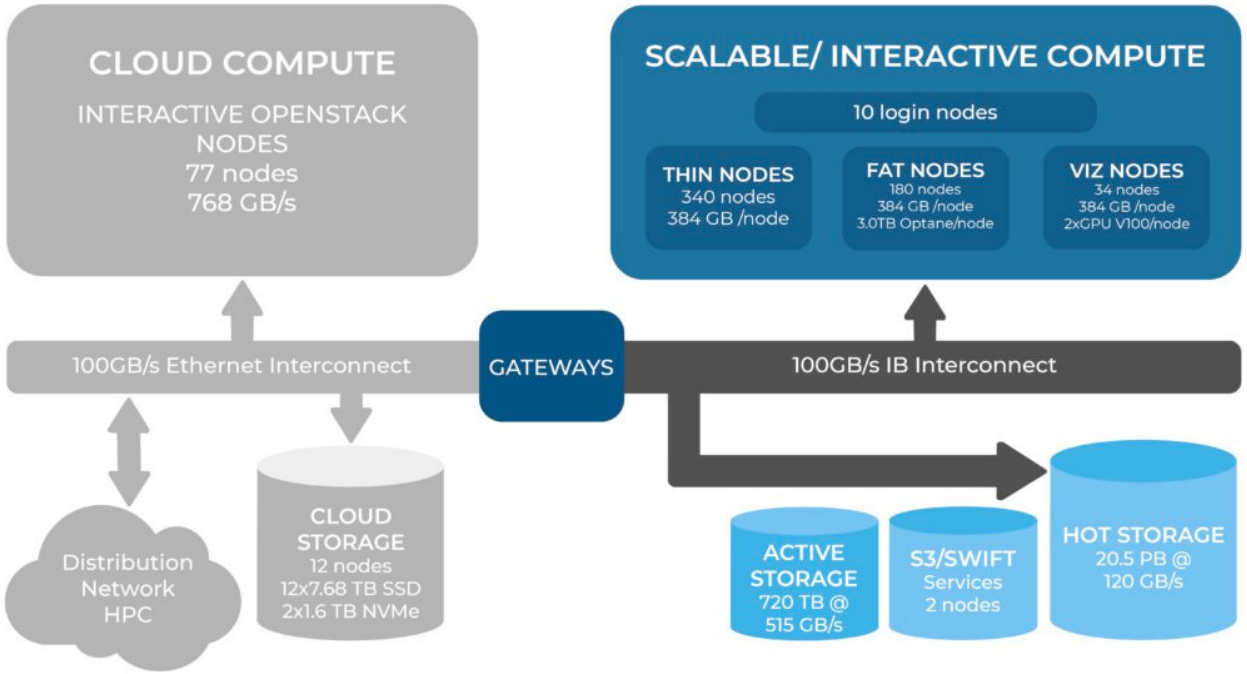
Esercitazione sulla programmazione parallela in MPI
27 Maggio 2024

L'infrastruttura HPC Galileo100



Esempio: Galileo 100, Cineca:

528 computing nodes each 2 CPU Intel CascadeLake 8260, with 24 cores each, 2.4 GHz, 384GB RAM, subdivided in:

- **348 standard nodes ("thin nodes") 480 GB SSD**
- **180 data processing nodes ("fat nodes") 2TB SSD, 3TB Intel Optane**
- **34 GPU nodes with 2x NVIDIA GPU V100 with 100Gbs Infiniband interconnection and 2TB SSD.**

→ 25.344 cores

Accounting

Esercitazioni SOM 2023-24:

- Account: **tra24_IngInfBo** → 9000 ore di calcolo totali

Login su Galileo100:

