Alma Mater Studiorum - Università di Bologna

# Sistemi Operativi M
# Progetto HPC

8 Luglio 2024

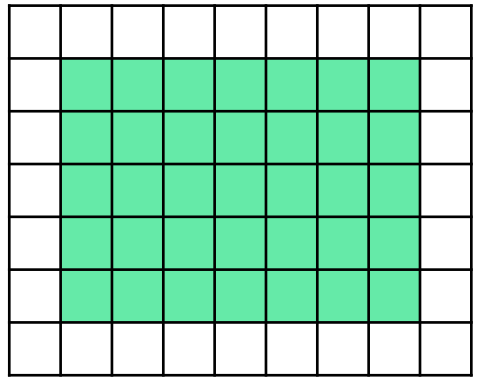
Luca Cimino

# Analisi del problema

- Algoritmo ***embarassingly parallel,*** il calcolo di un elemento di B è indipendente dal calcolo degli altri
  - Dati "condivisi" acceduti solamente in lettura
  - Complessità proporzionale a N

```
// Calcolo di un elemento di B
sum = 0;
for(k = -1; k <= 1; k++) {
    for(h = -1; h <= 1; h++) {
        sum += rows_A[(i+k)*N + j+h];
    }
}

if(sum / 9 < THRESHOLD)
    rows_B[((i-1) * (N-2)) + (j-1)] = 0;
else
    rows_B[((i-1) * (N-2)) + (j-1)] = 1;
```

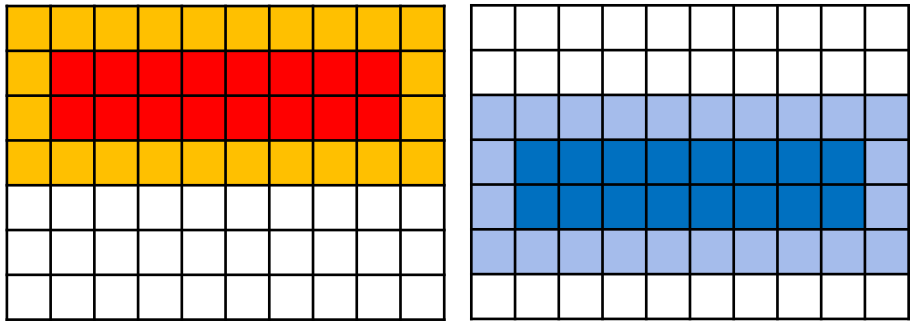
- Data la matrice A di dimensione NxN, la **matrice B** finale ha dimensione (N-2) x (N-2)
  - Gli elementi sul confine sono esclusi dal calcolo



# Soluzione MPI

- Ogni processo esegue il calcolo di **`rows_per_thread = (int) (N-2) / num_processi`** righe della matrice B
  - In caso `(N-2)` non sia multiplo di `num_proc` le X righe in eccesso sono assegnate ai primi X processi
- Ogni processo riceve le `rows_per_thread` righe che deve calcolare + 2 (una sopra e una sotto)
- Il processo master invia le sezioni della matrice A con **send sincrone punto-punto**.

```
for(int dest = 0; dest < size; dest++) {
        MPI_Send(A + (row_count * N), N * (rows_per_thread + 2),
                 MPI_DOUBLE, dest, 0, MPI_COMM_WORLD);
        row_count += rows_per_thread; }
```



- Il thread master raccoglie le sezioni della matrice B calcolate dai processi attraverso la funzione collettiva **`MPI_Gatherv`**.

