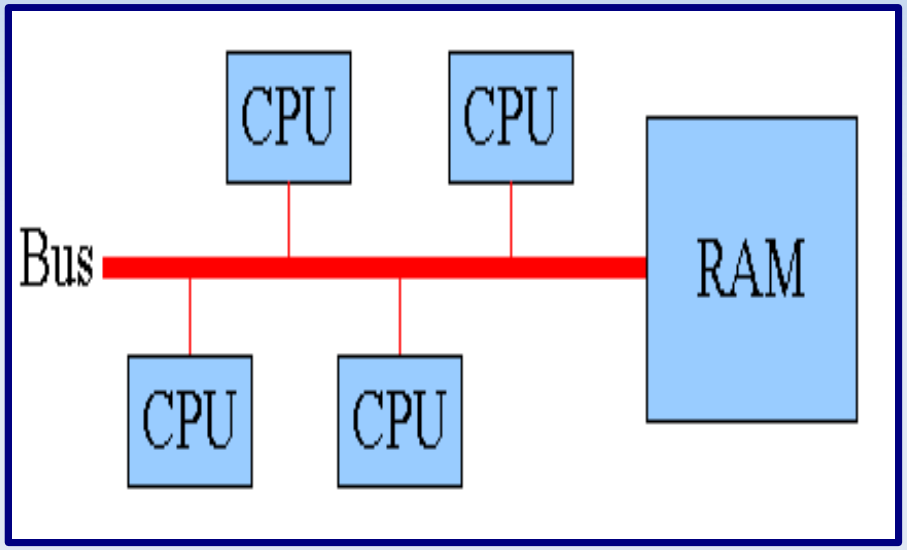
Architecture à mémoire partagée

Avantages

- pas de distribution de données
- échange de données implicite et rapide via la mémoire

Inconvénients

- bus mémoire à large bande passante
- gestion des accès concurents
- peu-scalable



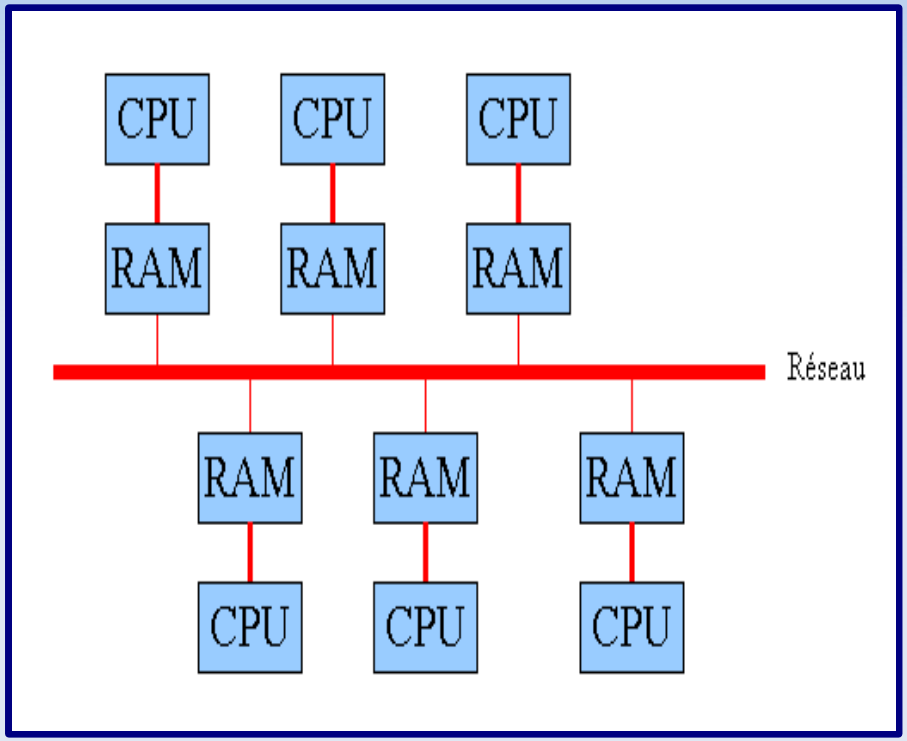
Architecture à mémoire distribuée

Avantages :

- pas d'accès concurents à la mémoire
- bande passante réseau moins critique
- extensiblité (données, calcul, ...)

Inconvénients :

- Implique une distribution de données
- Utilisation de messages explicites entre les unités