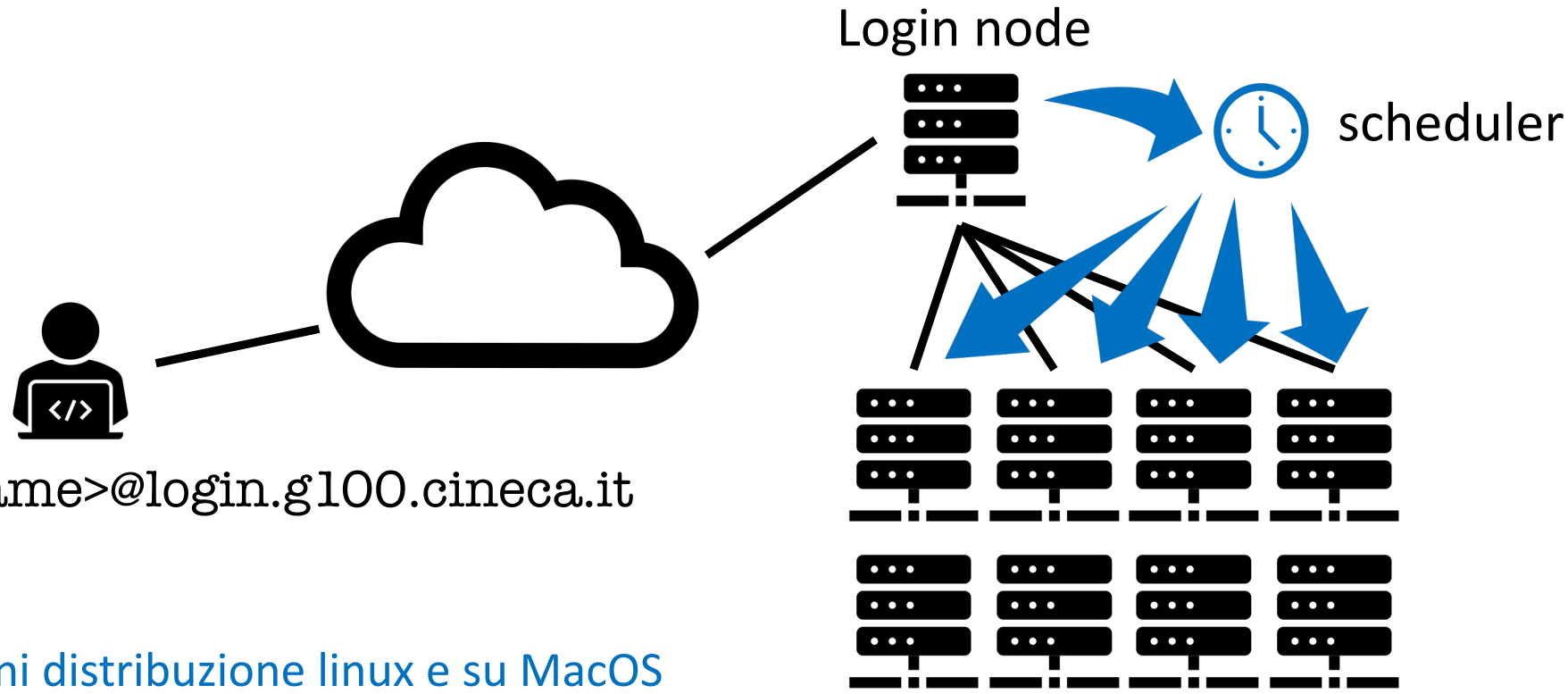
Ogni studente/gruppo utilizzerà lo username personale assegnatagli dal Cineca (con la registrazione sul servizio UserDB).

Autenticazione a 2 fattori.

$ step ssh login ‘<email>' --provisioner cineca-hpc

$ ssh <username>l@login.g100.cineca.it

Accesso all'infrastruttura CINECA



ssh <username>@login.g100.cineca.it

NB:

- ssh è disponibile in ogni distribuzione linux e su MacOS
- in altri OS: putty, filezilla ecc.
- Check the user guide!

https://wiki.u-gov.it/confluence/display/SCAIUS/UG3.3%3A+GALILEO100+UserGuide

Compilazione programmi MPI su GALILEO100

- Una volta loggati siete nella home dello user <username> su GALILEO100. Il sistema operativo è **linux** (Centos 8.3):
  - Con `ls` potete visualizzare i file nella home (50GB per user).
- inviare il sorgente su GALILEO100:
  - da un'altra shell sul vostro computer:

    $ scp /path/to/sommavet.c <username>@login.g100.cineca.it:.
  - ora il file è nella cartella col vostro cognome su GALILEO100; Fare ls per verificarlo
- compilare
  - **module load autoload intelmpi** → carica il compilatore MPI Intel (consigliato)
  - **mpiicc -std=c99 sommavet.c –o sommavet**

Uso di Galileo 100: partizioni

- **Submitting parallel Batch Jobs**

To run parallel batch jobs on GALILEO100 you need to specify the partition and the qos that are described in this user guide.