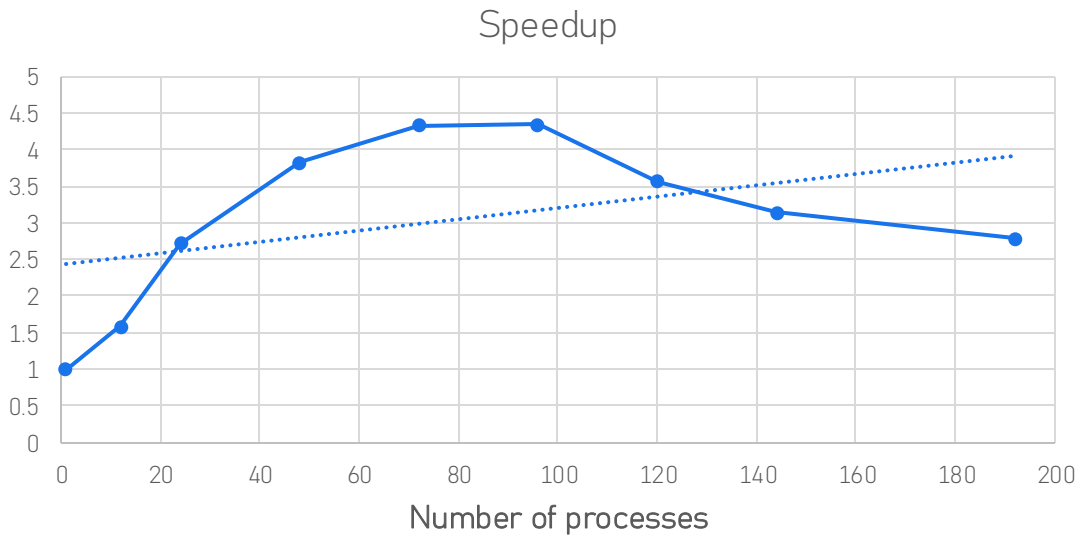
```
if(my_rank == 0)
    MPI_Gatherv(rows_B, (N-2) * (rows_per_thread), MPI_INT, B, gather_rows, displacements, MPI_INT, 0, MPI_COMM_WORLD);
else
    MPI_Gatherv(rows_B, (N-2) * (rows_per_thread), MPI_INT, NULL, NULL, NULL, MPI_INT, 0, MPI_COMM_WORLD);
```

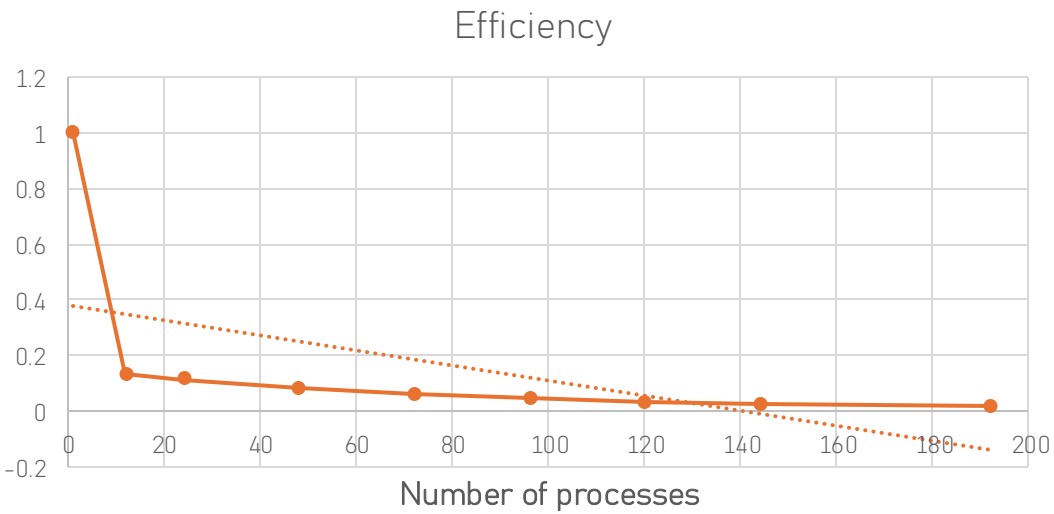
# MPI - Strong scalability

| N | Nodi | Task per nodo | Totale processi | Time (sec) | Speedup | Efficienza |
|---|---|---|---|---|---|---|
| 20000 | 1 | 1 | 1 | 14.567985 | 1 | 1 |
| 20000 | 1 | 12 | 12 | 9.212219 | 1.581 | 0.131 |
| 20000 | 1 | 24 | 24 | 5.360299 | 2.717 | 0.113 |
| 20000 | 1 | 48 | 48 | 3.813544 | 3.820 | 0.079 |
| 20000 | 2 | 36 | 72 | 3.367074 | 4.326 | 0.060 |
| 20000 | 2 | 48 | 96 | 3.350235 | 4.348 | 0.045 |
| 20000 | 3 | 40 | 120 | 4.081067 | 3.569 | 0.029 |
| 20000 | 3 | 48 | 144 | 4.643627 | 3.137 | 0.021 |
| 20000 | 4 | 48 | 192 | 5.238749 | 2.780 | 0.014 |

$$S = \frac{T_{seq}}{T_{par}} \quad \text{dove} \quad T_{par} = \frac{T_{seq}}{p} + T_{overhead}$$

