

Esercitazione 7 – OpenMP

20 Dicembre 2024

Sviluppo programmi OpenMP su Galileo100

Consigli:

- inviare il sorgente su Galileo100:

```
scp /path/to/sommavet.c <user>@login.g100.cineca.it:.
```

→ ora il file è nella vostra home su galileo100! Fare **ls** per verificarlo

- compilare:

```
gcc -fopenmp sommavetOMP.c -o sommavetOMP
```

Esecuzione batch

- Creare uno SLURM script: launcherOMP

```
#!/bin/bash

# direttive SBATCH
...

# execution line
srun ./sommavetOMP <n>
```

Esempio SLURM launcher

launcherOMP

```
#!/bin/bash
```

```
#direttive SBATCH
```

```
#SBATCH --account=tra24_IngInfBo
```

```
#SBATCH --partition=g100_usr_prod
```

```
#SBATCH -t 00:05:00
```

```
#SBATCH --nodes=1
```

```
#SBATCH --ntasks-per-node=1
```

```
#SBATCH -c 48 # CPU cores (OpenMP threads per task)
```

```
#SBATCH -o job<cognome>.out
```

```
#SBATCH -e job<cognome>.err
```

```
srun <nome eseguibile> <eventuali argomenti>
```

Solite direttive (v.es. 6) +:

- numero di nodi (OMP->1)
- numero di Task per nodo (OMP -> 1)
- numero di core (max 48)
- eseguirlo con:

sbatch

./launcherOMP

controllare stato in coda:

squeue -u <username>

Esempio: somma vettori in OpenMP

```
#include <stdio.h>
#include <stdlib.h>
#include <omp.h>
#define DIM 12

int main(int argc, char* argv[])
{
    double start, end;
    int size, my_rank;
    int A[DIM], B[DIM], C[DIM], i ;
    if (argc!=2)
    {
        printf("sintassi sbagliata -- %s n_proc", argv[0]);
        exit(1);
    }
    size=atoi(argv[1]); // il numero di processi viene passato come argomento
    if(size > DIM)
    {
        printf("il numero di processi %d è maggiore della dimensione %d.\n",size, DIM);
        exit(1);
    }
    // inizializzazione vettori:
    srand((unsigned int)time(NULL));
    for(i=0;i<DIM;i++){
        A[i]=rand()%100;
        B[i]=rand()%100;
    }
}
```

```
start = omp_get_wtime(); //campionamento del tempo di inizio

# pragma omp parallel num_threads(size) shared(A,B,C, size) private(i,my_rank)
{
    my_rank=omp_get_thread_num();
    printf("thread %d di %d: inizio il calcolo...\n", my_rank, size);
    # pragma omp for // il lavoro del for viene suddiviso tra i thread del team
    for(i=0; i<DIM; i++)
        C[i]=A[i]+B[i];
}

end = omp_get_wtime();
printf("[Master] Risultato C=A+B:\n");
for(i=0; i<DIM; i++)
    printf("\t%d\n", C[i]);
printf("Tempo di esecuzione: %lf\n", end-start);
return EXIT_SUCCESS;
}
```

SLURM launcher

launcherOMP

```
#!/bin/bash
                                tra24_IngInfB2
#SBATCH --account=tra24_IngInfBo
#SBATCH --partition=g100_usr_prod
#SBATCH -t 00:05:00
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=1
#SBATCH -c 48
#SBATCH -o job.out
#SBATCH -e job.err

srun ./sommavet 48
```

Solite direttive (v.es. 5) +:

