17ª Scuola Estiva di CALCOLO PARALLELO

ESERCIZI MPI

CINECA
Consorzio Interuniversitario

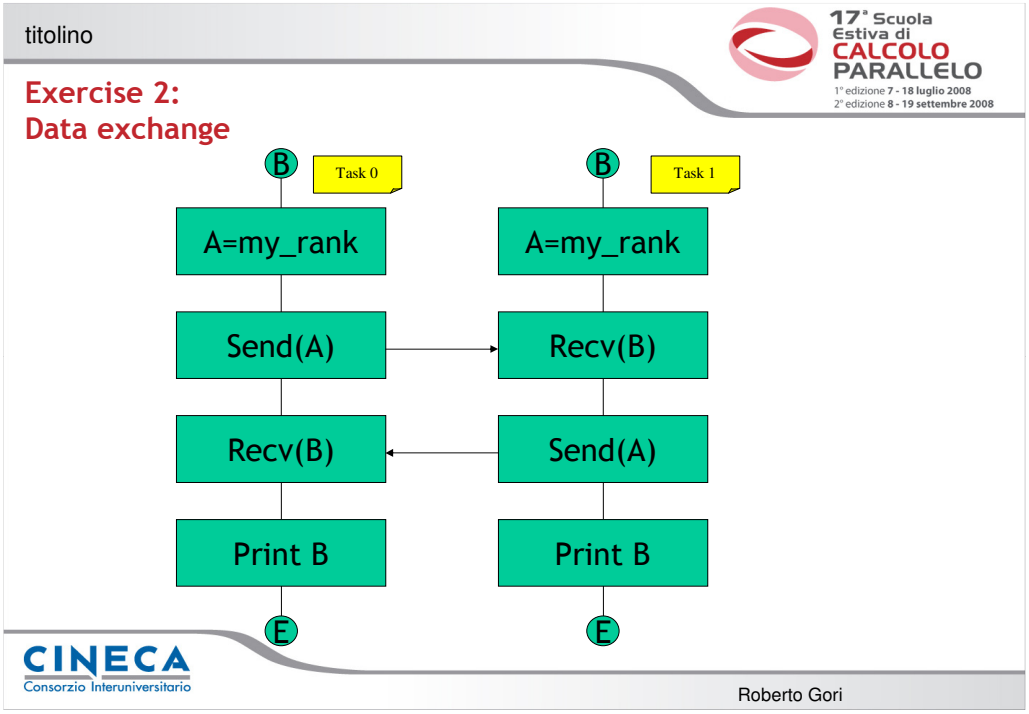
1° edizione **7 - 18 luglio 2008** 2° edizione **8 – 19 settembre 2008**

Exercise 1

Using MPI:

1. Hello world
2. Hello world, I am proc X of total Y (from 1 to total 4 tasks)
3. Do it in the queue


```
program hello
  implicit none
  include 'mpif.h'
  integer ierr,me,nprocs
  call MPI_INIT(ierr)
  call MPI_COMM_SIZE(MPI_COMM_WORLD, nprocs, ierr)
  call MPI_COMM_RANK(MPI_COMM_WORLD, me, ierr)
  print *,'hello, I am task ',me,' of ',nprocs
  call MPI_FINALIZE(ierr)
end program hello
```



