



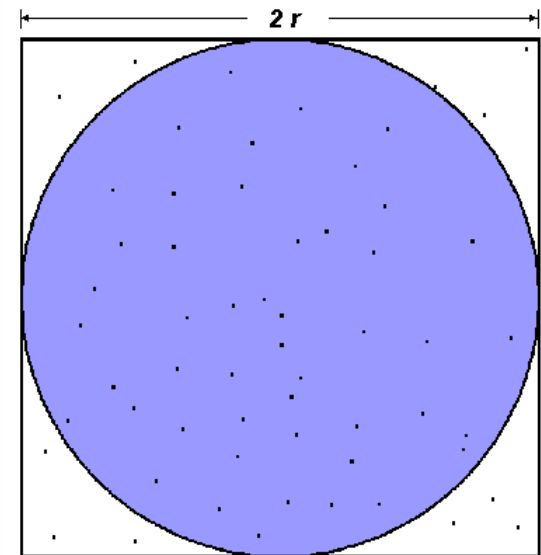
Esercizi Calcolo Parallelo

Algoritmi, Strutture Dati e Calcolo Parallelo

MPI

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

```
scanf("%d",&npoints); count = 0;
for(j=0, j<npoints; j++) {
    x = random();
    y = random();
    if (inCircle(x, y))
        count++;
}
PI = 4.0*count/npoints
```



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

$$A_C = \pi r^2$$

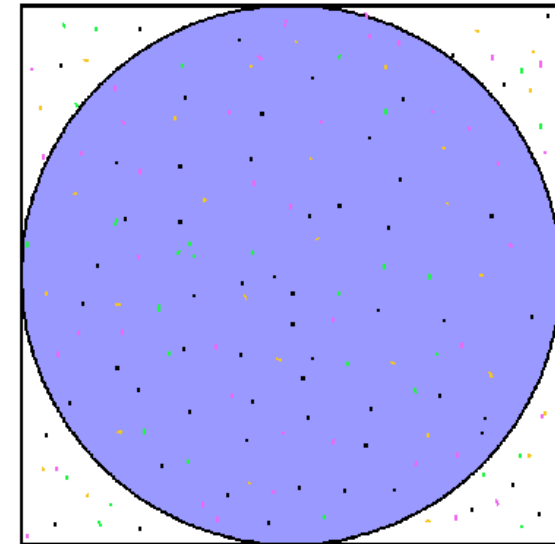
$$\pi = 4 \times \frac{A_C}{A_S}$$

- 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

```
scanf("%d",&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
}
```



task 1
task 2
task 3
task 4

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)
        scanf("%d", &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;
```

Esercizio 1: implementazione (2)

```
int main (int argc, char *argv[]){
...
    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;
        printf("PI = %10.8f\n",PI);
    }
    MPI_Finalize();
    return 0;
}
```

Esercizio 1: implementazione bis (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)
        scanf("%d",&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) printf("PI = %10.8f\n",PI/numtask);
    MPI_Finalize();
    return 0;
}
```

- 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;
    }
}
printf("found= %d, largest=%d\n", pc, foundone);
```


Esercizio 2 - Soluzione

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

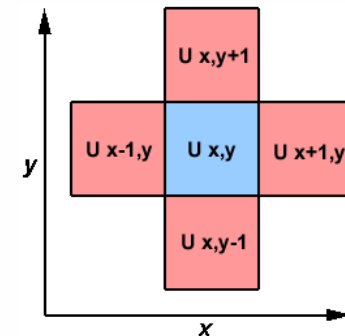
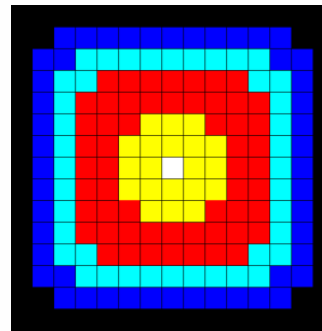
mystart = (rank*2)+11
stride   = ntasks*2;
pc=0; foundone = 0;

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

if (rank == FIRST) {
    pc = pc + 4; // conta i primi 4 (2,3,5,7)
    printf("Done. Largest prime is %d Total primes %d\n", maxprime, pcsum);
}
MPI_Finalize();
```

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

$$\begin{aligned}
 U_{x,y} &= U_{x,y} \\
 &+ C_x * (U_{x+1,y} + U_{x-1,y} - 2 * U_{xy}) \\
 &+ C_y * (U_{x,y+1} + U_{x,y-1} - 2 * U_{x,y})
 \end{aligned}$$

