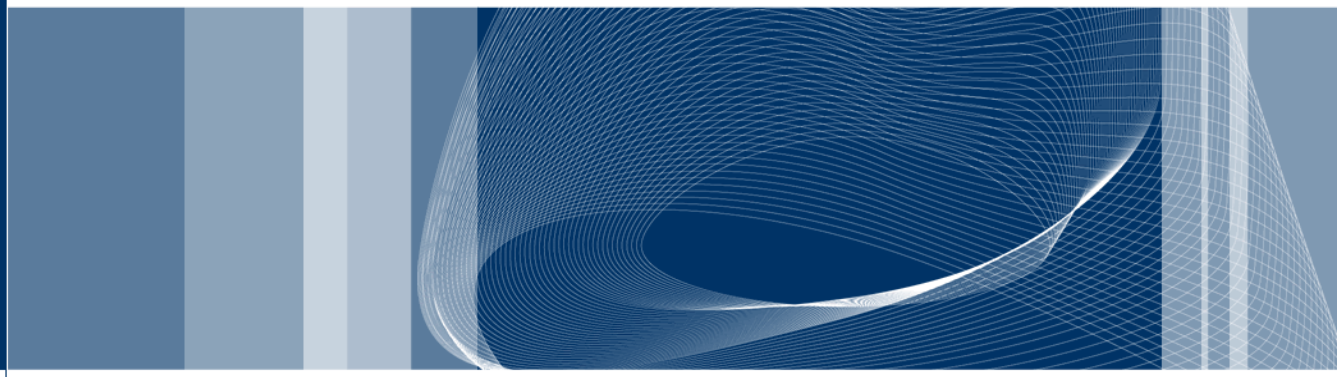
POLITECNICO DI MILANO



Esercizi MPI

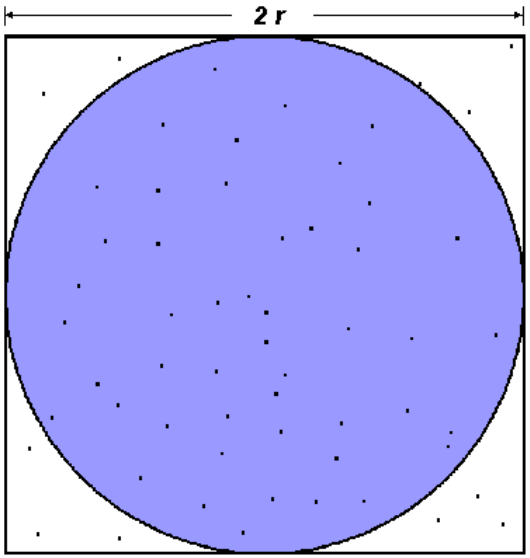
Algoritmi e Calcolo Parallelo

Daniele Loiacono

Esercizio 1

- Implementare in MPI una soluzione parallela del seguente algoritmo per approssimare PI

```
cin >> npoints; count = 0;
for(j=0, j<npoints; j++) {
      x = random();
      y = random();
      if (inCircle(x, y))
            count++;
}
PI = 4.0*count/npoints
```



