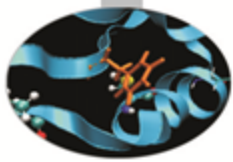
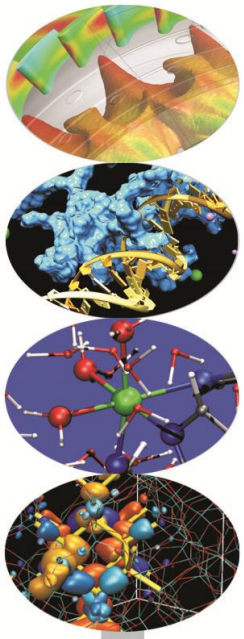
CINECA SCAI
SuperComputing Applications and Innovation



MPI
Laboratorio 2



Isabella Baccarelli
i.baccarelli@cineca.it

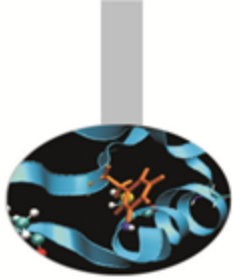
Cristiano Padrin
c.padrin@cineca.it

Mariella Ippolito
m.ippolito@cineca.it

Vittorio Ruggiero
v.ruggiero@cineca.it

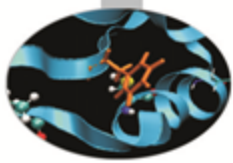
CINECA

Programma della 2° sessione di laboratorio

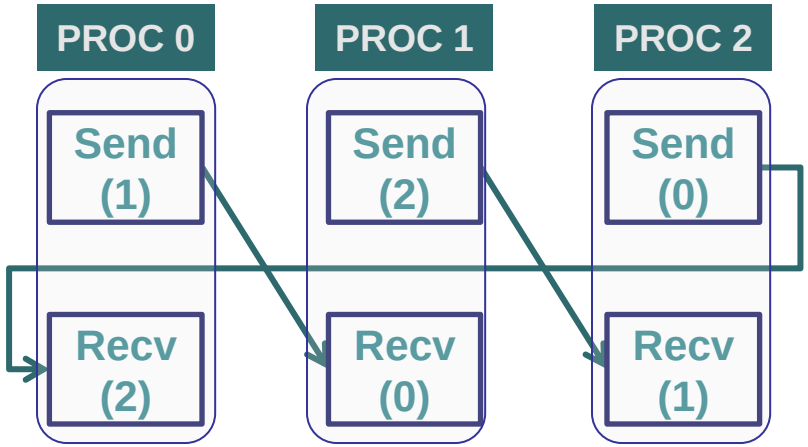


- Funzioni di comunicazione standard e pattern di comunicazione
  - *Shift* circolare con `MPI_Sendrecv` (Esercizio 7)
  - *Array Smoothing* (Esercizio 8)
- Utilizzare le funzioni collettive per implementare pattern di comunicazione standard
  - Calcolo di π con comunicazioni collettive (Eserc. 9)
  - Prodotto matrice-vettore (Esercizio 10)
  - Prodotto matrice-matrice (Esercizio 11)

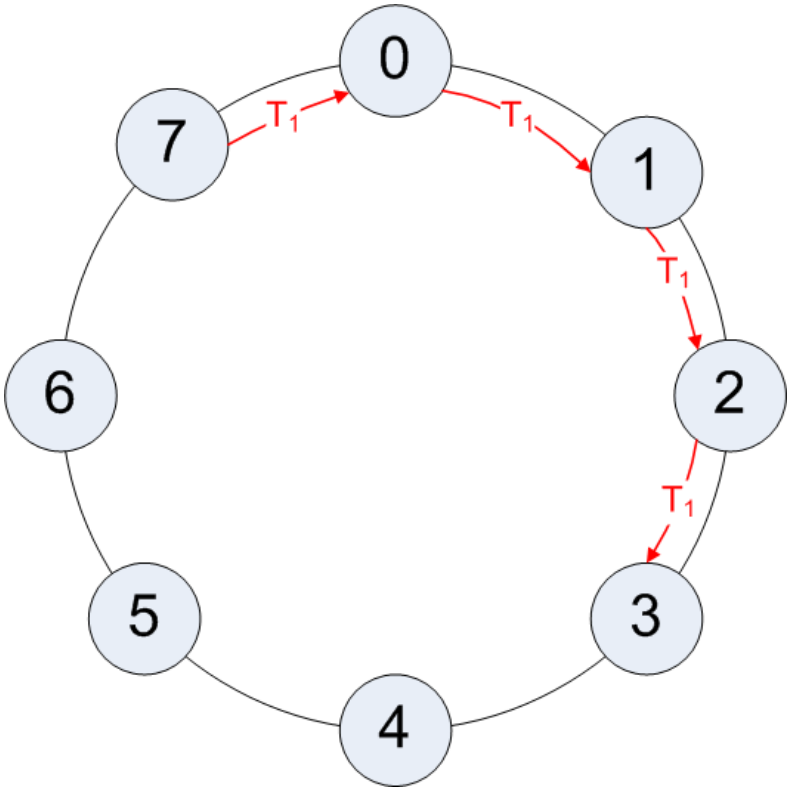
Shift circolare: versione con *Send-Receive*



```c
#include <stdio.h>
#include <mpi.h>
#define MSIZE 50000

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

    MPI_Status status;
    int rank, size, tag, to, from;

    int A[MSIZE], B[MSIZE], i;

    /* Start up MPI environment */
    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Comm_size(MPI_COMM_WORLD, &size);

    to = (rank + 1) % size;
    from = (rank + size - 1) % size;

    for (i = 0; i < MSIZE; i++)
        A[i] = rank;

    MPI_Sendrecv(A, MSIZE, MPI_INT, to, 201, /* sending info */
                 B, MSIZE, MPI_INT, from, 201, /* recving info */
                 MPI_COMM_WORLD, &status);

    printf("Proc %d sends %d integers to proc %d\n", rank, MSIZE, to);
    printf("Proc %d receives %d integers from proc %d\n", rank, MSIZE, from);

    /* Quit MPI environment */
    MPI_Finalize();
    return 0;
}
```