If you do not specify the partition, your jobs will try to run on the default partition g100_all_serial.

The minimum number of cores you can request for a batch job is 1. The **maximum number of cores** that you can request is 3072 (64 nodes). It is also possible to request a **maximum walltime of 24 hours**. Defaults are as follows:

If you do not specify the walltime (by means of the #SBATCH --time directive), a default value of **30 minutes** will be assumed.

If you do not specify the number of cores (by means of the "SBATCH -n" directive) a default value of **1 core** will be assumed.

If you do not specify the amount of memory (as the value of the "SBATCH --mem" DIRECTIVE), a default value of **7800 MB per core** will be assumed on g100_usr_prod and g100_usr_smem partitions. While on the g100_usr_bmem partition, the default value of memory assumed is **63200 MB per core**.

The maximum memory per node is **375300MB (366.5GB) for thin and viz nodes, about 3TB for fat nodes.** A study on the performances of the thin and fat nodes is ongoing.

On GALILEO100 there are four partitions dedicated to the production:

g100_usr_prod: this partition collects all the thin and persistent memory nodes (memory per node **375300 MB**).

g100_usr_smem: this partition is composed only of thin nodes (memory per node **375300 MB**).

g100_usr_pmem: this partition is composed only of Persistent memory nodes - more info in a following update (memory per node **375300 MB**).

g100_usr_bmem: this partition is composed only of fat nodes (memory per node **3 TB**). It is reserved for jobs that need more than 375300 MB of memory per node and is out of the g100_usr_prod partition.

We encourage you to use the partition more suitable for your job.

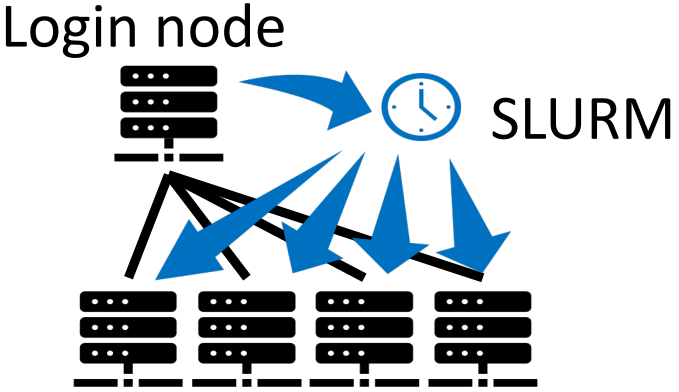
Gestione dei job

Dal **login node**, è possibile mandare in esecuzione i propri programmi paralleli sui nodi a disposizione per il calcolo (**compute node**).
Nelle infrastrutture HPC, la gestione di tutti i programmi per i quali gli utenti richiedono l'esecuzione viene normalmente affidata a uno scheduler (gestore dei job).

Per ogni job J:

- lo scheduler aggiunge la richiesta di esecuzione di J in un sistema di accodamento
- quando le risorse richieste per l'esecuzione di J saranno disponibili, lo scheduler preleverà il job dalla coda per allocarlo sui compute node selezionati → esecuzione

👉 Lo scheduler utilizzato dal Cineca è **SLURM**.