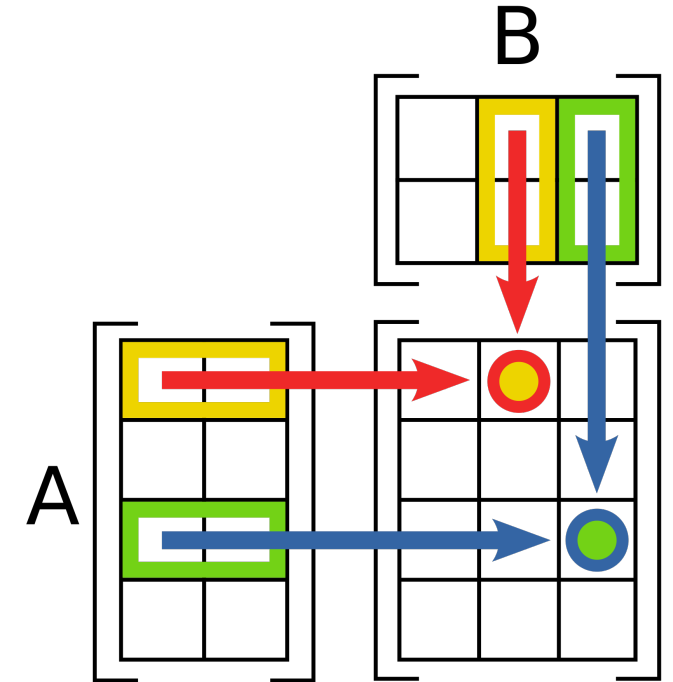
- numero di nodi (OMP->1)
- numero di Task per nodo (OMP -> 1)
- numero di core (max 48)
- eseguirlo con:
sbatch ./launcherOMP
- controllare stato in coda:
squeue -u <username>

Esercizio: PRODOTTO DI MATRICI QUADRATE

Realizzare un programma parallelo OMP che:

- date due matrici $DIM \times DIM$ A e B,
- ne calcoli il prodotto $C=A*B$
- ovvero per ogni elemento:


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



👉 Realizzare una soluzione **PARAMETRICA** in:

- **DIM** (-> allocazione dinamica delle matrici)
- **N** (numero dei thread)

Impostazione

- le matrici A e B sono i dati a partire dai quali verrà eseguita l'elaborazione.
- la matrice C è il risultato.
- **OMP**: possibilità di condividere i dati tra thread:
 `int A[DIM][DIM], B [DIM][DIM], C [DIM][DIM] → variabili shared.`
- Necessità di definire **sezioni critiche**?

Impostazione

HP: Ipotizziamo che la dimensione DIM delle matrici sia **multiplo intero del numero N dei threads**.

👉 Quale suddivisione del workload adottare nel **# pragma omp..for?**

- Se DIM non fosse multiplo intero di N, quale tipo di scheduling converrebbe adottare?

Schedule

es: dim 19, 4 thread → dim non è multiplo intero di N

- con lo scheduling di default: (1-4), (5-8), (9-12), (13-19)
→ l'ultimo processo fa 7 iterazioni
- `schedule(static, 1)`: (1,5,9,13,17) , (2,6,10,14,18) ,(3,7,11,15,19)
,(4,8,12,16)
→ il massimo lavoro è 5 iterazioni

Proposta di progetto per l'esame: riduzione di una matrice

Sia data una matrice quadrata $A[N][N]$, contenente N^2 valori reali appartenenti all'intervallo $[0,1]$.

Si supponga che la matrice A sia molto grande ($N \geq 2000$).

Realizzare un programma che calcoli una matrice $R[N/2][N/2]$ di valori reali ottenuta come segue.