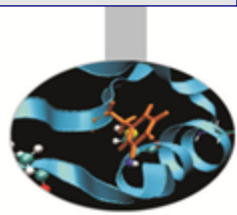


Shift Circolare periodico con `MPI_Sendrecv`

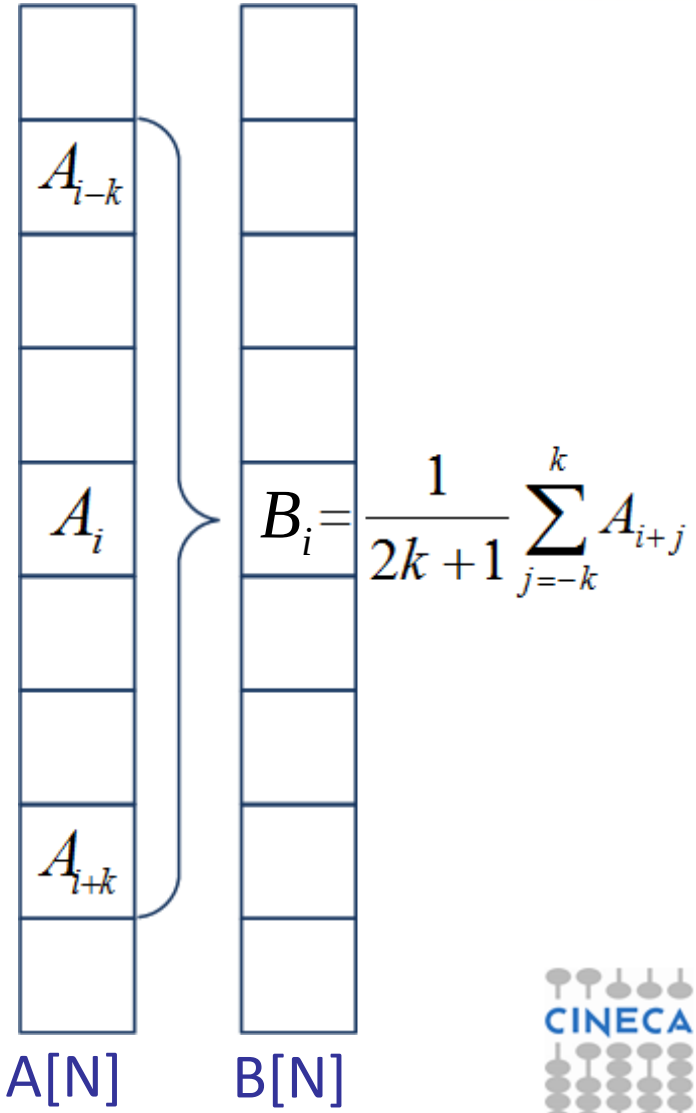


- Ogni processo genera un array `A`, popolandolo con interi pari al proprio rank
- Ogni processo invia il proprio array `A` al processo con rank immediatamente successivo
  - Periodic Boundary: L'ultimo processo invia l'array al primo processo
- Ogni processo riceve l'array `A` dal processo immediatamente precedente e lo immagazzina in un altro array `B`.
  - Periodic Boundary: il primo processo riceve l'array dall'ultimo processo
- Le comunicazioni devono essere di tipo *Sendrecv*