Comunicazione tra due processi

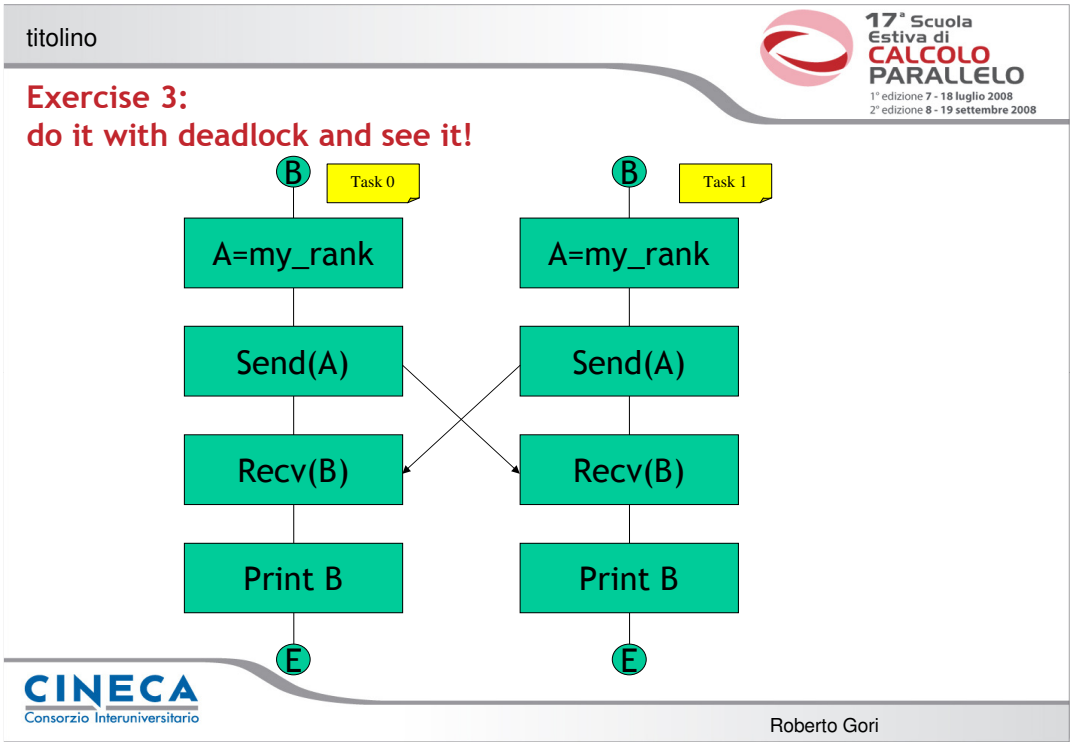
Obiettivo

Questo programma deve implementare lo schema di comunicazione della figura che prevede 2 soli processi:

1. Il processo 0 invia il proprio rank al processo 1;
2. Il processo 1 invia il proprio rank al processo 0;
3. Ciascun processo stampa il proprio rank e quello ricevuto.

Commenti

È buona norma controllare che il numero di processi sia esattamente 2 altrimenti il programma va fermato.



Comunicazione tra due processi con possibilità di deadlock

Obiettivo

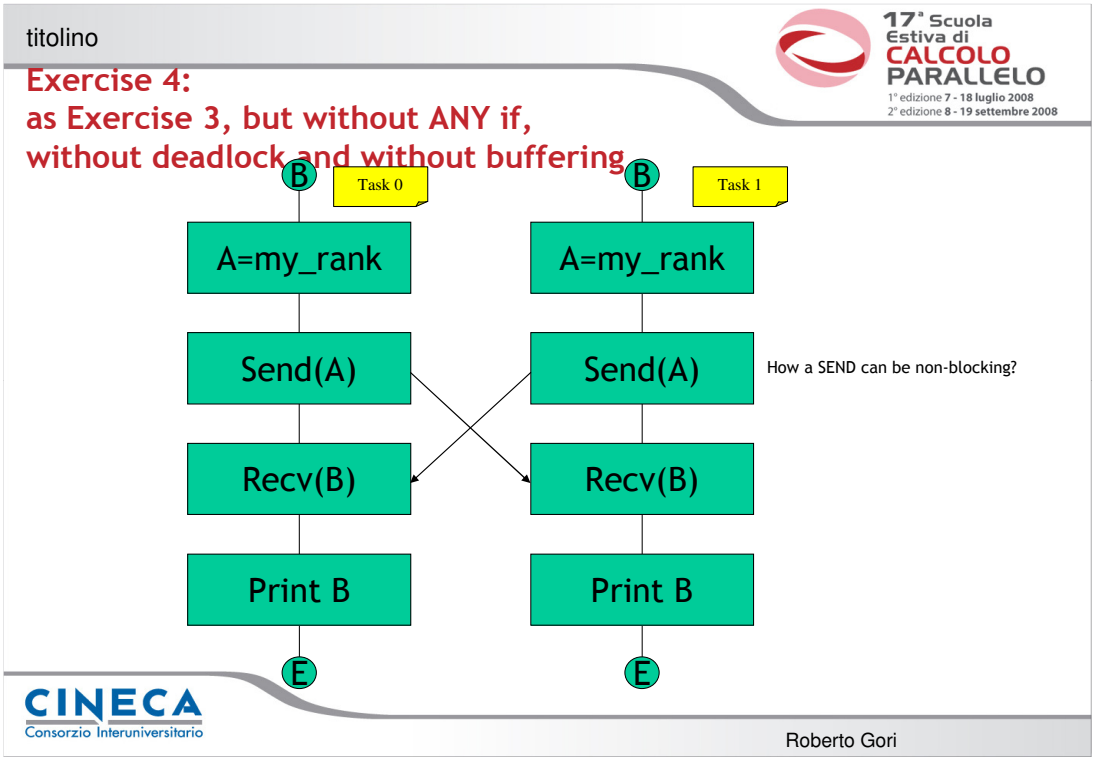
Questo programma deve implementare lo schema di comunicazione della figura che prevede 2 soli processi:

1. Entrambi i processi inviano il proprio rank all'altro
2. Ciascun processo stampa il proprio rank e quello ricevuto.

Commenti

È buona norma controllare che il numero di processi sia esattamente 2 altrimenti il programma va fermato.

Aumentando la dimensione del messaggio scambiato (in questo caso il rank) questo schema di comunicazione genera un deadlock. E' possibile verificare tale effetto trasferendo un vettore di ranks di dimensione opportuna.