Esecuzione batch: SLURM script

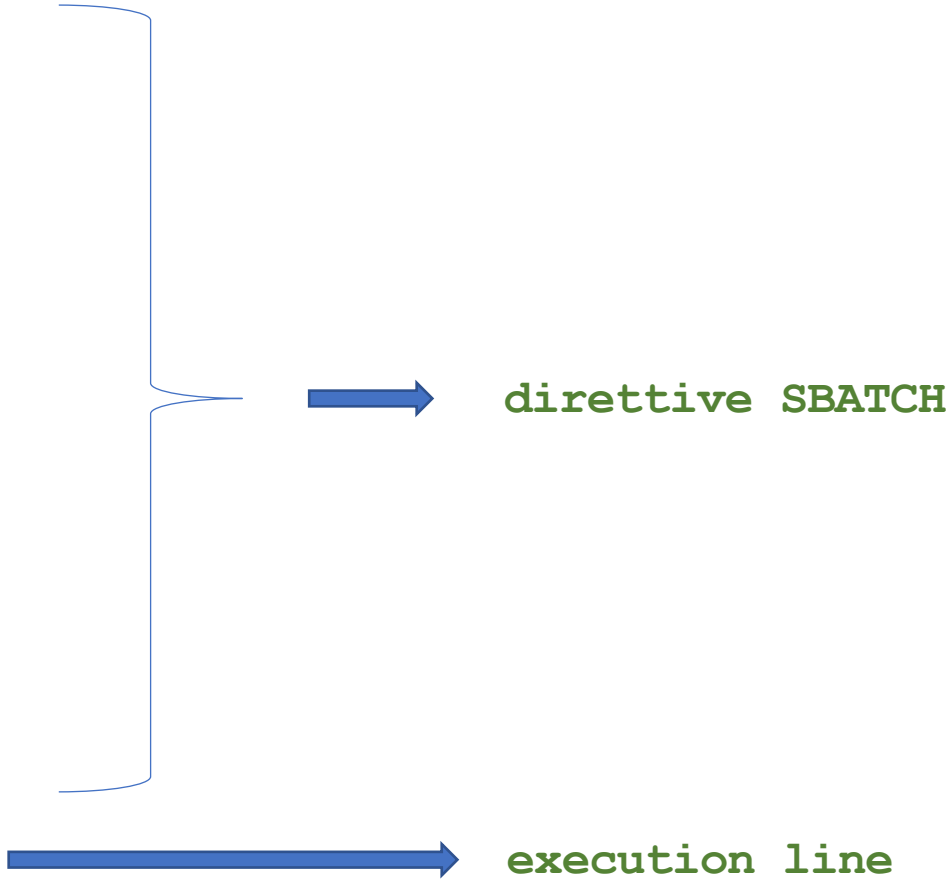
- Dopo la compilazione, non è possibile lanciare direttamente l'eseguibile.
- Creare uno SLURM script: nano launcher.sh

```
#!/bin/bash

# direttive SBATCH
…

# env. variables and modules
…

# execution line
…
```

SLURM job script

Esempio Job seriale:

```
#!/bin/bash
#SBATCH --job-name=myname
#SBATCH --output=job.out
#SBATCH --error=job.err
#SBATCH --mail-type=ALL
#SBATCH --mail-user=user@email.com
#SBATCH --time=00:30:00
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=1
#SBATCH --mem=10GB
#SBATCH --account=<my_account>
echo "I'm working on G100!"
```



Direttive SLURM

#SBATCH --job-name=myname, -J myname → indica il nome del job

#SBATCH --output=job.out, -o job.out → specifica il file dove sarà ridiretto lo standard output (default=slurm-<Pid>)

#SBATCH --error=job.err, -e job.err → specifica il file dove sarà ridiretto lo standard error (default=slurm-<Pid>)

#SBATCH --mail-type=ALL [opzionale] →richiede e-mail notification. Un' e-mail verrà inviata per ogni evento accaduto al job

#SBATCH --mail-user=user@email.com → Specifica l'indirizzo e-mail a cui inviare le notifiche (v.sopra)

#SBATCH --time=00:30:00, -t 00:30:00 →massima durata del job.

… Direttive SLURM

- **#SBATCH --nodes=1, -N 1**
- **#SBATCH --ntasks-per-node=36**
- **#SBATCH --mem=10GB**

specificano le risorse richieste per l'esecuzione del job:

- **nodes**: numero di compute nodes («chunks»)
- **ntasks-per-node:** numero di processi per nodo
- **mem**: memoria allocata per ogni nodo (default=3000MB, max=118000 MB)

SLURM: comandi

- La submission di un job viene richiesta con l'esecuzione del relativo SLURM Script, tramite il comando slurm **`sbatch`**:

```
$ sbatch <nome_script>
```

→ la richiesta di esecuzione del job viene presa in carico dallo scheduler SLURM, che ne avvierà l'esecuzione quando tutte le risorse saranno disponibili (tenendo conto della sua politica..)