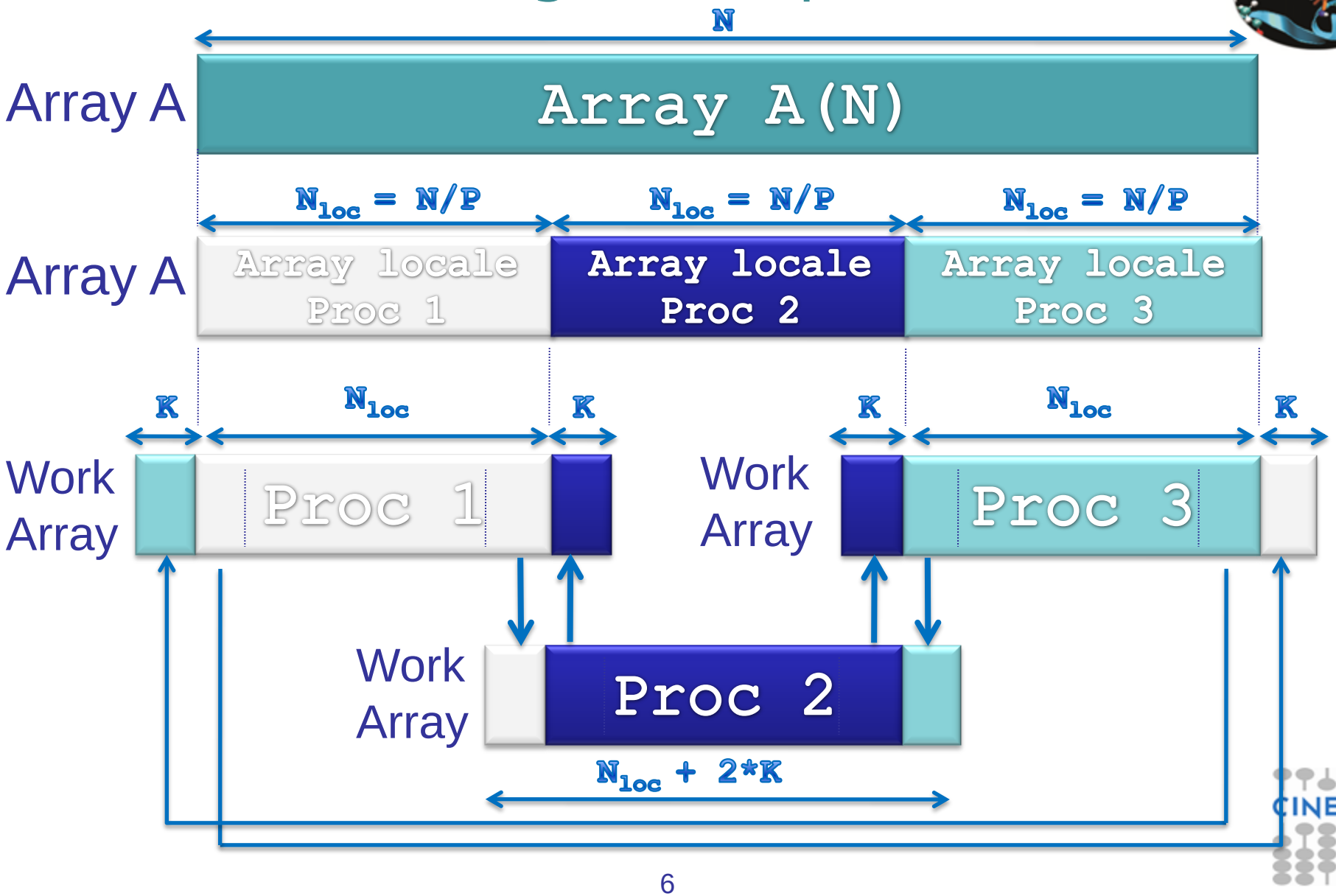
Array smoothing



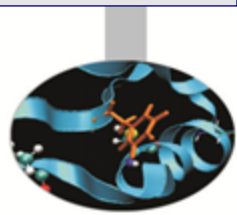
- dato un *array* A[N]
  - inizializzare e stampare il vettore A
- per iter volte:
  - calcolare un nuovo *array* B in cui ogni elemento sia uguale alla media aritmetica del suo valore e dei suoi K primi vicini al passo precedente
    - nota: l'array è periodico, quindi il primo e l'ultimo elemento di A sono considerati primi vicini
  - stampare il vettore B
  - copiare B in A e continuare l'iterazione



Array smoothing: algoritmo parallelo

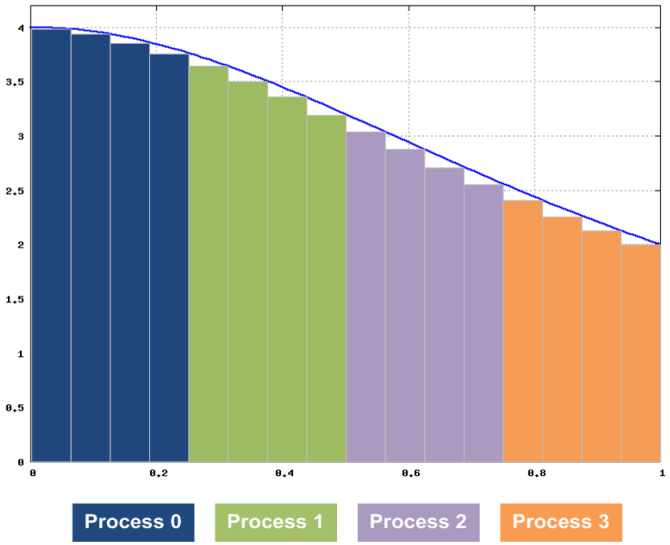


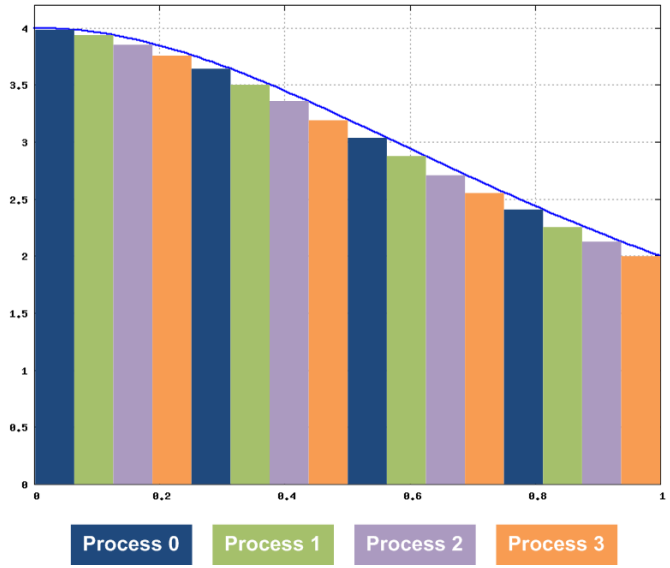
Array smoothing: algoritmo parallelo



- Il processo di *rank* 0
  - genera l'*array* globale di dimensione N , multiplo del numero P di processi
  - inizializza il vettore A con A[i] =i
  - distribuisce il vettore A ai P processi i quali riceveranno N_{loc} elementi nell'array locale (MPI_Scatter)
- Ciascun processo ad ogni passo di *smoothing*:
  - costruisce l'array di lavoro:
    - I primi K elementi dovranno ospitare la copia degli ultimi K elementi *dell'array* locale in carico al processo precedente (MPI_Sendrecv)
    - I successivi N_{loc} elementi dovranno ospitare la copia degli N_{loc} elementi dell'*array* locale in carico al processo stesso
    - Gli ultimi K elementi dovranno ospitare la copia dei primi K elementi del l'*array* locale in carico al processo di *rank* immediatamente superiore (MPI_Sendrecv)
  - Effettua lo *smoothing* degli N_{loc} elementi interni e scrive i nuovi elementi sull'array A
- Il processo di *rank* 0 ad ogni passo raccoglie (MPI_Gather) e stampa i risultati parziali

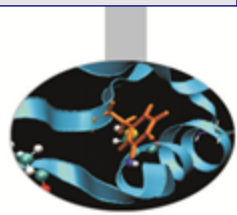
Calcolo di π con *reduction*: algoritmo parallelo