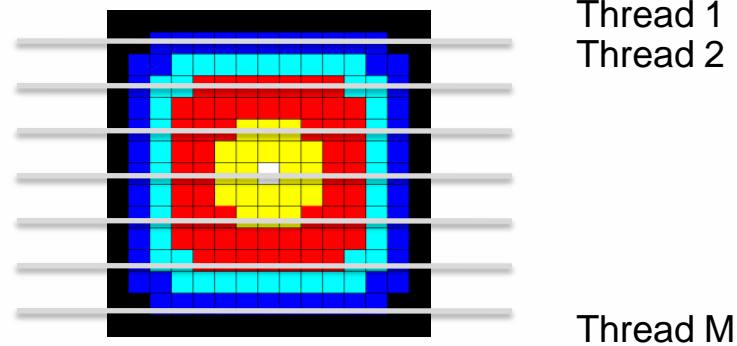


- La soluzione seriale del problema può essere schematizzata come:

```

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])
  
```

- 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;
    }
...
}
```

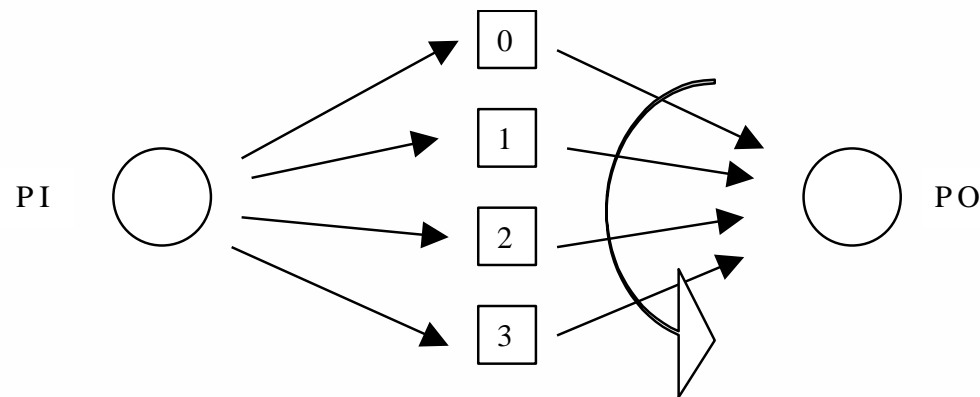

Esercizio 3 – Soluzione (invio risultato)

```
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);  
    MPI_Finalize();  
}
```

Threads

Esercizio 1 (condition variables)

- ❑ Il thread PI esegue ripetutamente le seguenti operazioni:
 - ▶ legge da tastiera una coppia di valori $\langle i, ch \rangle$, dove i è un numero tra 0 e 3, ch un carattere
 - ▶ inserisce il carattere ch nel buffer i (ognuno dei quattro buffer contiene al più un carattere)
- ❑ Il thread PO considera a turno in modo circolare i quattro buffer e preleva il carattere in esso contenuto, scrivendo uscita la coppia di valori $\langle i, ch \rangle$ se ha appena prelevato il carattere ch dal buffer i
- ❑ Implementare i thread in C++ in modo che l'accesso a ognuno dei buffer sia in mutua esclusione (PI rimane bloccato se il buffer a cui accede è pieno, PO se è vuoto)



Esercizio 1 - Soluzione

```
#define SIZE 5

char buffer[SIZE];
bool empty [SIZE];
pthread_mutex_t mutex_buffer[SIZE];
pthread_cond_t  cond_buffer[SIZE];
void *pi_main(void *arg);
void *po_main(void *arg);

int main(int argc, char *argv[]){
    pthread_t pi,po;
    /* Init variables*/
    for (int i = 0; i<SIZE; i++)
        empty[i] = true;
    printf("In main: creating thread PI\n");
    pthread_create(&pi, NULL, pi_main, NULL);
    pthread_create(&po, NULL, po_main, NULL);
    pthread_exit(NULL);
}
```

Esercizio 1 – Soluzione (2)

```
void *pi_main(void *arg){
    printf("Hello World! I am PI\n");
    while (true) {
        char ch;
        int i;
        scanf("%d",&i);
        scanf("%c",&ch);
        if (i>=0 && i < SIZE){
            pthread_mutex_lock(&mutex_buffer[i]);
            if (empty[i]==false)
                pthread_cond_wait(&cond_buffer[i], &mutex_buffer[i]);
            printf("In PI: writing  %c in buffer %d\n",ch,i);
            empty[i] = false;
            buffer[i] = ch;
            pthread_cond_signal((&cond_buffer[i]));
            pthread_mutex_unlock(&mutex_buffer[i]);
        }
    }
    pthread_exit(NULL);
}
```

Esercizio 1 – Soluzione (3)

```
void *po_main(void *arg)
{
    printf("Hello World! I am PO\n");
    while (true){
        for (int i=0; i<SIZE; i++){
            pthread_mutex_lock(&mutex_buffer[i]);
            if(empty[i]==true)
                pthread_cond_wait(&cond_buffer[i], &mutex_buffer[i]);
            empty[i] = true;
            printf("In PO: buffer[%d] = %c\n",i,buffer[i]);
            pthread_cond_signal(&cond_buffer[i]);
            pthread_mutex_unlock(&mutex_buffer[i]);
        }
    }
    pthread_exit(NULL);
}
```