SLURM: comandi

- **squeue** consente di verificare lo stato dei job schedulati :

```
$ squeue –u <username>
```

→viene fornita la lista di tutti i job dell'utente con il loro stato corrente (idle, running, closing...)

- **scontrol** permette di ottenere informazioni dettagliate su un job :

```
$ scontrol show job <job_id>
```

- **scancel** elimina un job dalla coda (se non ancora running), oppure lo termina forzatamente:

```
$ scancel <job_id>
```

SLURM job script

Esempio Job parallelo:

```
#!/bin/bash
#SBATCH --job-name=myname
#SBATCH --output=job.out
#SBATCH --error=job.err
#SBATCH --mail-type=ALL
#SBATCH --mail-user=user@email.com
#SBATCH --time=00:30:00
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=48
#SBATCH --mem=10GB
#SBATCH --account=<my_account>
srun echo "I'm working on G100!"
```

Il comando SLURM `srun` richiede l'esecuzione dell'eseguibile/comando seguente (echo) alle condizioni indicate dalle direttive SLURM

Lanciare programma MPI su GALILEO100

File launcher.sh

```
#!/bin/bash
#direttive SBATCH
#SBATCH --job-name=myname
#SBATCH --nodes 2
#SBATCH --ntasks-per-node=5
#SBATCH --output job.out
#SBATCH --error job.err
#SBATCH --time 00:05:00
#SBATCH --account=tra24_IngInfBo
#SBATCH --partition=g100_usr_prod

# env. variables and modules
module load autoload intelmpi
# execution line
srun ./trapezi
```

- **--account** nome dell'account
- **--partition** nome della partizione di GALILEO100 che vogliamo usare
- eseguire con: sbatch ./launcher.sh
- controllare stato in coda:

  squeue –u <username>

indispensabile per eseguire (oltre che per compilare) un programma MPI

Esecuzione in modalità interattiva

Oltre alla modalità batch, è possibile lanciare un job in modalità interattiva.
Non occorre uno SLURM script, ma occorre usare il comando **salloc** per allocare le risorse necessarie

```
[aciampol@login01 anna]$ salloc -N 1 --ntasks-per-node=2 -A tra24_IngInfBO -p
g100_usr_prod
salloc: Granted job allocation 1179030
salloc: Waiting for resource configuration
salloc: Nodes r500n001 are ready for job
[a08trb01@login01 anna]$ srun ./helloworld
        Hello, I am task 1 of 2 running on r500n001
        Hello, I am task 0 of 2 running on r500n001
[a08trb01@login01 anna]$ exit
salloc: Relinquishing job allocation 1179030
```

la shell non risponde finchè SLURM non ha trovato le risorse richieste

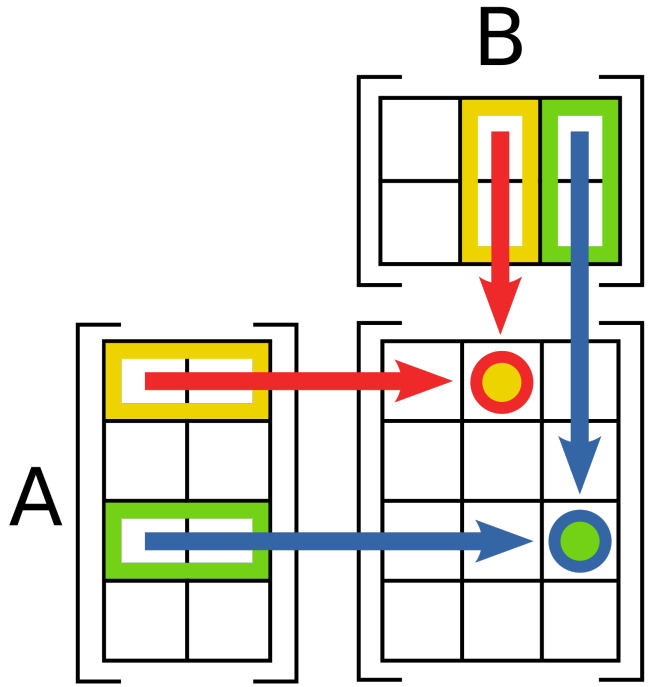
sembrerebbe un esecuzione sul nodo di login ma **MPI_Get_processor_name** ci rivela che il nodo di computazione è proprio r500n001