$A_S = (2r)^2 = 4r^2$

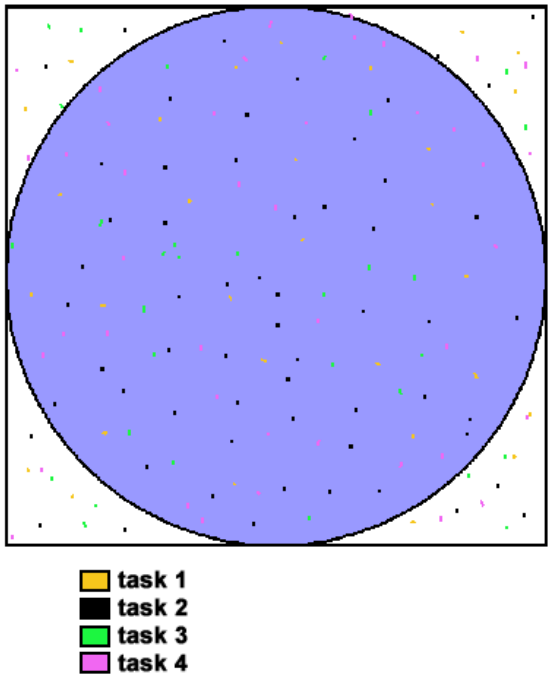
$A_C = \pi r^2$

$\pi = 4 \times \frac{A_C}{A_S}$

- Non è necessario implementare la funzione `inCircle()`
- Si implementi
  - una soluzione senza utilizzare le funzioni di comunicazione collettive
  - una soluzione in cui vengono utilizzate le funzioni di comunicazione collettive

Esercizio 1: soluzione


```
cin >> npoints; count = 0;
p = number of tasks; N = npoints/p;

for(j=1, j<N; j++) {
    x = random();
    y = random();
    if (inCircle(x, y))
         count++;
}
PI = 4.0*count/N
find out if I am MASTER or WORKER
if I am MASTER
   receive from WORKERS their PI
   compute PI (use MASTER and WORKER calculations)
else if I am WORKER {
  send to MASTER PI
}
```



Esercizio 1: implementazione (1)

```
int main (int argc, char *argv[]){
    MPI_Init(&argc,&argv);
    MPI_Comm_size(MPI_COMM_WORLD,&numtasks);
    MPI_Comm_rank(MPI_COMM_WORLD,&taskid);
    if (taskid == MASTER)
      cin >> npoints;
    Send/receive di npoints
    N = npoints/numtasks; count = 0;
    for(j=1, j<N; j++) {
       x = random(); y = random();
      if (inCircle(x, y))
        count++; }
    myPI = 4.0*count/N;
    Send/receive di myPI e riduzione PI
    if (taskid == MASTER)
      cout << "PI=" << PI << endl;
    MPI_Finalize();
    return 0;
}
```

Esercizio 1: implementazione (2)

Send/receive di npoints

```
if (taskid == MASTER)
{
   for (int i=1; i< numtasks; i++)
         MPI_Send(&npoints, 1, MPI_INT, i, 0, MPI_COMM_WORLD);
}
else
{
   MPI_Recv(&npoints, 1, MPI_INT, MPI_ANY_SOURCE,
                  MPI_ANY_TAG, MPI_COMM_WORLD, &status);
}
```

Esercizio 1: implementazione (3)

Send/receive di myPI e riduzione PI

```
if (taskid != MASTER) {
     MPI_Send(&myPI, 1, MPI_DOUBLE, MASTER, 0, MPI_COMM_WORLD);
else {
     PI=myPI;
     for (i = 1; i < numtasks; i++) {
            MPI_Recv(&pirecv, 1, MPI_DOUBLE, MPI_ANY_SOURCE,
                        MPI_ANY_TAG, MPI_COMM_WORLD, &status);
            PI = PI + pirecv;
     }
     PI = PI/numtasks;
```

Esercizio 1: implementazione alternativa

```
int main (int argc, char *argv[]){
    MPI_Init(&argc,&argv);
    MPI_Comm_size(MPI_COMM_WORLD,&numtasks);
    MPI_Comm_rank(MPI_COMM_WORLD,&taskid);
    if (taskid == MASTER)
       cin >> npoints;
    MPI_Bcast(&npoints, 1, MPI_INT, MASTER, MPI_COMM_WORLD);
    N = npoints/numtasks; count = 0;
    for(j=1, j<N; j++) {
       x = random(); y = random();
       if (inCircle(x, y))
        count++;}
    myPI = 4.0*count/N;
    MPI_Reduce(&myPI, &PI, 1, MPI_DOUBLE, MPI_SUM, MASTER,
        MPI_COMM_WORLD);
   if (taskid == MASTER) cout << "PI=" << PI/numtasks << endl;
   MPI_Finalize();
   return 0;
}
```

Esercizio 2

- Parallelizzare in MPI il seguente algoritmo che
  - conta i numeri primi inferiori a LIMIT
  - ritorna il numero primo più grande trovato

```
pc=4;   /* 2,3,5,7 are counted here */
for (n=11; n<=LIMIT; n=n+2) {
   if (isprime(n)) {
      pc++;
      foundone = n;
   }
}
cout << "found= " << pc << " largest= " << foundone << endl;
```