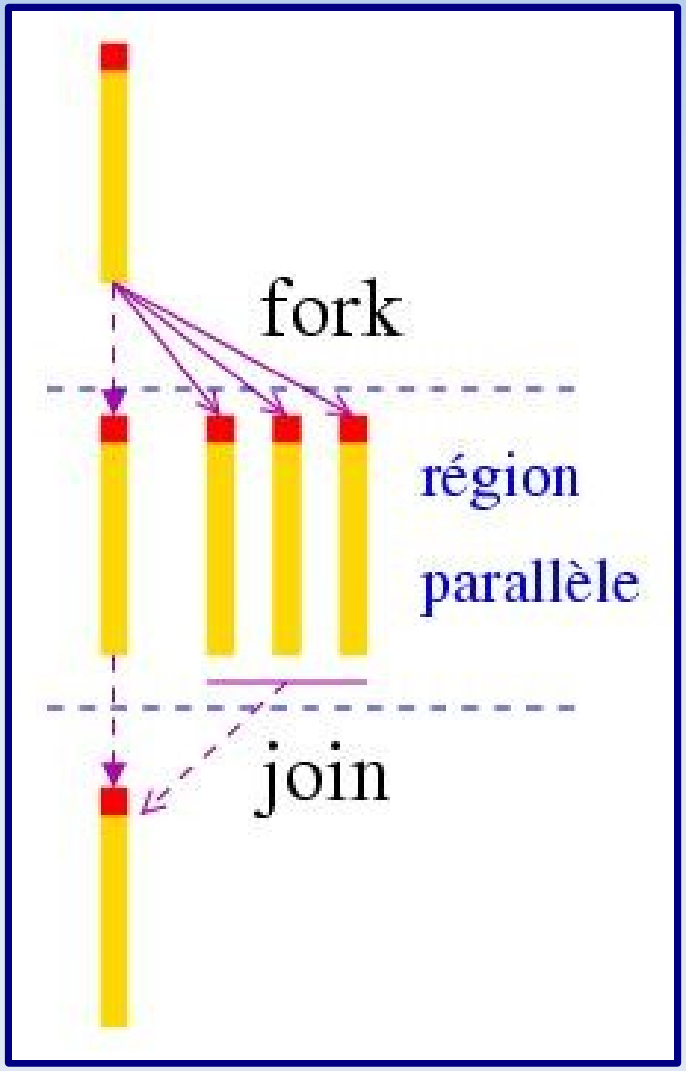


Parallélisme à mémoire partagée (OpenMP)

Un ensemble de directives de compilation pour

Paralléliser un programme

- parallélisation de haut niveau
- simplicité d'utilisation



Principales directives

- Parallélisation : `parallel`
  - Début / fin
  - Plusieurs zones
- Variables :
  - `shared` / `private`
- Ordonnancement : `schedule`
  - `static` / `dynamic` / `guided` / `runtime`
- Synchronisation :
  - `master`, `single`, `section`, `critical`

Programme C / OpenMP

```
#include <omp.h>
#include <stdio.h>
#include <stdlib.h>

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

/* Fork a team of threads giving them their own copies of variables */
#pragma omp parallel private(nthreads, tid)
  {
        /* Obtain thread number */
        tid = omp_get_thread_num();
        printf("Hello World from thread = %d\n", tid);

        /* Only master thread does this */
        if (tid == 0)     {
            nthreads = omp_get_num_threads();
            printf("Number of threads = %d\n", nthreads);
        }
  }  /* All threads join master thread and disband */
}
```

Programme Fortran / OpenMP

```
      PROGRAM HELLO

      INTEGER NTHREADS, TID, OMP_GET_NUM_THREADS,
     +        OMP_GET_THREAD_NUM

C     Fork a team of threads giving them their own copies of variables
!$OMP PARALLEL PRIVATE(NTHREADS, TID)


C     Obtain thread number
      TID = OMP_GET_THREAD_NUM()
      PRINT *, 'Hello World from thread = ', TID

C     Only master thread does this
      IF (TID .EQ. 0) THEN
        NTHREADS = OMP_GET_NUM_THREADS()
        PRINT *, 'Number of threads = ', NTHREADS
      END IF

C     All threads join master thread and disband
!$OMP END PARALLEL

      END
```

Exécution OpenMP

- Compilation

```
mesocomte0$  icc -openmp omp_hello.c -o omp_hello
omp_hello.c(22): (col. 1) remark: OpenMP DEFINED REGION WAS PARALLELIZED.
mesocomte0$
```

- Script

```
#!/bin/bash
#$ -l h_rt=360:00:0
#$ -o $JOB_NAME.o$JOB_ID
#$ -e $JOB_NAME.e$JOB_ID
#$ -N job_openmp
#$ -q normal15d
OMPRUN=./omprun
DATADIR=.
BINARY=omp_hello
export OMP_NUM_THREADS=5
$OMPRUN $DATADIR/$BINARY
exit 0
```

- Exécution

```
Hello World from thread = 4
Hello World from thread = 1
Hello World from thread = 3
Hello World from thread = 0
Number of threads = 5
Hello World from thread = 2
```

Matrices C / OpenMP

```
#define NRA 62     /* number of rows in matrix A */
#define NCA 15     /* number of columns in matrix A */
#define NCB 7      /* number of columns in matrix B */

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

 int    tid, nthreads, i, j, k, chunk;
 double      a[NRA][NCA],  /* matrix A */
        b[NCA][NCB],          /* matrix B */
        c[NRA][NCB];          /* result matrix C */

 chunk = 10;              /* loop iteration chunk size */

#pragma omp parallel shared(a,b,c,nthreads,chunk)
   private(tid,i,j,k)  {
  tid = omp_get_thread_num();
  if (tid == 0)   {
    nthreads = omp_get_num_threads();
    printf(" multiply  with %d threads\n",nthreads);
  }

  /*** Initialize matrices ***/
  #pragma omp for schedule (static, chunk)
  for (i=0; i<NRA; i++)  for (j=0; j<NCA; j++) a[i][j]= i+j;

  #pragma omp for schedule (static, chunk)
  for (i=0; i<NCA; i++) for (j=0; j<NCB; j++) b[i][j]= i*j;
```

```
  /* Do matrix multiply sharing iterations on outer loop */
  printf("Thread %d starting matrix multiply...\n",tid);
  #pragma omp for schedule (static, chunk)
  for (i=0; i<NRA; i++)
    {
    printf("Thread=%d did row=%d\n",tid,i);
    for(j=0; j<NCB; j++)
      for (k=0; k<NCA; k++)
        c[i][j] += a[i][k] * b[k][j];
    }
  }   /*** End of parallel region ***/

/*** Print results ***/
printf("******************************************\n");
printf("Result Matrix:\n");
for (i=0; i<NRA; i++)
  {
  for (j=0; j<NCB; j++)
    printf("%6.2f   ", c[i][j]);
  printf("\n");
  }
printf("******************************************\n");
printf ("Done.\n");

}
```