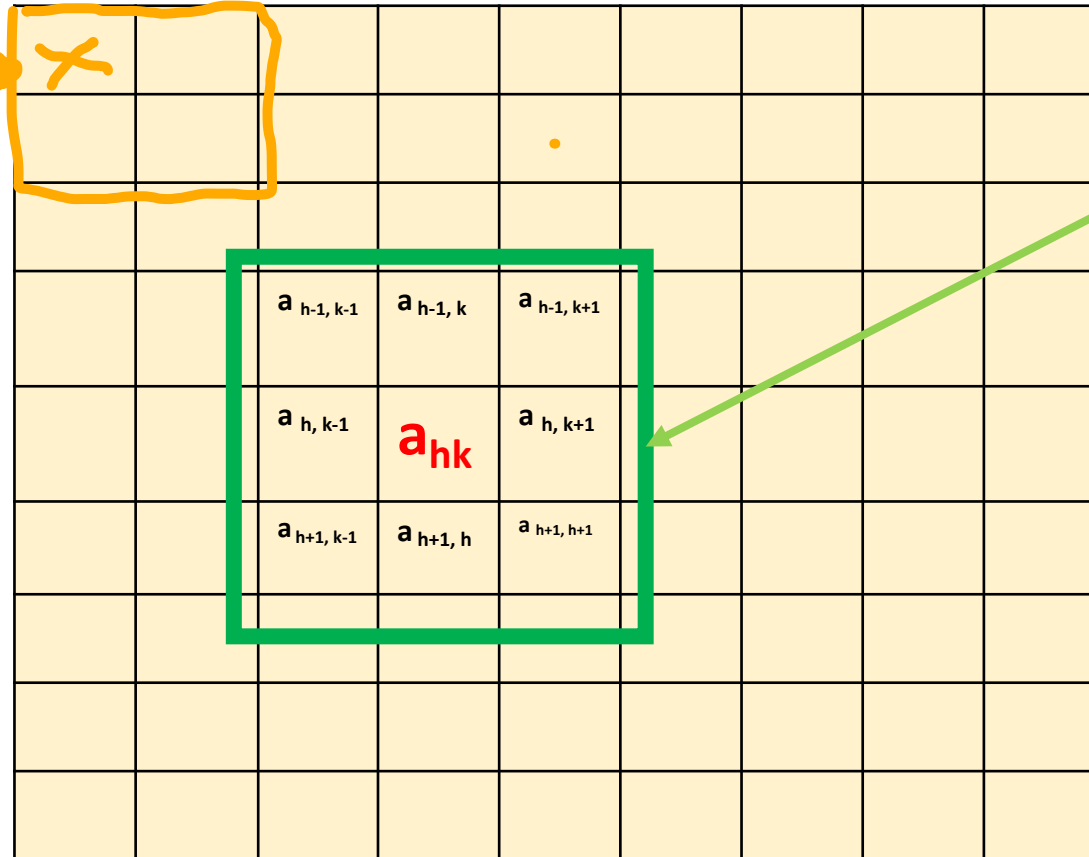
per ogni $i \in \left[0, \frac{N}{2} - 1\right]$, $j \in \left[0, \frac{N}{2} - 1\right]$, $R_{ij} = \text{Media_Intorno}(a_{2i, 2j})$

dove:

$\text{Media_Intorno}(a_{hk})$ è il valore della **media aritmetica** di tutti i valori contenuti nell'**intorno** di a_{hk}

Definiamo «**intorno**» di a_{hk} la sottomatrice di A di dimensione 3×3 che ha a_{hk} nella posizione centrale.

A



Intorno di a_{hk}

Si assume che la matrice dell'intorno degli elementi di confine (bordo) sia una 2×2

$\frac{1}{4}$

$$Media_Intorno(a_{hk}) = \frac{1}{9} \cdot \sum_{i=k-1, j=h-1}^{i=k+1, j=h+1} a_{ij}$$

Proposta di progetto per l'esame

Da fare:

1] Realizzare una soluzione parallela MPI nella quale:

- ogni nodo calcoli una diversa porzione della matrice risultato R

Pertanto:

- ogni nodo calcolerà gli elementi di R della porzione assegnatagli utilizzando i corrispondenti elementi della matrice dei dati A;
- a questo scopo la matrice A verrà suddivisa tra tutti i nodi

Al termine, il nodo «master» aggregnerà i risultati prodotti da tutti i nodi nella matrice risultato.

[NB: decidere come gestire il calcolo degli elementi sul confine di ogni porzione]

2] Realizzare una soluzione parallela OMP che calcoli la matrice R nella quale:

- le matrici A e R siano condivise tra tutti i nodi;
- ogni nodo calcolerà gli elementi di R della porzione assegnatagli utilizzando i corrispondenti elementi della matrice dei dati A;

3] Svolgere l'analisi delle prestazioni mediante scalabilità strong e weak di entrambe le soluzioni.

👉 Entrambe le soluzioni dovranno essere **PARAMETRICHE** in:

- N (-> allocazione dinamica delle matrici)
- p (numero dei thread/nodi)

Inoltre, dovrà essere sempre misurato il **tempo di esecuzione**.

Progetto: Analisi delle prestazioni

Dopo aver risolto lo stesso problema (calcolo della matrice R) in due modi (punti 1 e 2):

- MPI
- OpenMP

Misuriamo le prestazioni di entrambe attraverso l'analisi della scalabilità **strong** e **weak** di ognuna delle due soluzioni.

Scalabilità strong & weak

1. **Scalabilità strong:** mantenendo la dimensione delle matrici costante, si valuta l'efficienza al crescere del numero dei nodi. (V. legge Amdahl).