Comunicazione tra due processi evitando il deadlock

Obiettivo

Questo programma deve implementare lo schema di comunicazione della figura che prevede 2 soli processi:

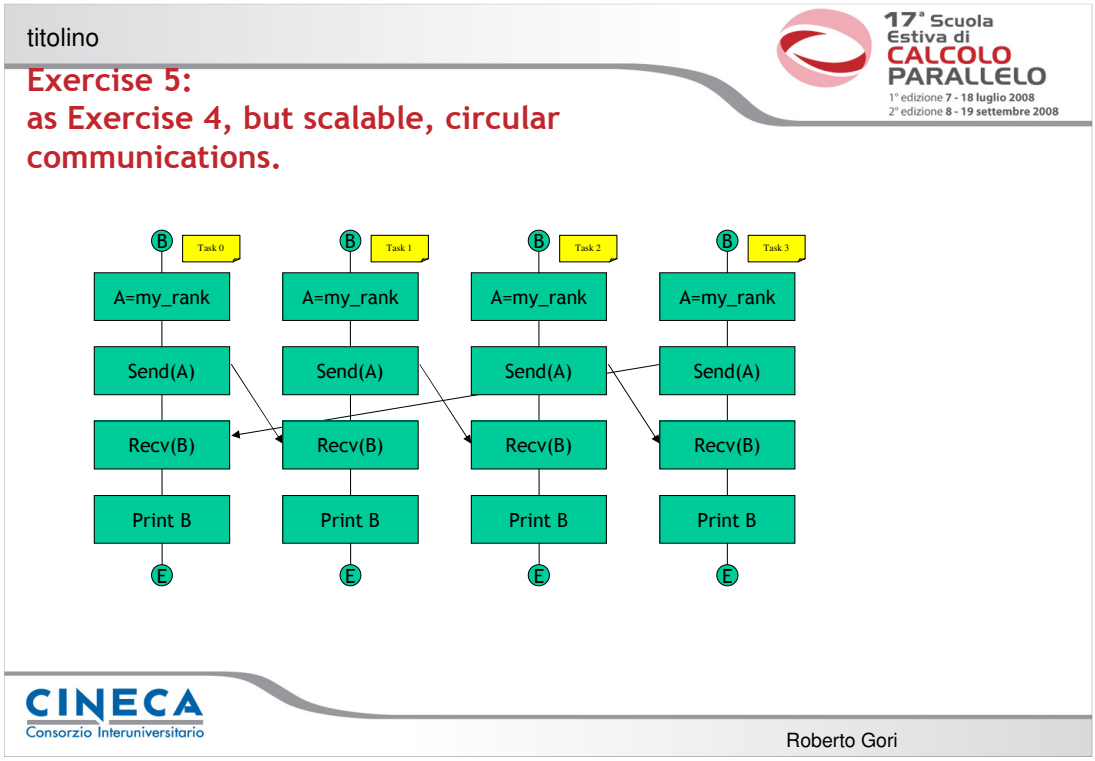
1. Entrambi i processi inviano il proprio rank all'altro
2. Ciascun processo stampa il proprio rank e quello ricevuto.

Commenti

È buona norma controllare che il numero di processi sia esattamente 2 altrimenti il programma va fermato.

Utilizzando opportune primitive (non bloccanti) è possibile evitare il deadlock.

E' possibile implementare questo schema evitando test del tipo "if (myrank == n)"



Comunicazione circolare

Obiettivo

Questo programma prevede un numero qualunque di processi.

Ogni processo deve:

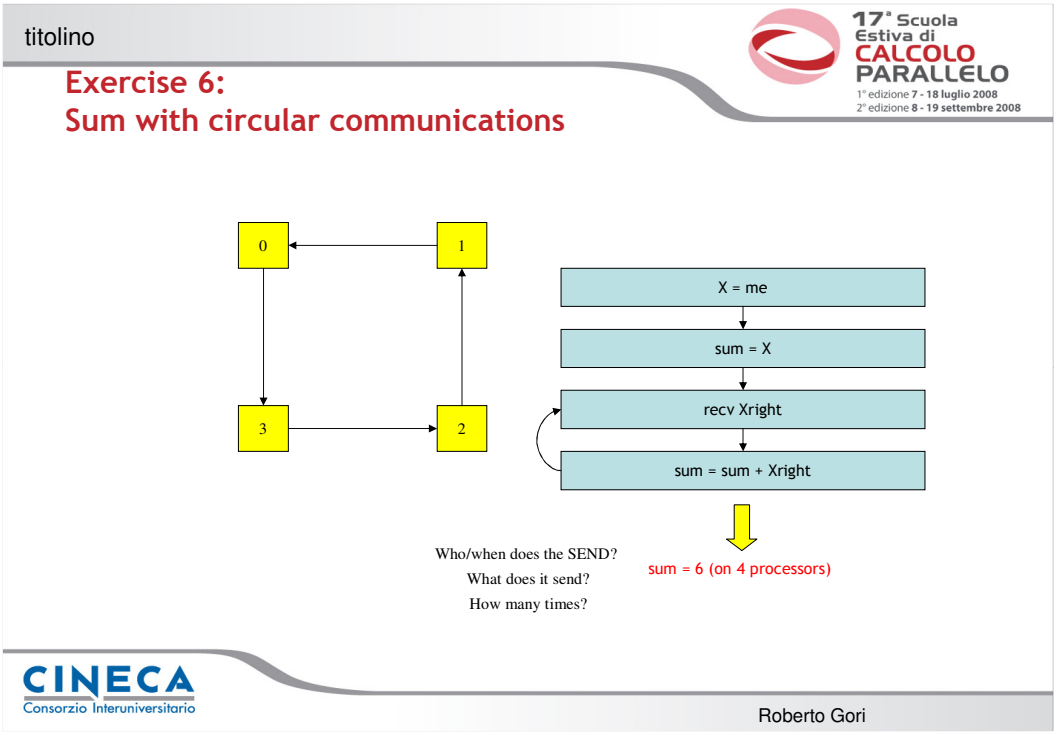
1. inviare il proprio rank al successivo e ricevere quello del precedente;
2. stampare il proprio rank e quello ricevuto.