Matrices OpenMP

- Compilation

```
mesocomte0$ icc -openmp omp_mm.c
omp_mm.c(38): (col. 3) remark: OpenMP DEFINED LOOP WAS PARALLELIZED.
omp_mm.c(42): (col. 3) remark: OpenMP DEFINED LOOP WAS PARALLELIZED.
omp_mm.c(46): (col. 3) remark: OpenMP DEFINED LOOP WAS PARALLELIZED.
omp_mm.c(54): (col. 3) remark: OpenMP DEFINED LOOP WAS PARALLELIZED.
omp_mm.c(28): (col. 1) remark: OpenMP DEFINED REGION WAS PARALLELIZED.
omp_mm.c(58): (col. 5) remark: PERMUTED LOOP WAS VECTORIZED
mesocomte0$
```

- Script

```
#!/bin/bash
#$ -l h_rt=360:00:0
#$ -o $JOB_NAME.o$JOB_ID
#$ -e $JOB_NAME.e$JOB_ID
#$ -N job_openmp
#$ -q normal15d
OMPRUN=./omprun
DATADIR=.
BINARY=omp_mm
export OMP_NUM_THREADS=5
$OMPRUN $DATADIR/$BINARY
exit 0
```

- Exécution

```
Starting matrix multiple example with 5 threads
Initializing matrices...
Thread 0 starting matrix multiply...
Thread 2 starting matrix multiply...
Thread=2 did row=20
Thread=2 did row=21
Thread=2 did row=22
Thread=2 did row=23
Thread=2 did row=24
Thread=2 did row=25
Thread 1 starting matrix multiply...
Thread 3 starting matrix multiply...
Thread=1 did row=10
Thread=3 did row=30
Thread 5 starting matrix multiply...
...
```

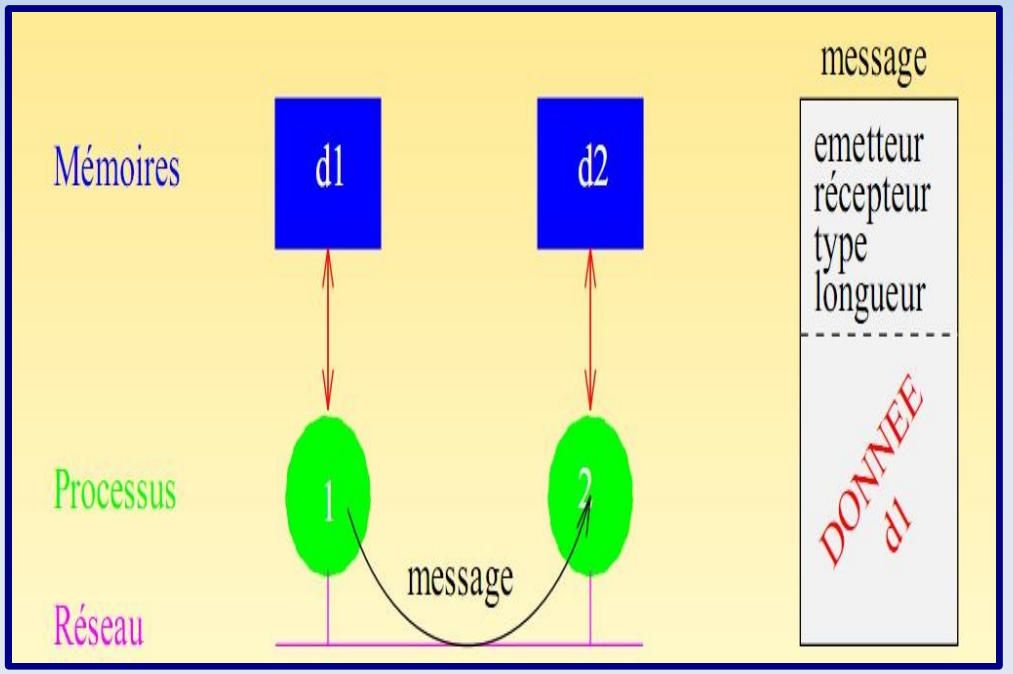
Matrices Fortran / OpenMP

```
      PROGRAM MATMULT
       INTEGER  NRA, NCA, NCB, TID, NTHREADS, I, J, K,
      +  CHUNK, OMP_GET_NUM_THREADS, OMP_GET_THREAD_NUM
C      number of rows in matrix A
       PARAMETER (NRA=62)
C      number of columns in matrix A
       PARAMETER (NCA=15)
C      number of columns in matrix B
       PARAMETER (NCB=7)
       REAL*8 A(NRA,NCA), B(NCA,NCB), C(NRA,NCB)
C      Set loop iteration chunk size
       CHUNK = 10
C      Parallel region explicitly scoping all variables
!$OMP PARALLEL SHARED(A,B,C,NTHREADS,CHUNK)
PRIVATE(TID,I,J,K)
       TID = OMP_GET_THREAD_NUM()
       IF (TID .EQ. 0) THEN
         NTHREADS = OMP_GET_NUM_THREADS()
         PRINT *, 'Matrix multiple example with', NTHREADS,
      +           'threads'
         PRINT *, 'Initializing matrices'
       END IF
C      Initialize matrices
!$OMP DO SCHEDULE(STATIC, CHUNK)
       DO 30 I=1, NRA
         DO 30 J=1, NCA
           A(I,J) = (I-1)+(J-1)
  30   CONTINUE
!$OMP DO SCHEDULE(STATIC, CHUNK)
       DO 40 I=1, NCA
         DO 40 J=1, NCB
           B(I,J) = (I-1)*(J-1)
  40   CONTINUE
!$OMP DO SCHEDULE(STATIC, CHUNK)
       DO 50 I=1, NRA
         DO 50 J=1, NCB
           C(I,J) = 0
  50   CONTINUE
```

```
C      Do matrix multiply sharing iterations on outer loop
       PRINT *, 'Thread', TID, 'starting matrix multiply...'
!$OMP DO SCHEDULE(STATIC, CHUNK)
       DO 60 I=1, NRA
       PRINT *, 'Thread', TID, 'did row', I
         DO 60 J=1, NCB
           DO 60 K=1, NCA
             C(I,J) = C(I,J) + A(I,K) * B(K,J)
  60   CONTINUE
C      End of parallel region
!$OMP END PARALLEL
C      Print results
       PRINT *, '******************************************'
       PRINT *, 'Result Matrix:'
       DO 90 I=1, NRA
         DO 80 J=1, NCB
           WRITE(*,70) C(I,J)
  70       FORMAT(2x,f8.2,$)
  80       CONTINUE
           PRINT *, ' '
  90       CONTINUE
       PRINT *, '******************************************'
       PRINT *, 'Done.'
       END
```