In questo caso:

- 👉 il lavoro totale da eseguire è costante
- 👉 il lavoro del singolo nodo diminuisce al crescere del numero dei nodi

2. **Scalabilità weak:** mantenendo costante il carico di lavoro per singolo nodo, si valuta l'efficienza al crescere del numero dei nodi con lo stesso fattore (V. legge Gustafson).

NB: selezionare un carico di lavoro per nodo che garantisca uno speedup soddisfacente (v. studio scalabilità strong)

In questo caso:

- 👉 la **dimensione del problema** aumenta in proporzione al numero **p** di nodi utilizzati.

Indicazioni

Ognuna delle 2 soluzioni dovrà essere parametrica in N e P.

Pianificare le prove in modo **batch**, facendo variare N e/o P:

→ automatizzare i test, ad es. realizzando launcher script ciclici. Es:

```
#!/bin/bash

#SBATCH --account=tra24_IngInfBo
#SBATCH --partition=g100_usr_prod
#SBATCH -t 00:05:00
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=1 # Run a single task per node, more explicit than '-n 1'
#SBATCH -c 24                # number of CPU cores i.e. OpenMP threads per task
#SBATCH -o job.out
#SBATCH -e job.err

for I in 12 24 48; do
    echo "Launching calculator $I"
    srun calculator $I
done
```

→ I risultati dovranno essere raccolti in **tabelle** e i relativi **speedup** dovranno essere rappresentati in **grafici**.

Esempi tabelle:

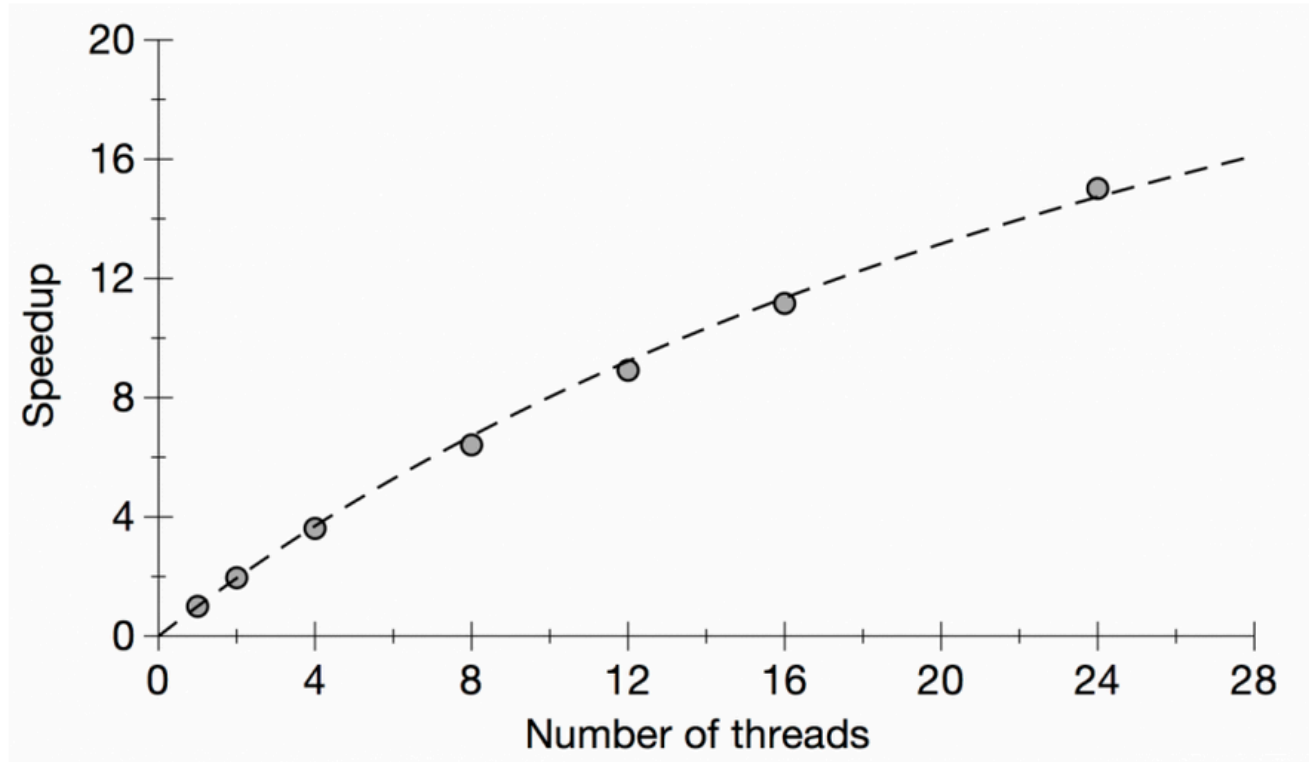
Strong scalability

DIM	threads	time
2000	1	3.932 sec
2000	2	2.006 sec
2000	4	1.088 sec
2000	8	0.613 sec
2000	12	0.441 sec
2000	16	0.352 sec
2000	24	0.262 sec

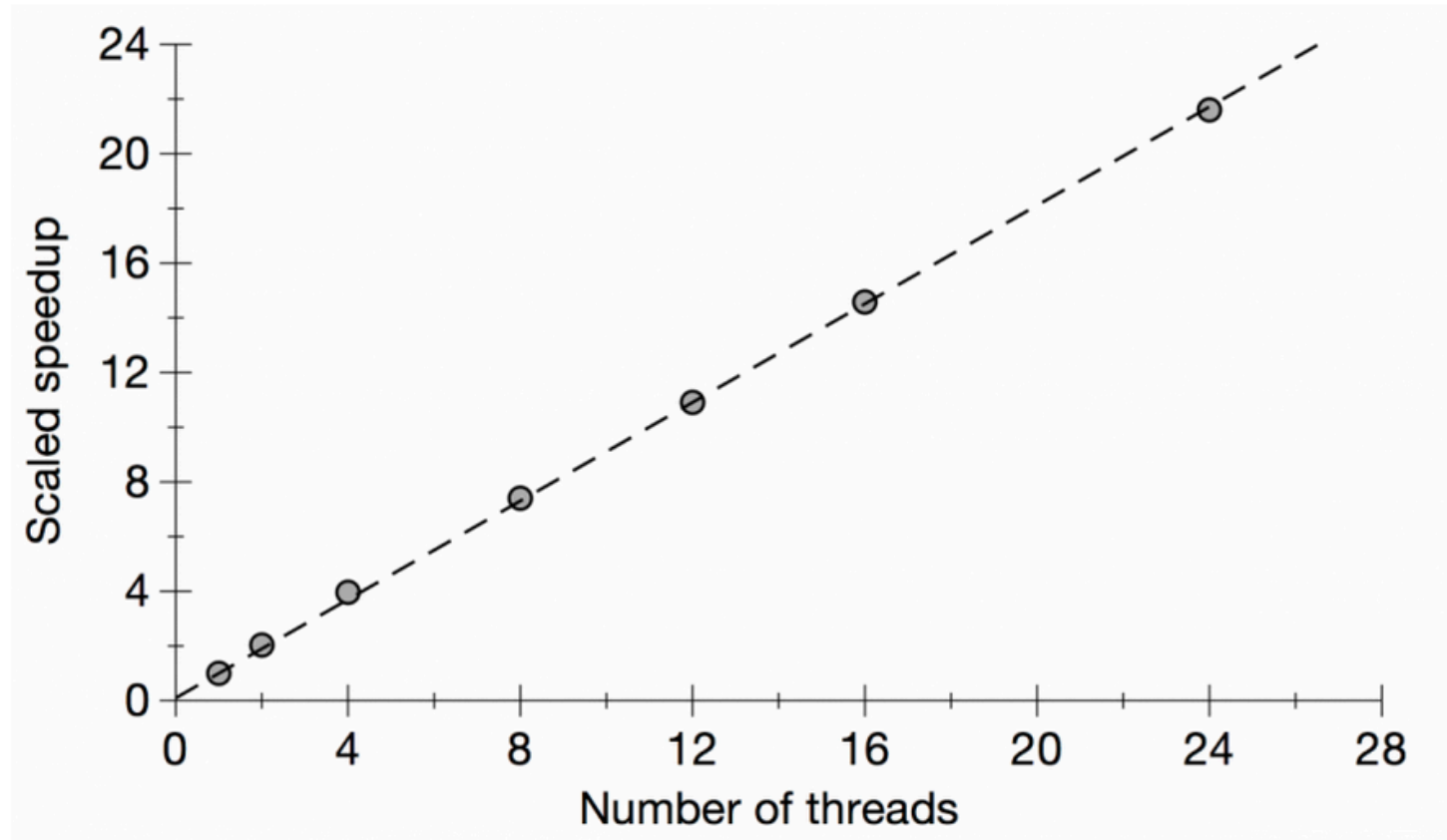
Weak Scalability

DIM	threads	time
10000	1	3.940 sec
20000	2	3.874 sec
40000	4	3.977 sec
80000	8	4.258 sec
120000	12	4.335 sec
160000	16	4.324 sec
240000	24	4.378 sec