termino la sessione interattiva e rilascio le risorse

Esercizio

- Realizzare un programma parallelo MPI che:
  - date due matrici $n \times n$ A e B,
  - ne calcoli il prodotto $C = A * B$
  - ovvero per ogni elemento:

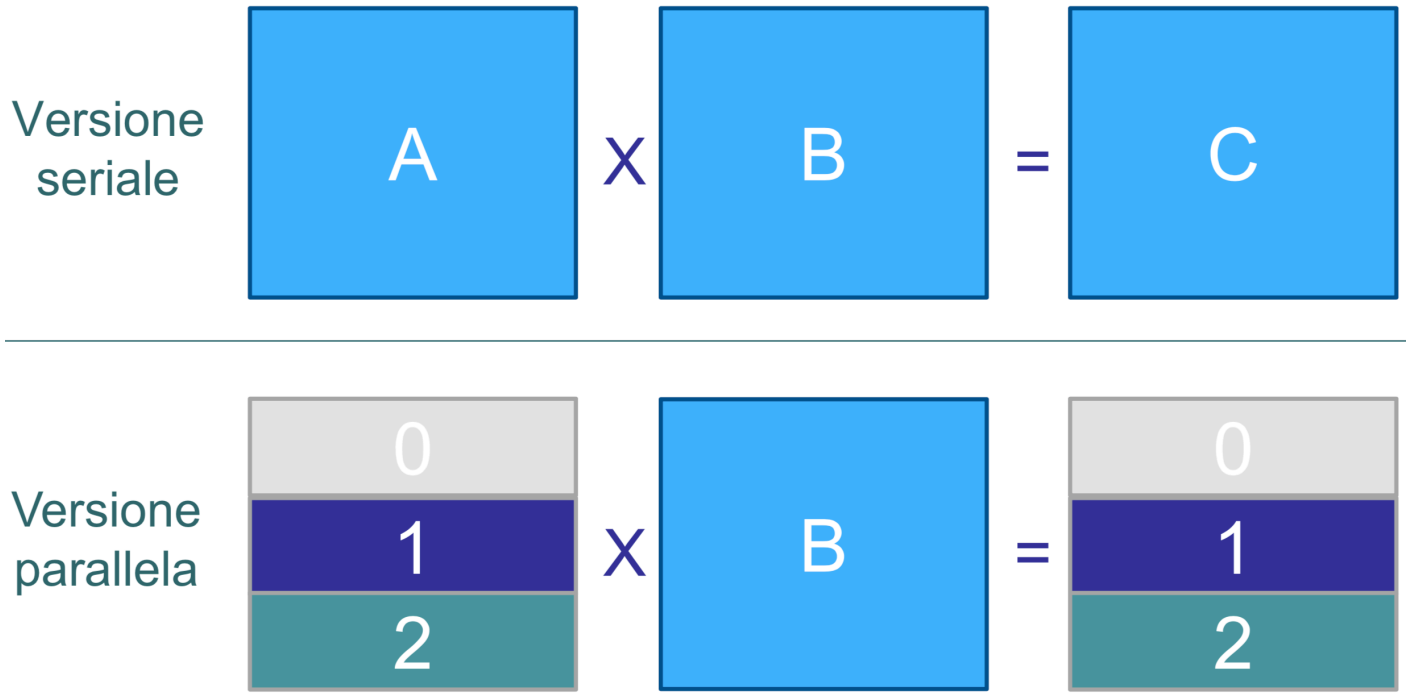
$$C_{i,j} = \sum_{k=1}^{n} A_{i,k} B_{k,j}$$