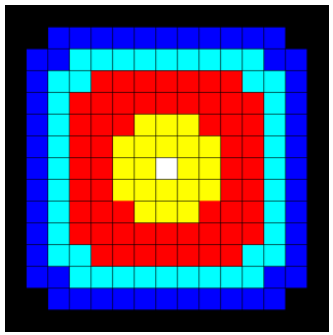
Esercizio 2 - Soluzione

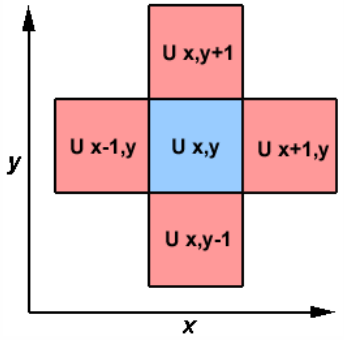
```
MPI_Init(&argc,&argv);
MPI_Comm_rank(MPI_COMM_WORLD,&rank);
MPI_Comm_size(MPI_COMM_WORLD,&ntasks);

mystart = (rank*2)+11
stride   = ntasks*2;
pc=0; foundone = 0;

for (n=mystart; n<=LIMIT; n=n+stride) {
  if (isprime(n)) {
    pc++;
    foundone = n;
}
MPI_Reduce(&pc,&pcsum,1,MPI_INT,MPI_SUM,MASTER,MPI_COMM_WORLD);
MPI_Reduce(&foundone,&maxprime,1,MPI_INT,MPI_MAX,MASTER,MPI_COMM_WORLD);

if (rank == MASTER) {
   pcsum = pcsum + 4;  // conta i primi 4 (2,3,5,7)
   cout << "found= " << pcsum << " largest= " << maxprime << endl;}

MPI_Finalize();
```

Esercizio 3

- Proporre una soluzione MPI per parellizzare su una macchina con M processori il calcolo di una semplice equazione di calore su una matrice NxN:

$$U_{x,y} = U_{x,y} + C_x * (U_{x+1,y} + U_{x-1,y} - 2 * U_{xy}) + C_y * (U_{x,y+1} + U_{x,y-1} - 2 * U_{x,y})$$



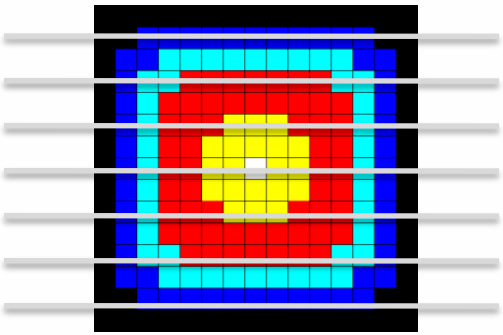


- La soluzione seriale del problema può essere schematizzata come:

```
for (x=1; x < N-1; x++)
   for (y=1; y < N-1; y++)
        u2[x][y] = u[x][y] + cx*(u[x+1][y] + u[x+1][y]
        - 2* u[x][y]) + cy*(u[x][y+1] + u[x][y-1]
        - 2* u[x][y]
```

Esercizio 3 – Soluzione

- Si può scomporre la matrice in blocchi ed utilizzare processo per aggiornare un diverso blocco della matrice



- Attenzione alle righe vicino ai bordi: richiedono di essere sincronizzate attraverso la comunicazione fra processi

Esercizio 3 – Soluzione (dichiarazioni/init)

```
float  u[2][NXPROB][NYPROB]
MPI_Init(&argc,&argv);
MPI_Comm_size(MPI_COMM_WORLD,&numtasks);
MPI_Comm_rank(MPI_COMM_WORLD,&taskid);
numworkers = numtasks-1;
```