Commenti

Rispetto all'esercizio precedente occorre generalizzare il metodo che identifica il processo con cui comunicare.

Anche in questo caso è necessario evitare i deadlocks per cui si ricorre a una delle seguenti alternative:

- Primitive non bloccanti
- MPI_Sendrecv (facendo attenzione a distinguere tra i buffers di invio e ricezione)
- Scambiare l'ordine delle primitive tra i processi pari e quelli dispari
- …



Somma circolare

Obiettivo

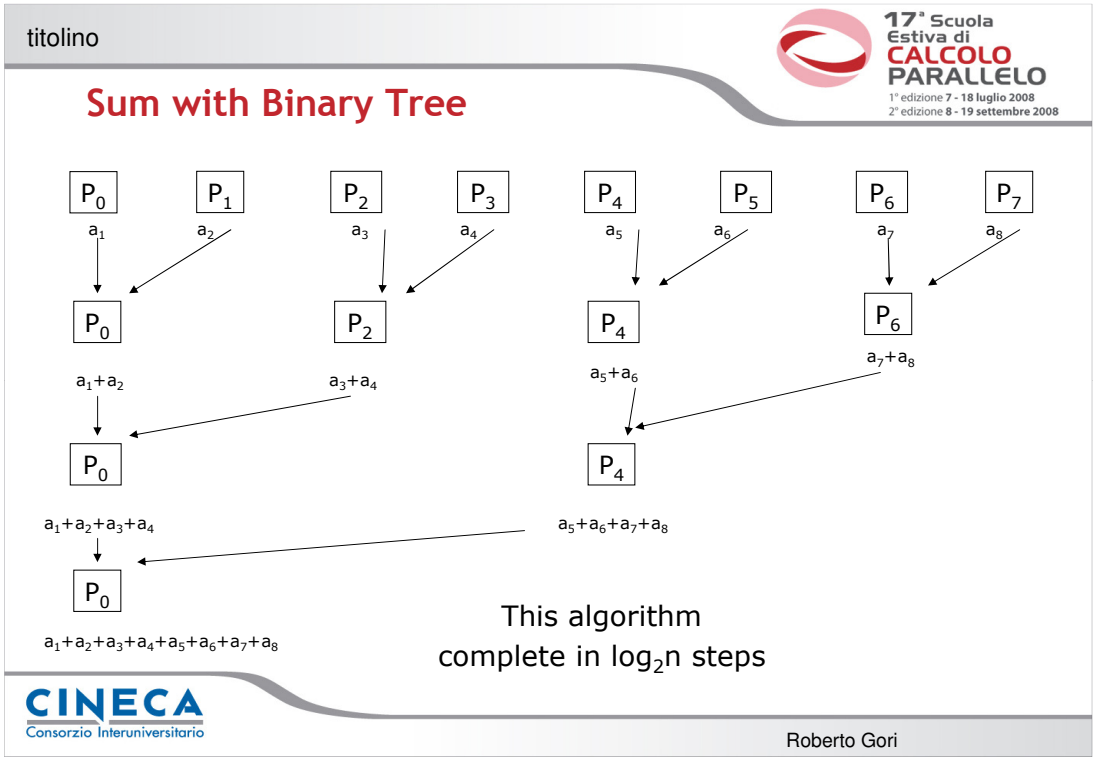
Ciascun processo deve stampare la somma dei ranks scambiando informazioni in maniera circolare come nel caso precedente.

Commenti

Si tratta di applicare iterativamente lo schema dell'esercizio precedente scambiando le opportune informazioni.

Con questo algoritmo è possibile effettuare una somma in un tempo lineare rispetto al numero dei processi.

Ogni processo comunica solo con i vicini.



Somma ad albero binario

Obiettivo

Ottenere nel processo 0 la somma dei ranks secondo lo schema ad albero binario rappresentato nella slide per un numero di processi NP qualunque (anche diverso da una potenza di 2).

Commenti

Il numero di steps è inferiore a quello di un somma circolare ma la somma è disponibile sul solo processo 0.