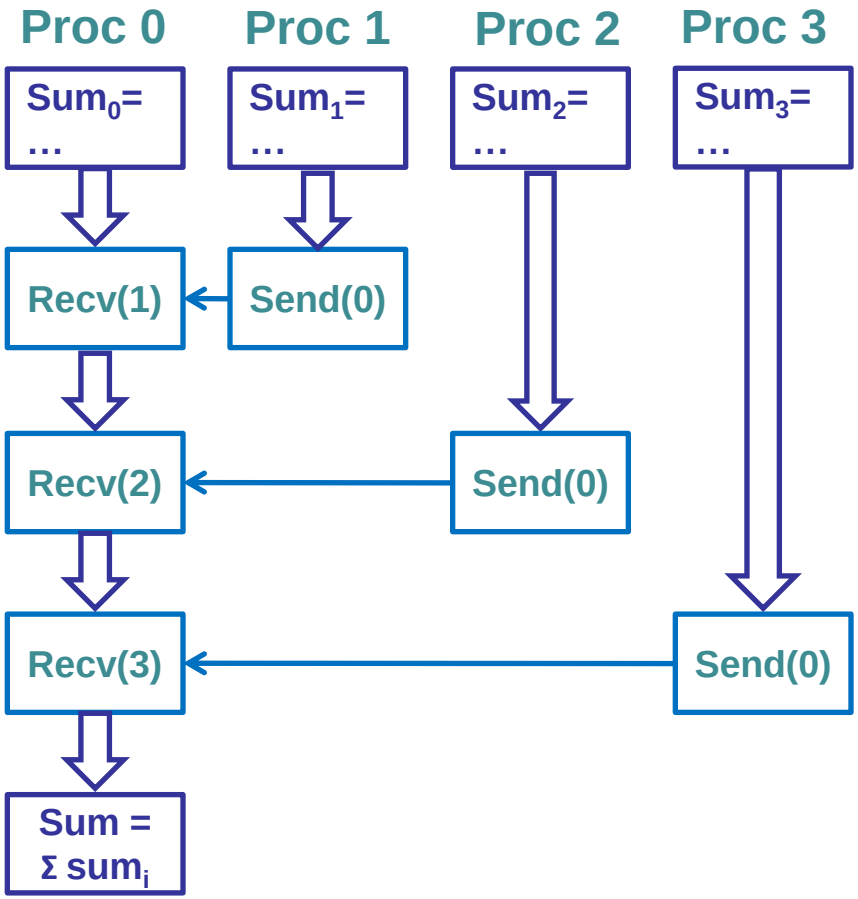


- Ogni processo calcola la somma parziale di propria competenza rispetto alla decomposizione scelta, come nel caso dell'esercizio 3
- Tutti i processi contribuiscono all'operazione di somma globale utilizzando la funzione di comunicazione collettiva `MPI_Reduce`

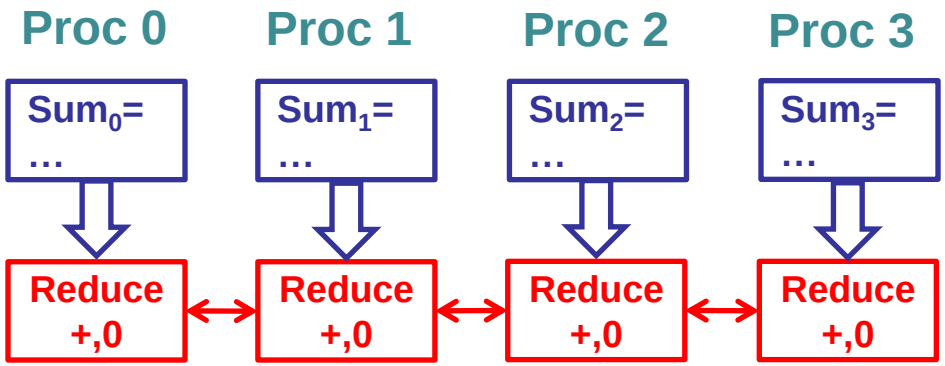
Calcolo di π in parallelo con *reduction*: *flowchart*



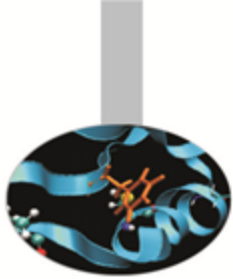
Standard



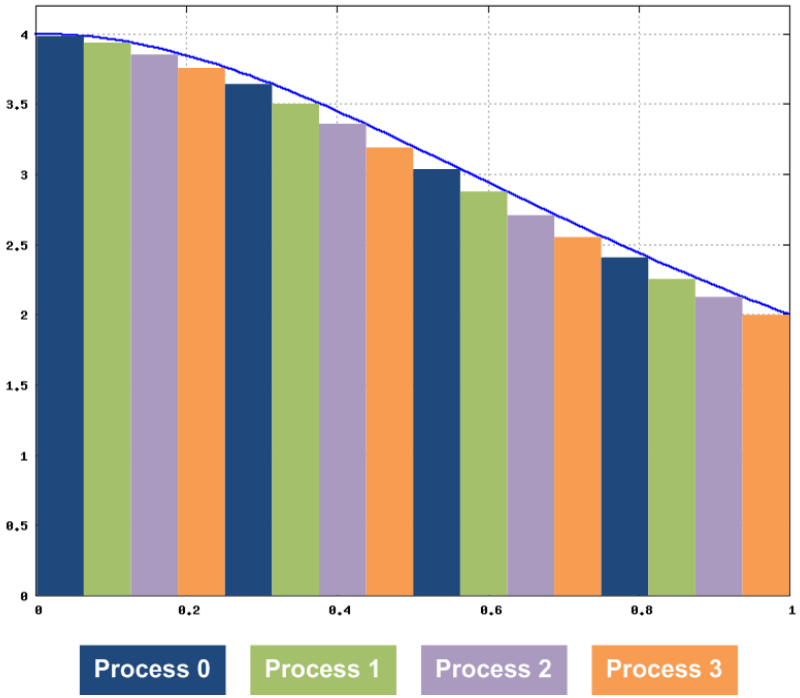
Reduction



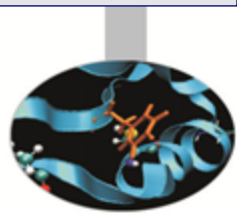
Calcolo di π con *reduce*



```c
#include <stdio.h>
#include "mpi.h"
#define INTERVALS 10000

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

    int rank, nprocs, tag;
    int i;
    int interval = INTERVALS;
    double x, dx, f, sum, pi;

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

    sum = 0.0; dx = 1.0 / (double) interval;

    /* each process computes integral */
    for (i = rank; i < interval; i = i+nprocs) {
        x = dx * ((double) (i - 0.5));
        f = 4.0 / (1.0 + x*x);
        sum = sum + f;
    }
    pi = dx*sum;
    sum = pi; /* using variable sum as sending buffer */

    MPI_Reduce(&sum, &pi, 1, MPI_DOUBLE, MPI_SUM, 0, MPI_COMM_WORLD);

    if (rank == 0)
        printf("Computed PI %.24f\n", pi);

    /* Quit */
    MPI_Finalize();
    return 0;
}
```