- Supponiamo per semplicità che n sia multiplo del numero di processi

Simple domain decomposition

In C l'allocazione delle matrici avviene per righe



Tutti i processi ricevono:

- la matrice B
- un set di righe contigue della matrice A

Tutti i nodi calcolano un set di righe contigue della matrice C

Passi

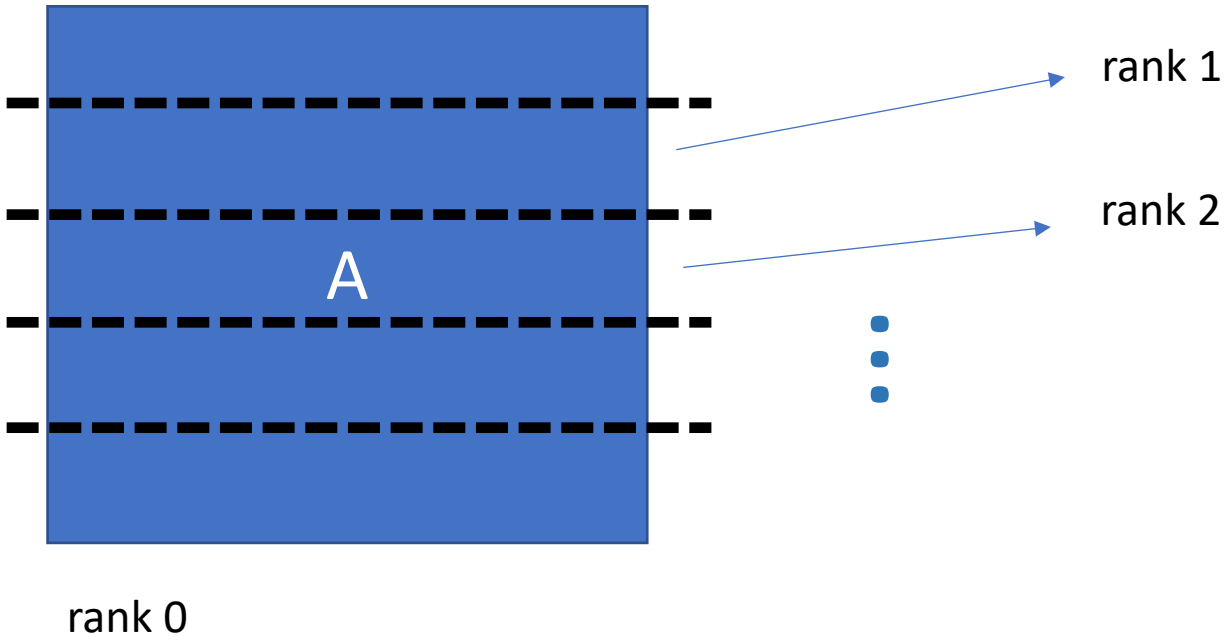
- rank 0 genera randomicamente le matrici A e B
- A distribuita per blocchi di righe
- B inviata interamente a tutti i nodi
- ogni nodo usa B e la sua porzione di A per calcolare la sua porzione di C:

$$C_{i,j} = \sum_{k=1}^{n} A_{i,k} B_{k,j}$$

- rank 0 raccoglie i risultati in un unica matrice C

Versione base

- Usando primitive di comunicazione **point-to-point**



Versione base - passi

- rank 0 genera randomicamente le matrici A e B
- rank 0 divide A in tanti blocchi di righe quanti sono i nodi (rank0 incluso?)
  - rank 0 manda un blocco ad ogni nodo
  - gli altri nodi ricevono da rank 0
- rank 0 invia B a tutti i nodi
  - gli altri nodi ricevono una copia di B
- ogni nodo computa: $C_{i,j} = \sum_{k=1}^{n} A_{i,k} B_{k,j}$
- tutti i nodi (tranne rank 0) mandano a rank0 un pezzetto di C
  - rank0 riceve da ogni altro nodo un pezzetto di C

Versione ottimizzata

- Risolvere il problema usando primitive di comunicazione **collettive**