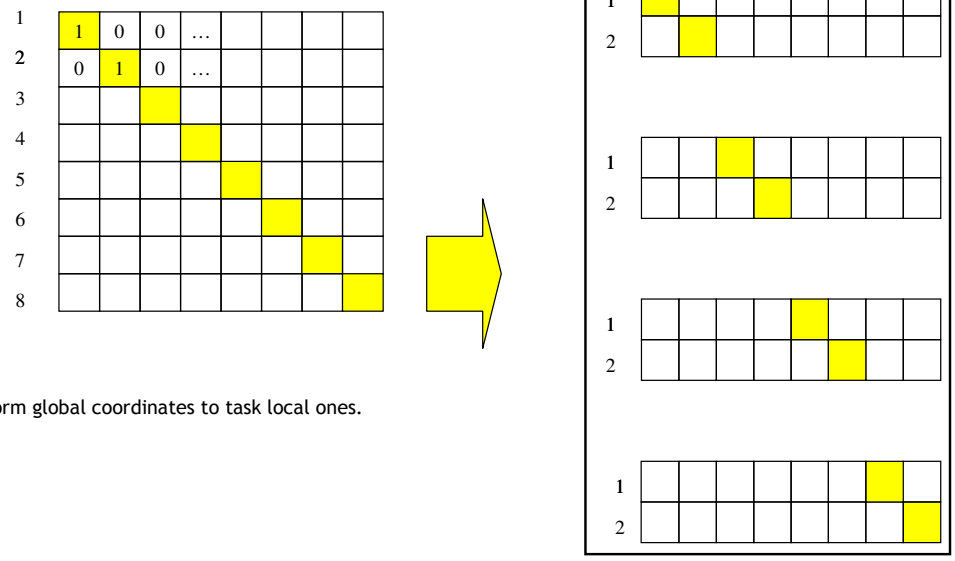
Lo stesso risultato può essere ottenuto con la primitiva collettiva MPI_Reduce.

Inoltre l'algoritmo binario è il migliore solo nel caso di processori/rete totalmente omogenei. Per esempio, nel caso di reti miste (come quella presentata dai cluster di SMP) questo non è necessariamente l'algoritmo migliore.

titolino

17ª Scuola Estiva di CALCOLO PARALLELO
1ª edizione 7 - 18 luglio 2008
2ª edizione 8 - 19 settembre 2008

Exercise 7:
Data distribution of identity



Transform global coordinates to task local ones.

CINECA Consorzio Interuniversitario

Roberto Gori

Matrice identità in forma distribuita

Obiettivo

1. Ogni processo deve allocare ed assegnare la propria porzione di matrice identità
2. Stampare a video o scrivere su file l'intera matrice.
3. Il programma deve funzionare per valori qualsiasi della dimensione della matrice e del numero di processi

Commenti

A seconda del linguaggio utilizzato occorre prestare attenzione alla convenzione nella memorizzazione delle array multidimensionali (Fortran, column-major e C, row-major).

L'output può essere gestito separatamente dai vari processi o delegato al processo 0 (cercando di minimizzare l'impiego di memoria). Nel caso di scrittura concorrente su file è obbligatorio usare le primitive di I/O di MPI-2.

In Fortran è bene salvare utilizzando l'accesso diretto – il più portabile perché non cos- blocked. Leggere il file con "od –f" o "od –F". Su CLX, con il compilatore INTEL bisogna utilizzare il flag "-assume byterecl": di default recl è in unità "word", 4 byte. Unico caso, ma lo standard fortran non lo specifica.

titolino

17ª Scuola Estiva di CALCOLO PARALLELO
1ª edizione 7 - 18 luglio 2008
2ª edizione 8 - 19 settembre 2008

Exercise 8: Physics: transport problem

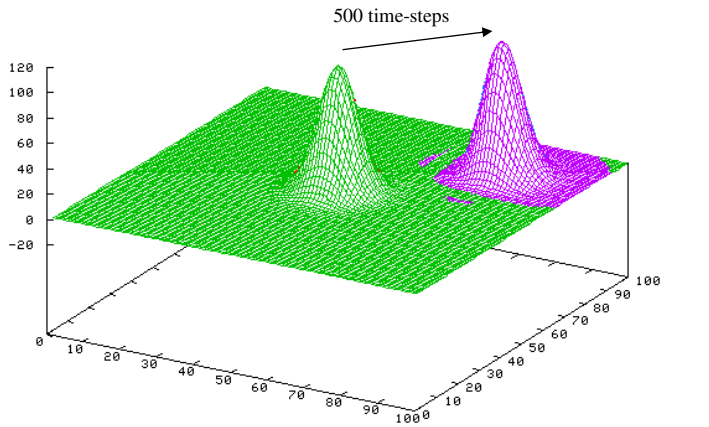
d/dx + d/dy = d/dt

This is the equation of motion, which let original data to be moved along Y=X direction.

Original data are set to be a gaussian distribution

The gaussian moves toward the up-right corner of the system.