Prodotto Matrice-Vettore

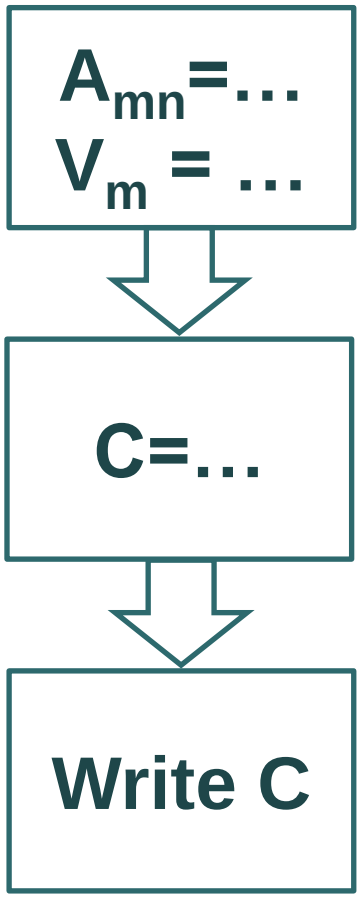


- Data una matrice A, di dimensione `size*size`, ed un vettore V di dimensione `size`, calcolare il prodotto C=A*V
- Ricordando che:

$$C_m = \sum_{n=1}^{size} A_{mn} V_n$$

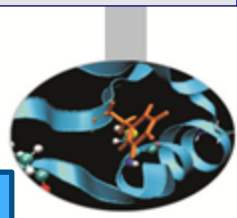
- Nella versione parallela, per semplicità, assumiamo che `size` sia multiplo del numero di processi

Prodotto matrice-vettore: algoritmo seriale

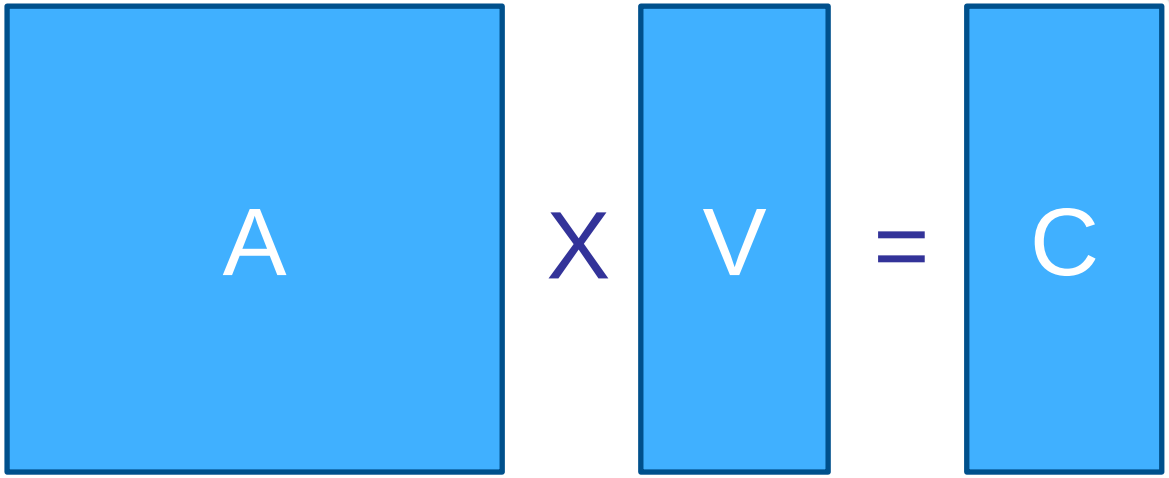


- Inizializzare gli array A e V
  - $A_{mn} = m+n$
  - $V_m = m$
- Core del calcolo
  - Loop esterno sull'indice m=1,`size` di riga della matrice A (e del vettore C)
  - Loop interno sull'indice n=1,size del vettore V
  - Calcolo del prodotto $A_{mn}*V_n$ ed accumulo su C_m
- Scrittura del vettore C