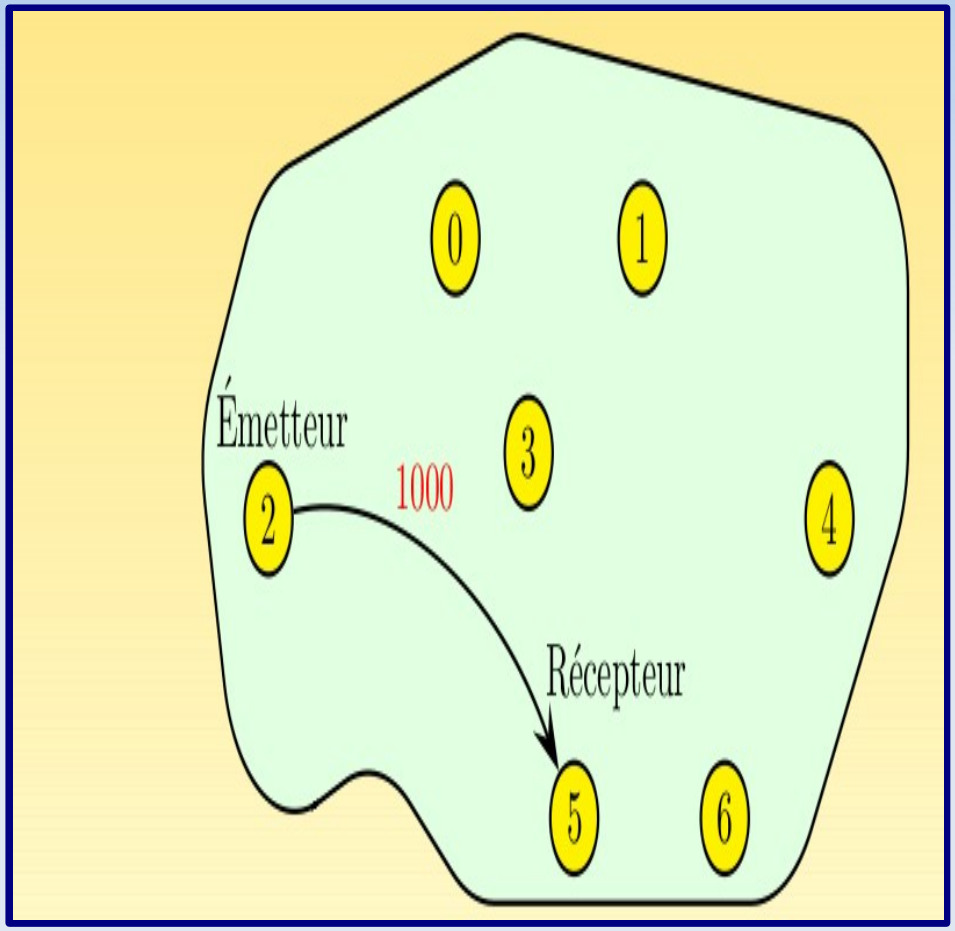
Parallélisme par échanges de messages (MPI)

- distribution de données
- passage de messages explicite entre les proceseurs (point à point, collectives)
- parallélisation à gros grain
- MPICH, LAM, OMPI



MPI : communication point à point

- Opération entre deux processeurs (un émeteur et un récepteur)
- Les données sont encapsulés dans une enveloppe :
  - numéro d'émetteur
  - numéro de récepteur
  - étiquette (TAG)
  - communicateur



Exemple MPI : Send / Receive

```
int main(int argc, char **argv) {
  int rank, size;
  double x[10];
  MPI_Status status;
  MPI_Init(&argc, &argv);
  MPI_Comm_size(MPI_COMM_WORLD, &size);
  MPI_Comm_rank(MPI_COMM_WORLD, &rank);
  if (rank == 0) {
    for (int i=0; i<10 ; i++) x[i] = 0.1*i;
       MPI_Send(x, 10, MPI_DOUBLE, 1, 666,MPI_COMM_WORLD);
  }else if (rank == 1) {
       MPI_Recv(x, 10, MPI_DOUBLE, 0, 666,  MPI_COMM_WORLD, &status);
   }
  MPI_Finalize();
}
```

MPI : communications collectives

Faire dans une seule opération une série de communications point à point.