Esercizio 3 – Soluzione (divisione lavoro)

```
if (taskid == MASTER) {
   /* Initialize grid here*/
   averow = NXPROB/numworkers;
   extra = NXPROB%numworkers;
   offset = 0;
   for (i=1; i<=numworkers; i++)
   {
      rows = (i <= extra) ? averow+1 : averow;
      if (i == 1) left = NONE;
      else left = i - 1;
      if (i == numworkers) right = NONE;
      else right = i + 1;
      dest = i;
      MPI_Send(&offset, 1, MPI_INT, dest, BEGIN, MPI_COMM_WORLD);
      MPI_Send(&rows, 1, MPI_INT, dest, BEGIN, MPI_COMM_WORLD);
      MPI_Send(&left, 1, MPI_INT, dest, BEGIN, MPI_COMM_WORLD);
      MPI_Send(&right, 1, MPI_INT, dest, BEGIN, MPI_COMM_WORLD);
      MPI_Send(&u[0][offset][0], rows*NYPROB, MPI_FLOAT, dest, BEGIN,
               MPI_COMM_WORLD);
      offset = offset + rows;
   }
```

Esercizio 3 – Soluzione (ricezione risultato)

```
if (taskid == MASTER) {

 /*   DIVISIONE LAVORO  */

   for (i=1; i<=numworkers; i++)
   {
      source = i;
      MPI_Recv(&offset, 1, MPI_INT, source, DONE, MPI_COMM_WORLD,
               &status);
      MPI_Recv(&rows, 1, MPI_INT, source, DONE, MPI_COMM_WORLD, &status);
      MPI_Recv(&u[0][offset][0], rows*NYPROB, MPI_FLOAT, source,
               msgtype, MPI_COMM_WORLD, &status);
   }

   /* Write output*/

   MPI_Finalize();
}
```

Esercizio 3 – Soluzione (setup iniziale worker)

```
if (taskid != MASTER)
   {

      MPI_Recv(&offset, 1, MPI_INT, MASTER, BEGIN, MPI_COMM_WORLD, &status);
      MPI_Recv(&rows, 1, MPI_INT, MASTER, BEGIN, MPI_COMM_WORLD, &status);
      MPI_Recv(&left, 1, MPI_INT, MASTER, BEGIN, MPI_COMM_WORLD, &status);
      MPI_Recv(&right, 1, MPI_INT, MASTER, BEGIN, MPI_COMM_WORLD, &status);
      MPI_Recv(&u[0][offset][0], rows*NYPROB, MPI_FLOAT, MASTER, BEGIN,
               MPI_COMM_WORLD, &status);

      start=offset;
      end=offset+rows-1;
      if (offset==0)    start=1;
      if ((offset+rows)==NXPROB)   end--;
```

…

Esercizio 3 – Soluzione (calcolo risultato)

```
 if (taskid != MASTER){
…
    /* SETUP WORKER */
       iz = 0;
       for (it = 1; it <= STEPS; it++){
          if (left != NONE){
             MPI_Send(&u[iz][offset][0], NYPROB, MPI_FLOAT, left,
                      RTAG, MPI_COMM_WORLD);
             MPI_Recv(&u[iz][offset-1][0], NYPROB, MPI_FLOAT, left,
                       LTAG, MPI_COMM_WORLD, &status);
          }
          if (right != NONE){
             MPI_Send(&u[iz][offset+rows-1][0], NYPROB, MPI_FLOAT, right,
                       LTAG, MPI_COMM_WORLD);
             MPI_Recv(&u[iz][offset+rows][0], NYPROB, MPI_FLOAT, right, RTAG,
                       MPI_COMM_WORLD, &status);
           }
          update(start,end,NYPROB,&u[iz][0][0],&u[1-iz][0][0]);
          iz = 1 - iz;
       }
       …
```

Esercizio 3 – Soluzione (invio risultato)

```
 if (taskid != MASTER){
…
    /* SETUP WORKER */
    /* CALCOLO RISULTATO */

       MPI_Send(&offset, 1, MPI_INT, MASTER, DONE, MPI_COMM_WORLD);
       MPI_Send(&rows, 1, MPI_INT, MASTER, DONE, MPI_COMM_WORLD);
       MPI_Send(&u[iz][offset][0], rows*NYPROB, MPI_FLOAT, MASTER, DONE,
                MPI_COMM_WORLD);
}
```