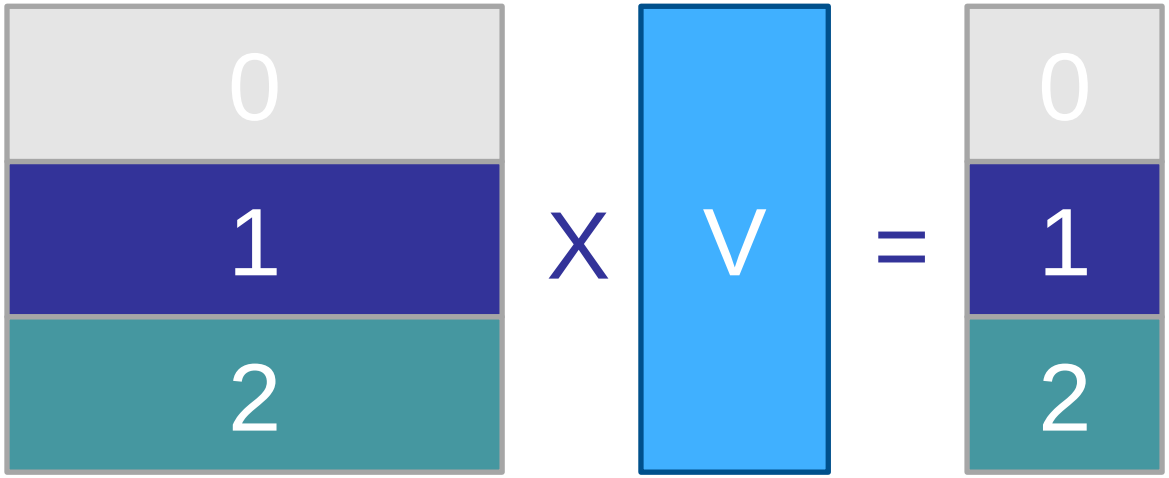
Prodotto matrice-vettore



Versione seriale

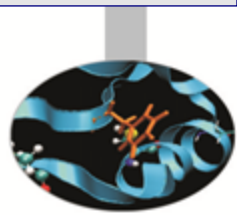


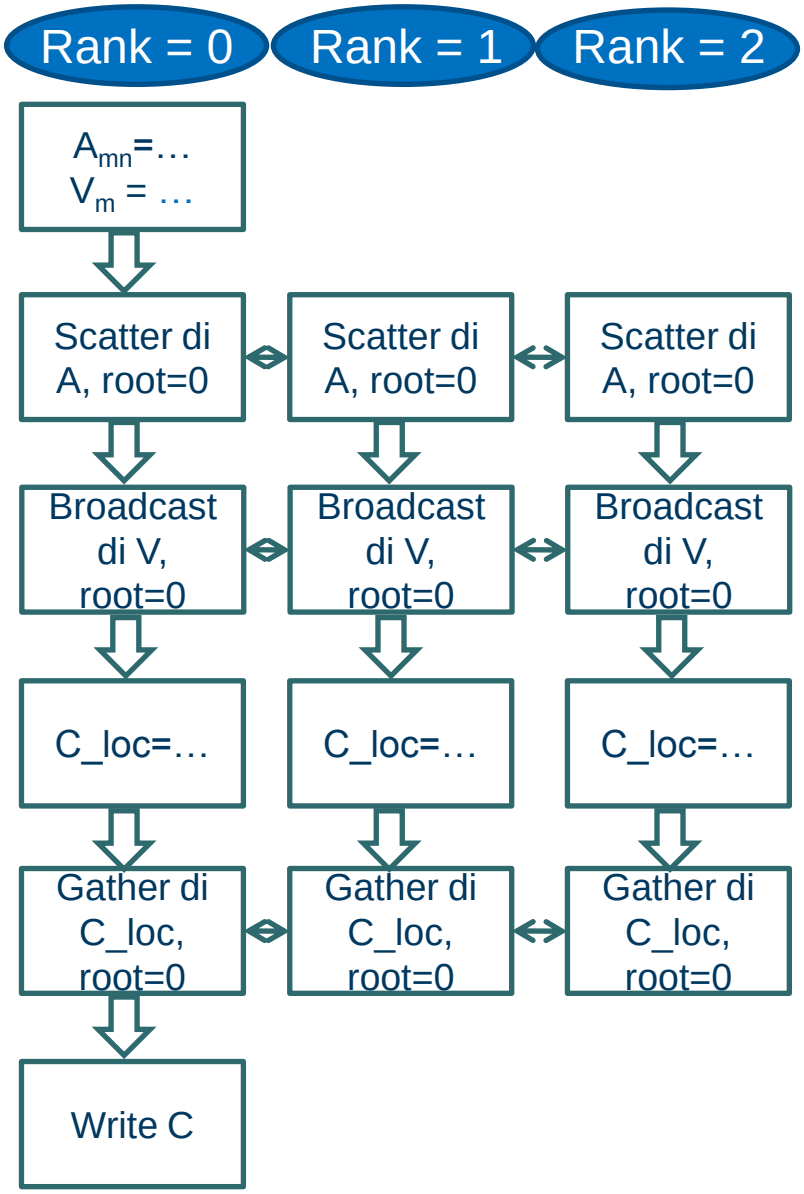
Versione parallela



N.B. Poiché in Fotran le matrici sono allocate per colonne,
è necessario effettuare la trasposizione della matrice A

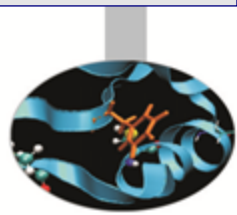
Prodotto matrice-vettore in parallelo





- Inizializzare gli array `A` e `V` sul solo processo master (rank = 0)
- Scatter della matrice `A` e broadcast del vettore `V`
  - Il processo master distribuisce a tutti i processi, se stesso incluso, un sotto-array (un set di righe contigue) di `A`
  - Il processo master distribuibuisce a tutti i processi, se stesso incluso, l'intero vettore `V`
  - I vari processi raccolgono i sotto-array di `A` e `V` in array locali al processo (es. `A_loc` e `V_loc`)
- Core del calcolo sui soli elementi di matrice locali ad ogni processo (es. `A_loc` e `V_loc`)
  - Loop esterno sull'indice `m=1,size/nprocs`
  - Accumulo su C_loc_m
- Gather del vettore `C`
  - Il processo master (rank = 0) raccoglie gli elementi di matrice del vettore risultato C calcolate da ogni processo (C_loc)
- Scrittura del vettore `C` da parte del solo processo master

Prodotto Matrice-Matrice

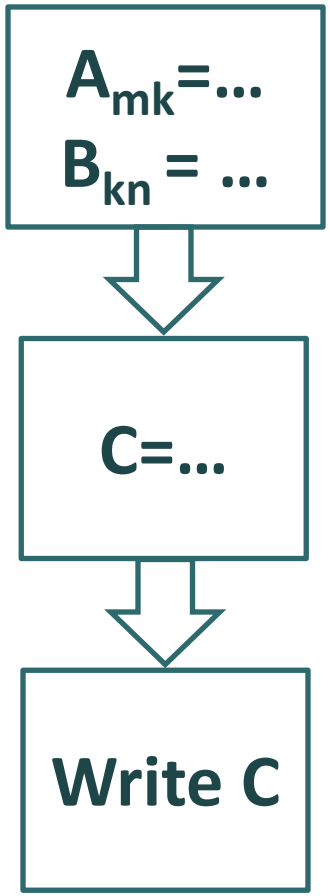


- Date due matrici A e B di dimensione size*size, calcolare il prodotto C=A*B
- Ricordando che:

$$C_{mn} = \sum_{k=1}^{size} A_{mk} B_{kn}$$

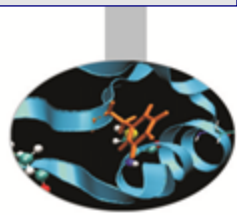
- La versione parallela andrà implementata assumendo che size sia multiplo del numero di processi