Execise: MAKE IT PARALLEL!!!



splot 'transport.dat' w l, 'transport_end.dat' w l

CINECA Consorzio Interuniversitario

Roberto Gori

Parallelizzazione di un algoritmo per la soluzione di un'equazione di trasporto

Obiettivo

Parallelizzare un codice seriale preesistente (transport-serial.f90)

Commenti

Il codice contiene:

temp = array dei dati al tempo "t"

temp_new = array al tempo "t+dt"

NX,NY = dimensione dei dati di griglia

LX,LY = dimensioni reali del sistema

function ix2x(), iy2y() = conversioni coordinate discrete-reali

subroutine init_transport = condizioni iniziali, gaussiana

subroutine save_gnuplot = salva in formato gnuplot/splot

subroutine update_boundaries_FLAT = applica le flat boundary conditions: copia l'ultimo dato dell'area di lavoro sulla cornice

L'algoritmo usato è instabile oltre un certo di iterazioni.

Per ottenere il grafico occorre lanciare gnuplot ed seguire il comando riportato nella slide.

Exercise 8:
Data distribution & ghost-cells / frames

-Distribuition by “leading dimension”:

-FORTRAN: first coordinate

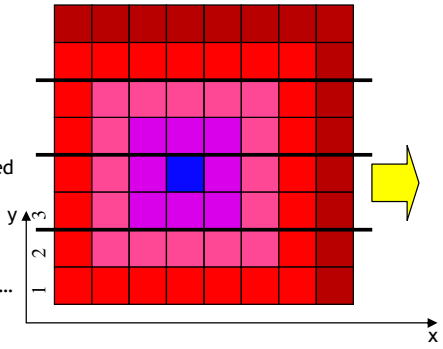
-C: last coordinate

-The important is that the DATA I want to send/recv are CONTIGUOUS in memory. Else I need to copy them to/from a temporary, contiguous, buffer.

MORE TASKS: these are easy for the serial version… do it in parallel too.

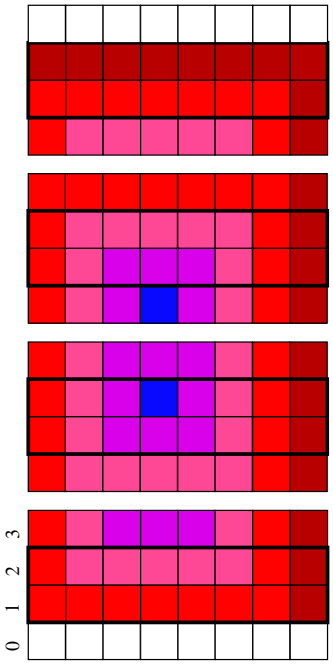
-Evaluate the average over all the system and check it every 10 steps

-Find the global maximum and print its position every 10 steps



THIS IS THE FORTRAN DATA DISTRIBUTION… Why is it so “orizontal”?

Say it without saying “matrix”, “row”, “column”: these are XY!

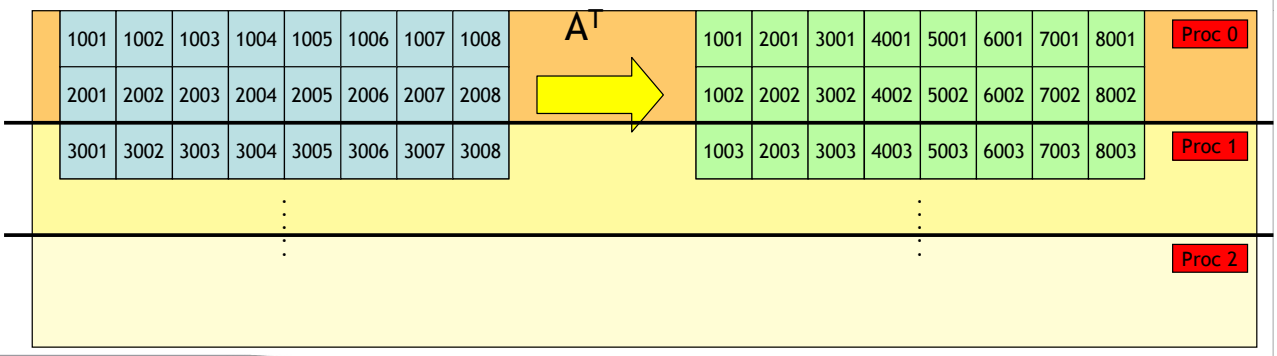


titolino

17ª Scuola Estiva di CALCOLO PARALLELO
1ª edizione 7 - 18 luglio 2008
2ª edizione 8 - 19 settembre 2008

Exercise 9: distributed matrix transposition

- Initialize A, a 8x8 distributed matrix, so that A_{ij} = 1000*i+j
- Both A and B are distributed by lines over 4 processors
- Print out A
- Evaluate: B = A^T
- Print B