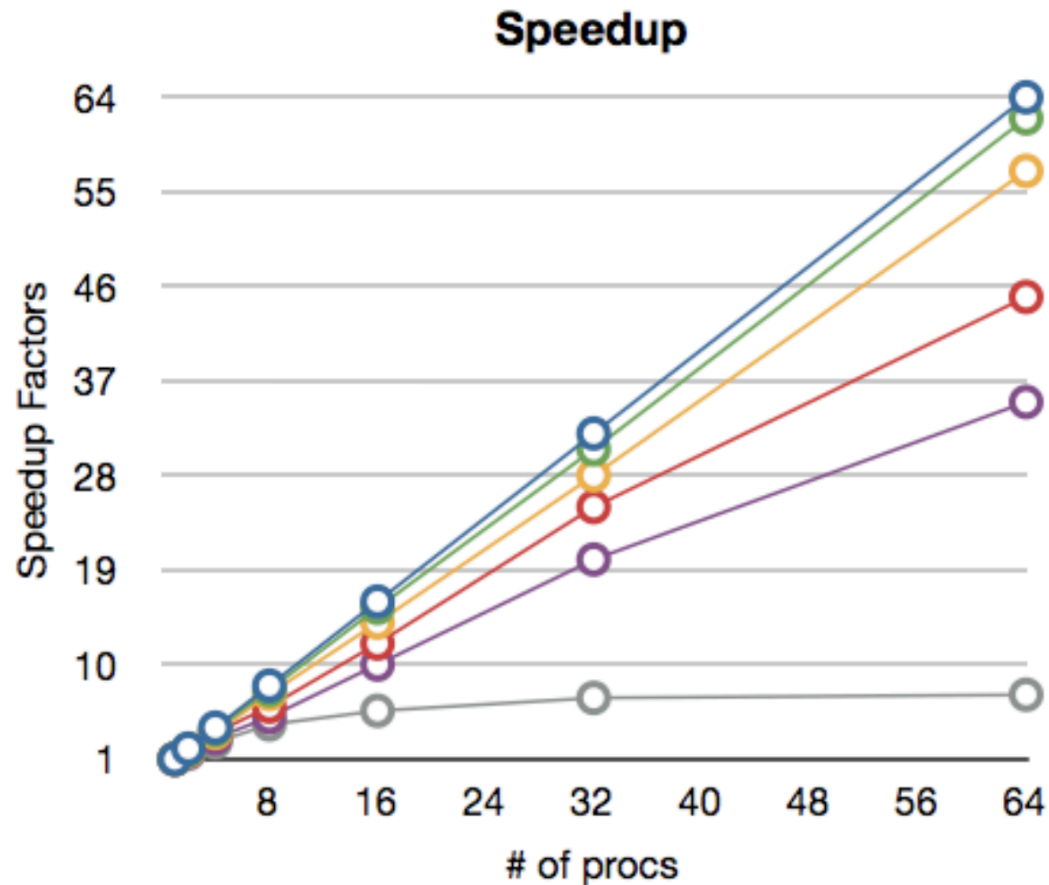
Esempio grafico strong scalability



Esempio grafico weak scalability



Grafici di sintesi



I risultati forniti da ogni soluzione dovranno essere aggregati in 2 grafici di sintesi:

- scalabilità **strong**
- scalabilità **weak**

Incentivo per l'esame

Ogni studente che **vorrà** portare a termine l'attività di **sviluppo e testing** delle soluzioni parallele al problema dato **avrà la possibilità di presentarla e discuterla in sede d'esame orale.**

👉 La valutazione di questa presentazione/discussione potrà (se positiva) fare media con le valutazioni degli altri 2 quesiti «standard» posti nella prova orale, e portare quindi un incremento alla valutazione complessiva della prova orale.