Prodotto matrice-matrice: algoritmo seriale

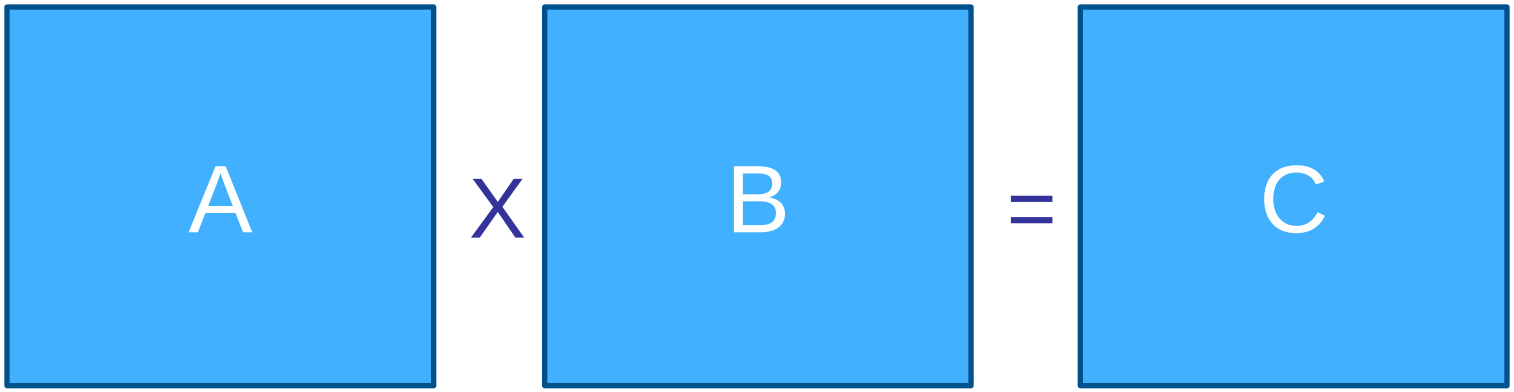


- Inizializzazione degli array A e B
  - A_{mk} = m+k
  - B_{kn} = n+k
- Core del calcolo
  - Loop esterno sull'indice m=1,size di riga della matrice A (e della matrice C)
  - Loop intermedio sull'indice n=1,size di colonna della matrice B (e della matrice C)
  - Loop interno sull'indice k=1,size di colonna della matrice A e di riga della matrice B
  - Calcolo del prodotto A_{mk}*B_{kn} ed accumulo su C_{mn}
- Scrittura della matrice C