CINECA Consorzio Interuniversitario

Roberto Gori

Trasposta di una matrice

Obiettivo

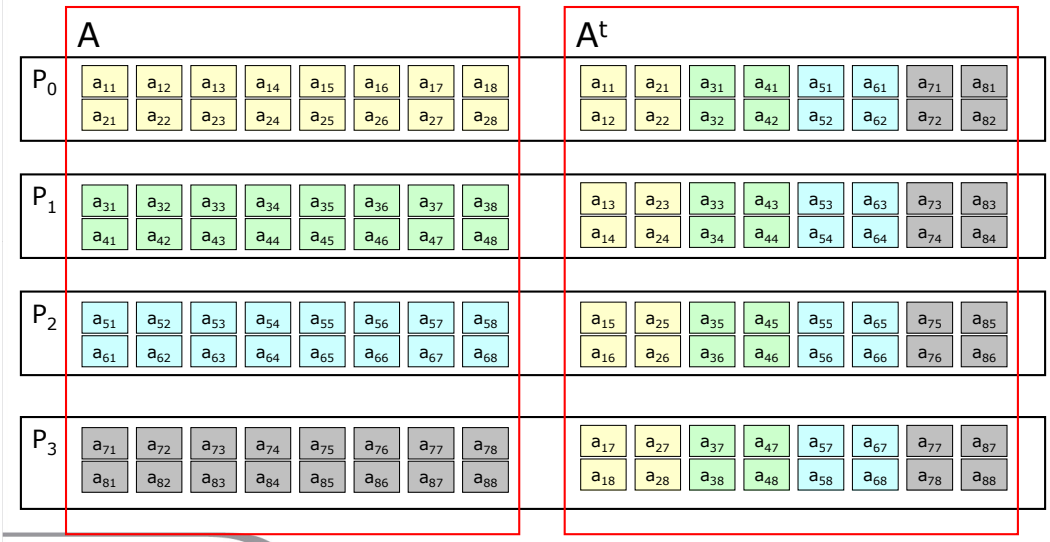
1. Creare e stampare una matrice quadrata A di dimensione NxN distribuita su P processi, inizializzata con valori A_{ij} = 1000*i+j.
2. Calcolare e stampare la matrice B trasposta di A minimizzando l'impiego di memoria e il numero di primitive MPI

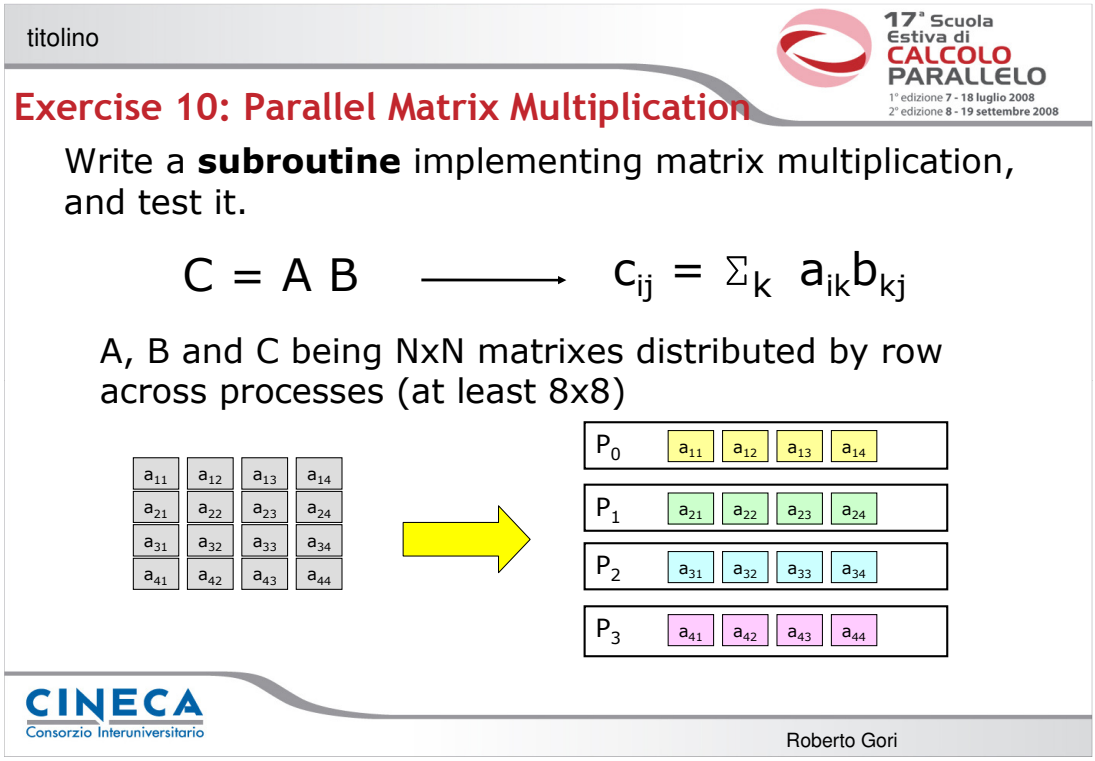
Commenti

È possibile risolvere il problema utilizzando una sola comunicazione tra processi seguita da una semplice operazione locale.