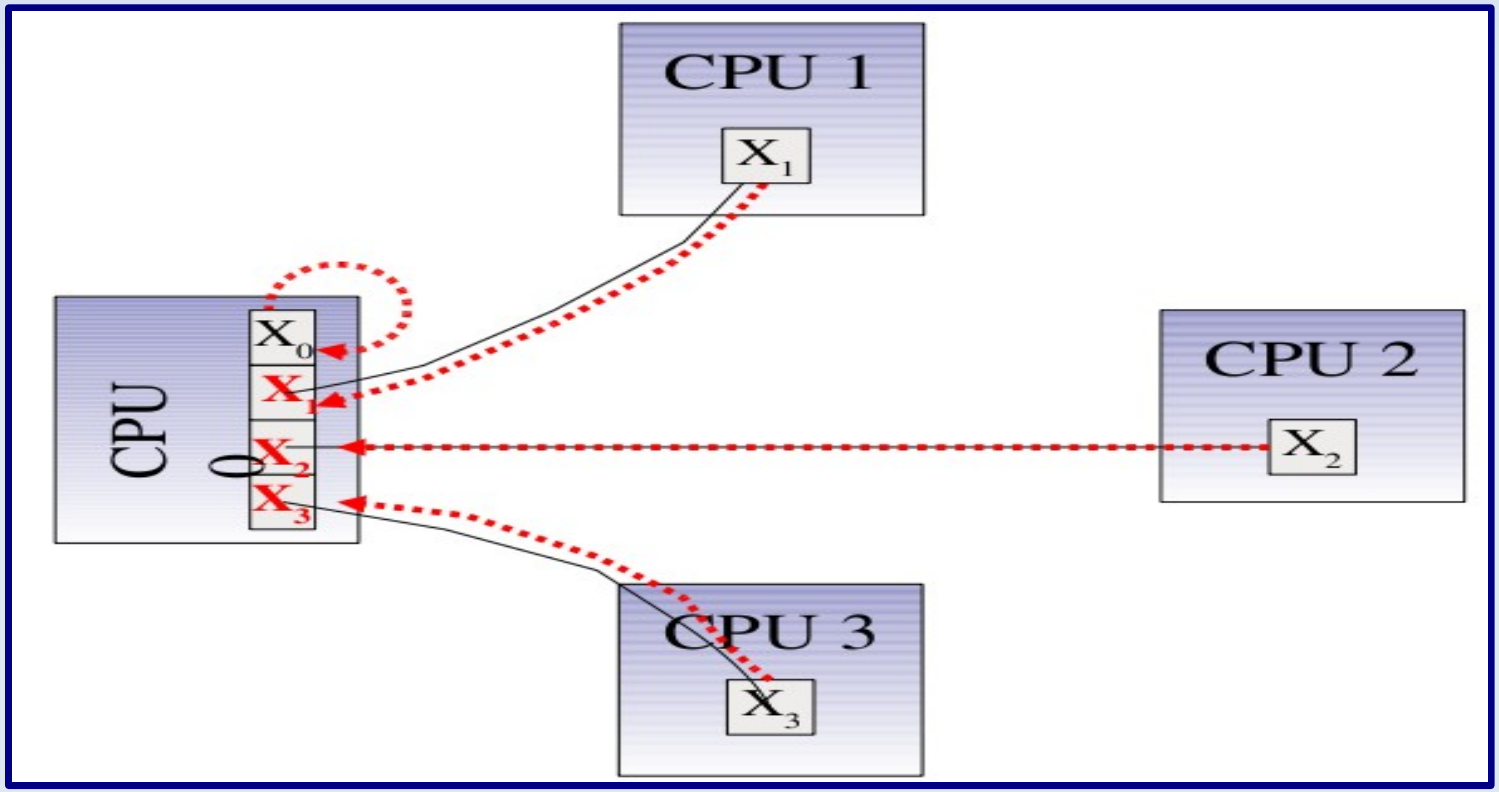
- La gestion des “tags” est transparente
- une communication collective concerne toujours tous les processus du communicateur indiqué

- Exemple :
  - bcast, gather, scatter, alltoall, ...

MPI : communication collective

Recevoir une quantité de données différentes de chaque processus.