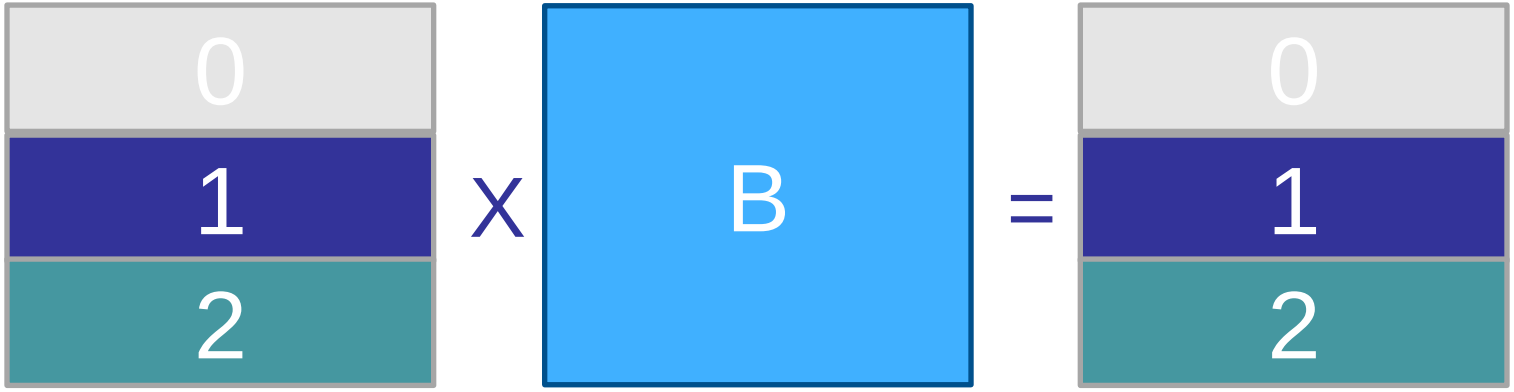
Prodotto matrice-matrice in C



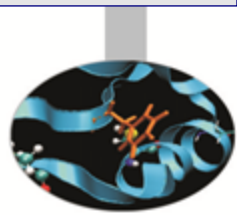
Versione seriale



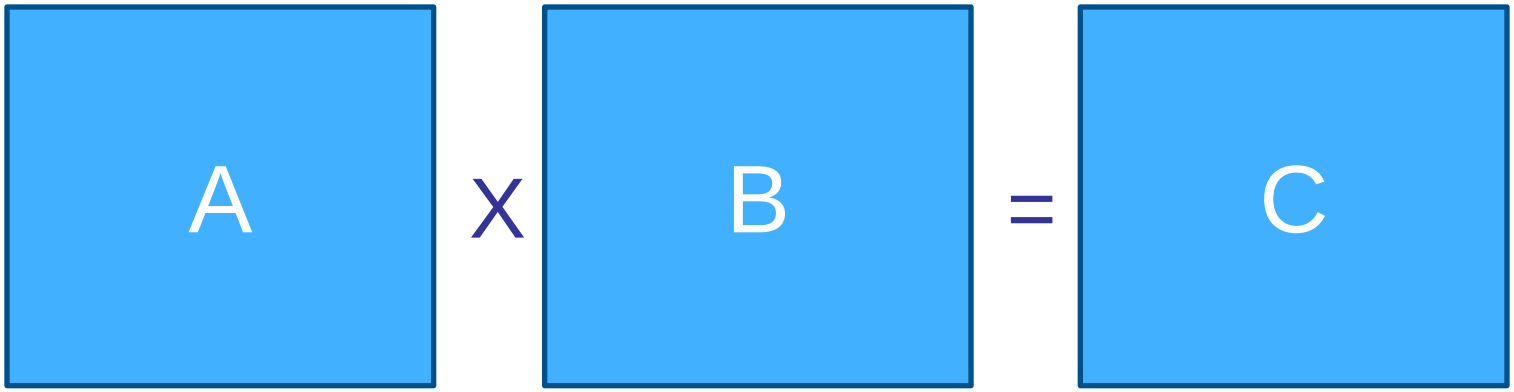
Versione parallela



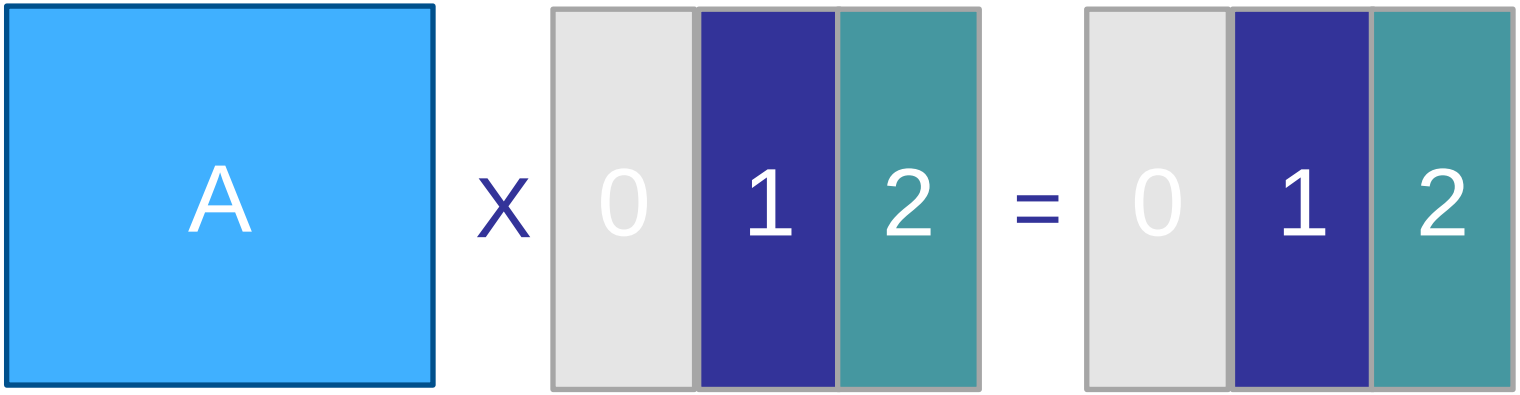
Prodotto matrice-matrice in Fortran



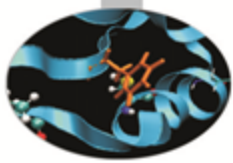
Versione seriale

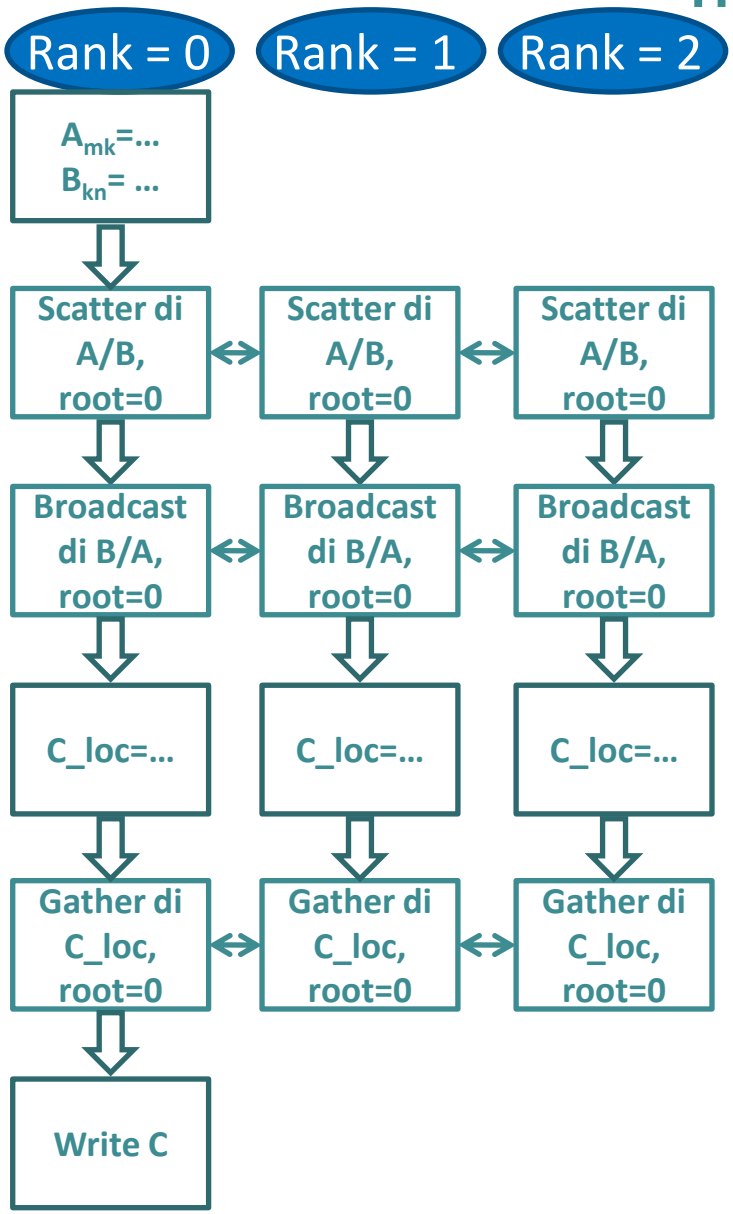


Versione parallela



Prodotto matrice-matrice in parallelo





- Inizializzare le matrici `A` e `B` sul solo processo master (rank = 0)
- In C Scatter della matrice `A` e Broadcast di `B` (viceversa in Fortran)
  - Il processo master distribuisce a tutti i processi, se stesso incluso, un sotto-array (un set di righe contigue) di `A` (B in Fortran)
  - Il processo root distribuisce a tutti i processi, se stesso incluso, l'intera matrice `B (A)`
  - I vari processi raccolgono i sotto-array di `A (B)` in un array locale al processo (es `A_loc`)
- Core del calcolo sui soli elementi di matrice locali ad ogni processo
  - Loop esterno sull'indice `m=1,size/nprocs`
  - Accumulo su C_loc_m
- Gather della matrice `C`
  - Il processo master (rank = 0) raccoglie gli elementi della matrice risultato C calcolate da ogni processo (C_local)
- Scrittura della matrice `C` da parte del solo processo master