A seconda del linguaggio utilizzato (C o Fortran) occorre scegliere opportunamente le dimensioni locali della matrice.

Exercise 9: distributed matrix transposition





Moltiplicazione di matrici

Obiettivo

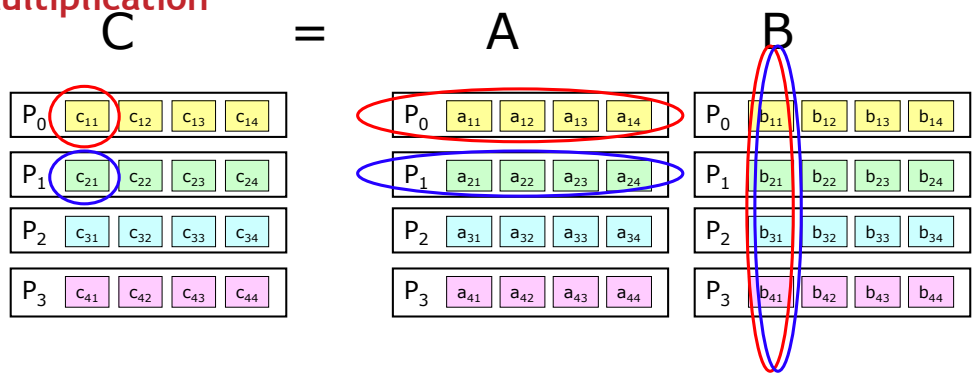
1. Creare e stampare due matrici quadrate A e B di dimensioni NxN distribuite su P processi, inizializzate rispettivamente con i valori A_{ij} = i*j e B_{ij} = 1 / (i*j) .
2. Calcolare e stampare la matrice C prodotto di A e B minimizzando l'impiego di memoria e il numero di primitive MPI

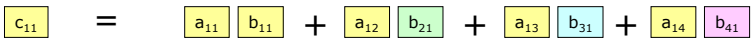
Commenti

È possibile risolvere il problema utilizzando N/P comunicazioni tra processi seguite da semplici operazioni locali.

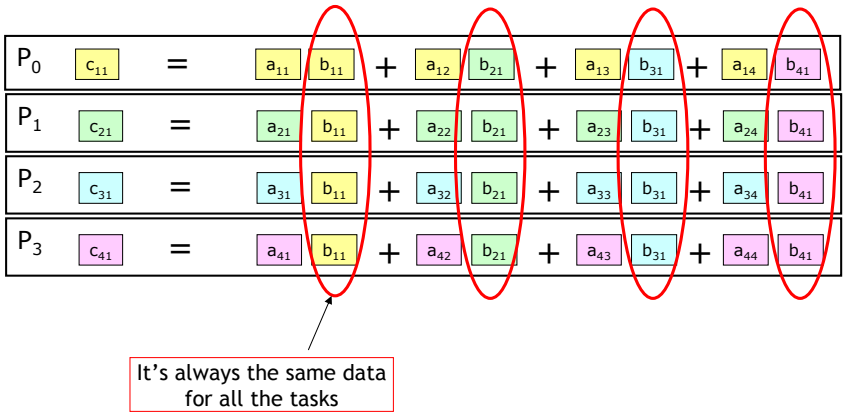
A seconda del linguaggio utilizzato (C o Fortran) occorre scegliere opportunamente le dimensioni locali della matrici.

Exercise 10: Parallel Matrix Multiplication





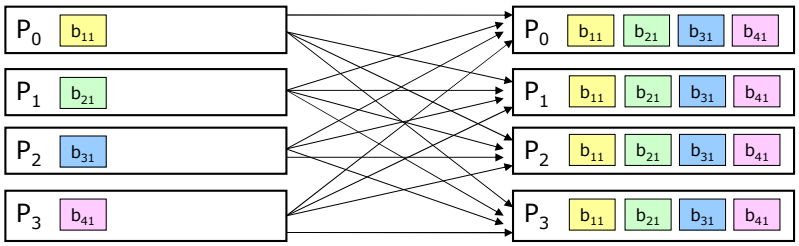
Exercise 10: Parallel Matrix Multiplication



It's always the same data for all the tasks

Exercise 10: Step 1

Perform an All gather, of the first column of blocks



Exercise 10: Step 2, local work

Each processor calculate the first block of the matrix C

P_0 $c_{11} = a_{11} b_{11} + a_{12} b_{21} + a_{13} b_{31} + a_{14} b_{41}$

P_1 $c_{21} = a_{21} b_{11} + a_{22} b_{21} + a_{23} b_{31} + a_{24} b_{41}$

P_2 $c_{31} = a_{31} b_{11} + a_{32} b_{21} + a_{33} b_{31} + a_{34} b_{41}$

P_3 $c_{41} = a_{41} b_{11} + a_{42} b_{21} + a_{43} b_{31} + a_{44} b_{41}$

Exercise 10: Generalize

Repeat Step 1 and Step 2 for each column elements or blocks of matrix C, until matrix C is complete