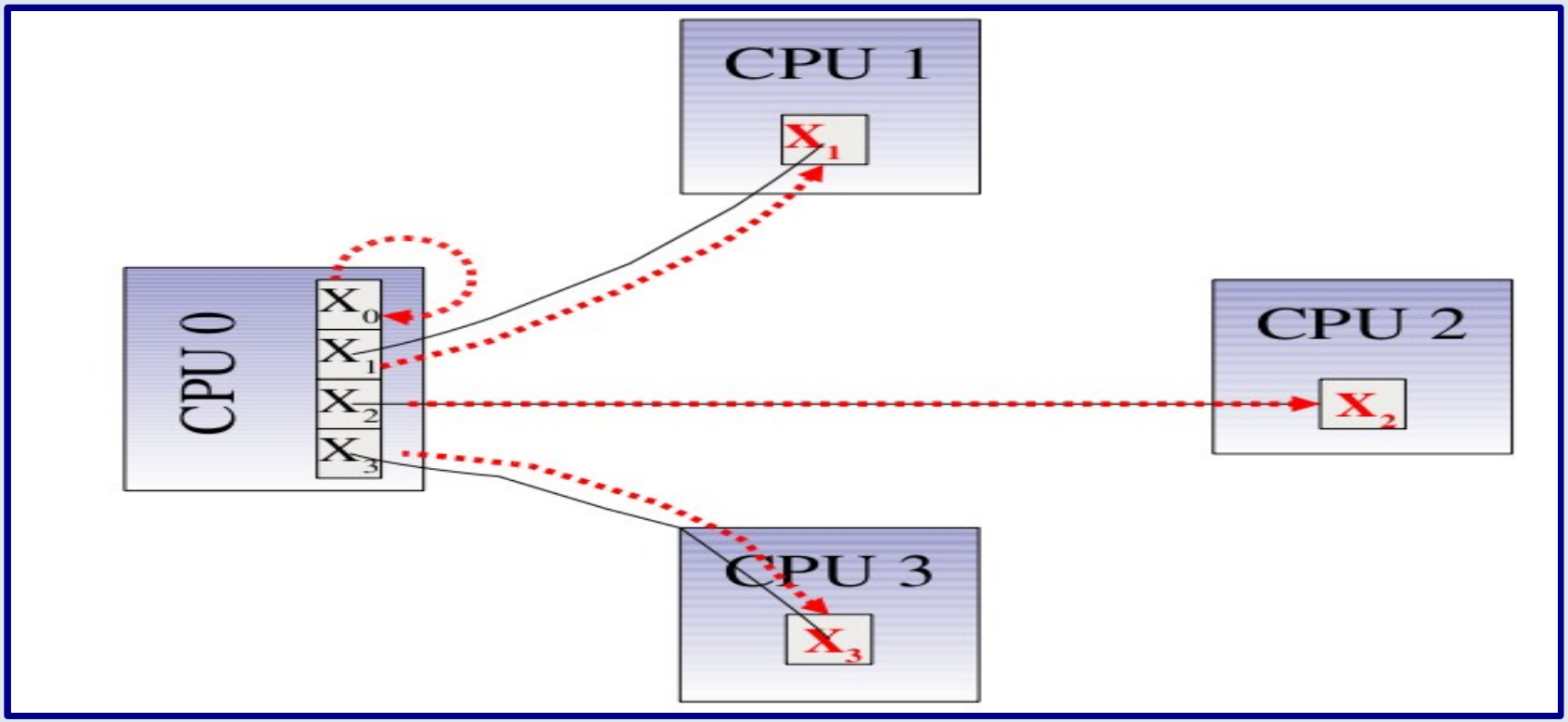
```
MPI_GATHER(s_add , s_count , s_type , r_add ,
r_count , r_type , root , comm , err);
```



MPI : communication collective

Répartir sur les processus du communicateur des données présentes sur l'un des processus.

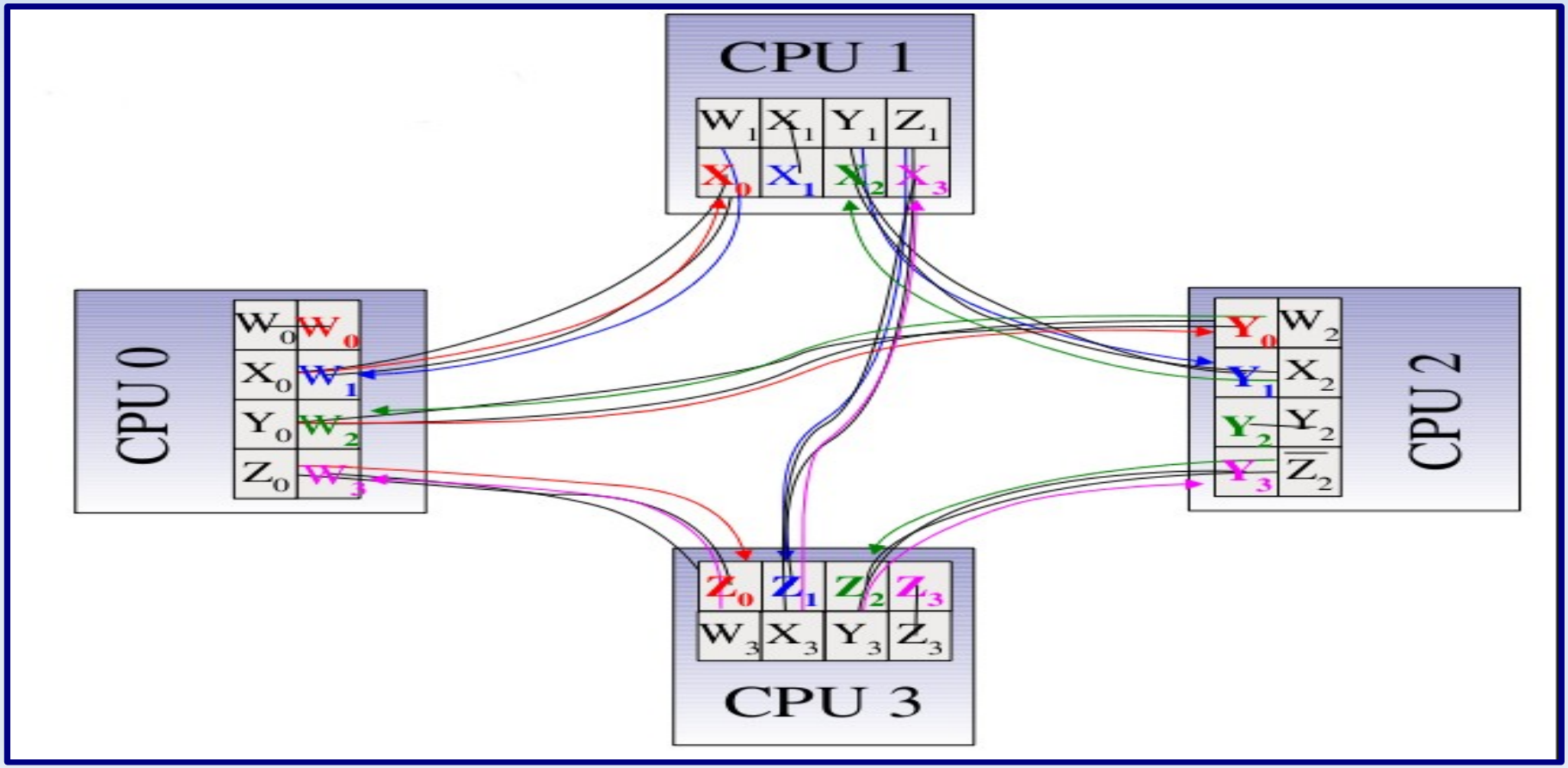
```
MPI_SCATTER(s_add , s_count , s_type , r_add ,
r_count , r_type , root , comm , err);
```



MPI : communications collectives

Chaque processeur envoi et reçoit les données des autres processeurs.

- MPI_ALLTOALL(s_add , s_count , s_type , r_add , r_count , r_type , comm , err);



Compiler un code MPI

- IMPI

```
mpiicc hello.c -o hello
mpiifort hello.f -o hello
```

- OMPI

```
mpicc hello.c -o hello
Mpif90 hello.c -o hello
```

```c
#include <stdio.h>
#include <mpi.h>
int main(int argc, char *argv[])
{
  int rank,size;
  MPI_Init(&argc, &argv);
  MPI_Comm_size(MPI_COMM_WORLD, &size);
  MPI_Comm_rank(MPI_COMM_WORLD, &rank);
  printf("Hello, world.  I am %d of %d.\n", rank,size);
  MPI_Finalize();

  return 0;
}
```

Script SGE (MPI)

```
#!/bin/bash

#$ -l h_rt=360:00:0

#$ -o $JOB_NAME.o$JOB_ID

#$ -N job_exemple_15d_par

#$ -pe impi_tight 10

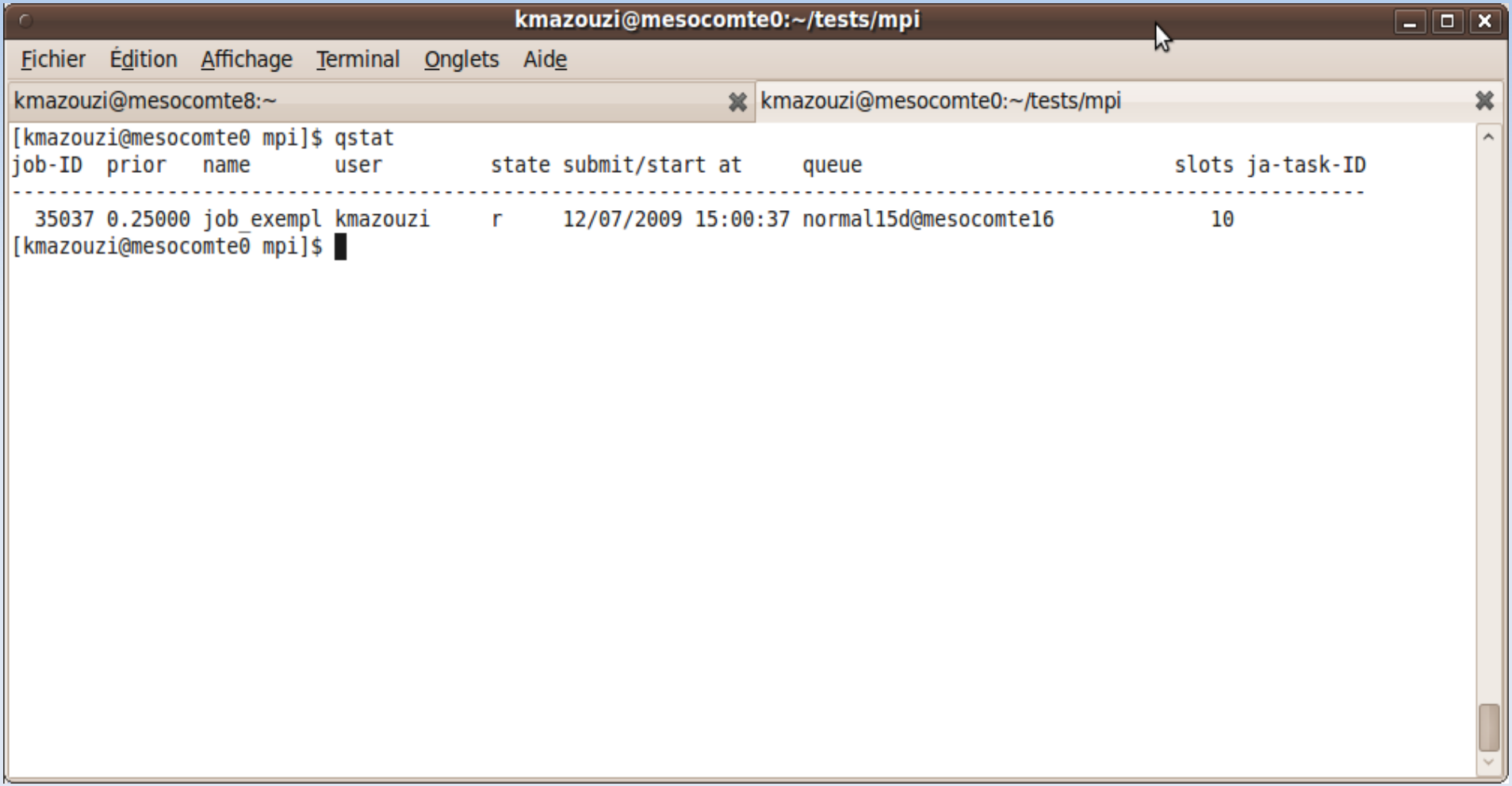

#$ -q normal15d


MPIRUN=/Softs/Intel/impi/3.2.1
.009/bin64/mpirun

$MPIRUN -n $NSLOTS ./hello
```

- Temps maximum pour l'exécution
- Nom du fichier ou stocker la sortie
- Nom du job
- Environnement parallele. 10 = nombre de coeurs
- Queue utilisée
- Le chemin du mpirun Intel 3.2.1.009
- Exécution de l'application

Résultats : qstat



Résultats de l'exécution

Hello, world. I am 7 of 10.

Hello, world. I am 6 of 10.

Hello, world. I am 1 of 10.

Hello, world. I am 3 of 10.

Hello, world. I am 5 of 10.

Hello, world. I am 0 of 10.

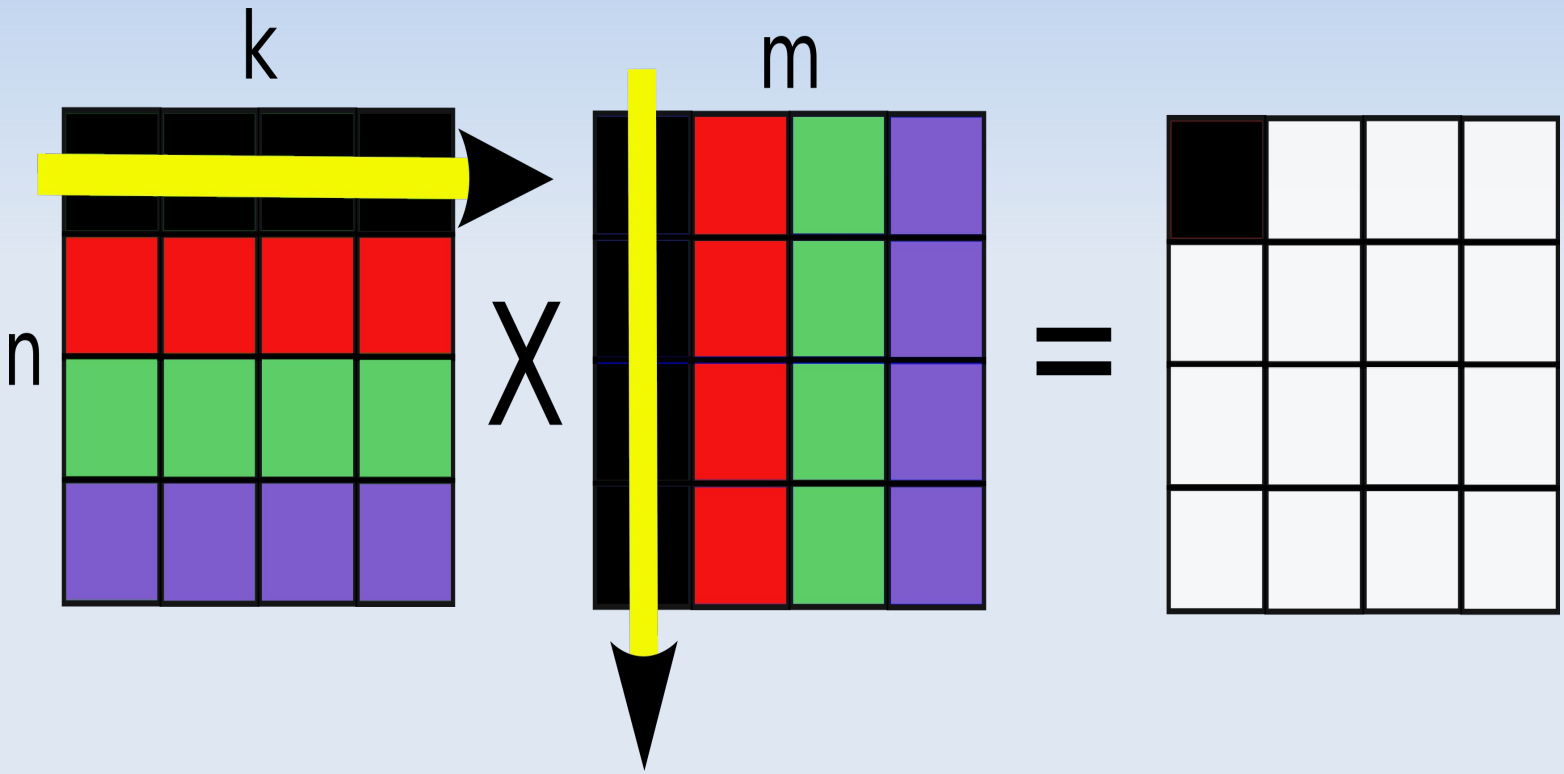
Hello, world. I am 4 of 10.

Hello, world. I am 8 of 10.

Hello, world. I am 2 of 10.

Hello, world. I am 9 of 10.

Produit matriciel MPI



Exemple matrice (C)

Exemple matmul_mpi.c

Matrices OpenMP

- Compilation

```
mesocomte0$ mpiicc matmul_mpi.c -o matmul
mesocomte0$
```

- Script

```
#!/bin/bash
#$ -l h_rt=360:00:0
#$ -o $JOB_NAME.o$JOB_ID
#$ -e $JOB_NAME.e$JOB_ID
#$ -N job_matmul_mpi
#$ -q normal15d
#$ -pe impi_tight 10

MPIRUN=/Softs/Intel/impi/3.2.1.009/bin64/mpirun

$MPIRUN -n $NSLOTS ./matmul_mpi 100000 10000 10000


exit 0
```

- Exécution

Questions ?