$$E = \frac{S}{p}$$



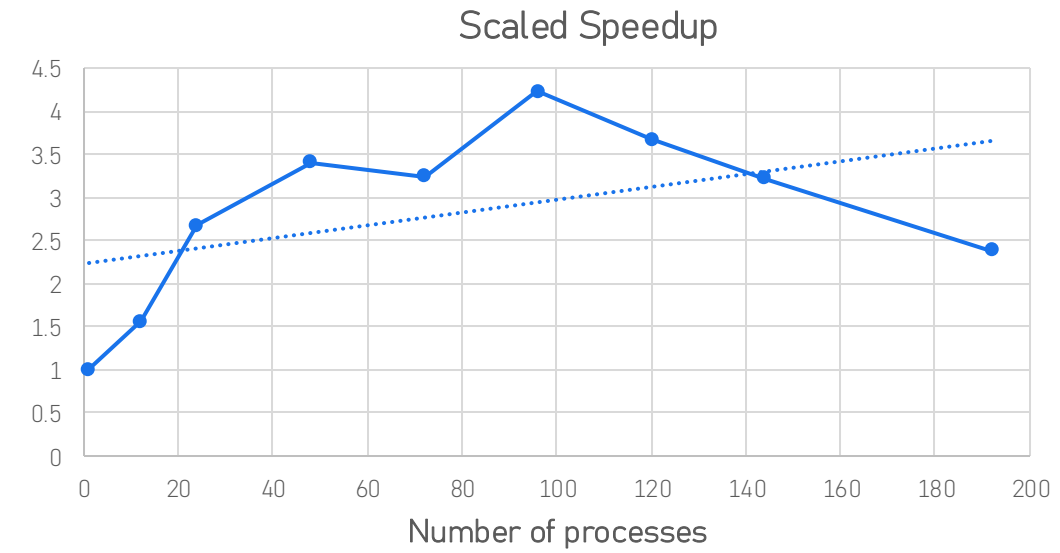



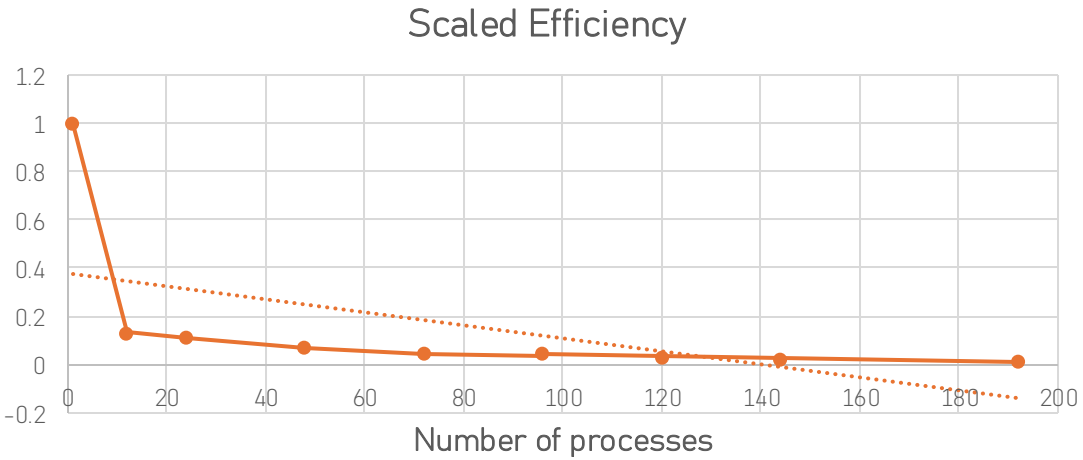
# MPI - Weak scalability

| N | Nodi | Task per nodo | Totale processi | Time (sec) | Speedup scalato | Efficienza scalata |
|---|---|---|---|---|---|---|
| 3000 | 1 | 1 | 1 | 0.328678 | 1 | 1 |
| 10392 | 1 | 12 | 12 | 2.517621 | 1.566 | 0.130 |
| 14696 | 1 | 24 | 24 | 2.94409 | 2.679 | 0.111 |
| 20784 | 1 | 48 | 48 | 4.634964 | 3.403 | 0.070 |
| 25455 | 2 | 36 | 72 | 7.304102 | 3.239 | 0.045 |
| 29393 | 2 | 48 | 96 | 7.458448 | 4.230 | 0.044 |
| 32863 | 3 | 40 | 120 | 10.751325 | 3.668 | 0.030 |
| 36000 | 3 | 48 | 144 | 14.709495 | 3.217 | 0.022 |
| 41600 | 4 | 48 | 192 | 26.461508 | 2.384 | 0.012 |

$$E_s(p,N) = \frac{T(N,1)}{T(pN,p)}$$

$$S_s(p,N) = E_s p$$

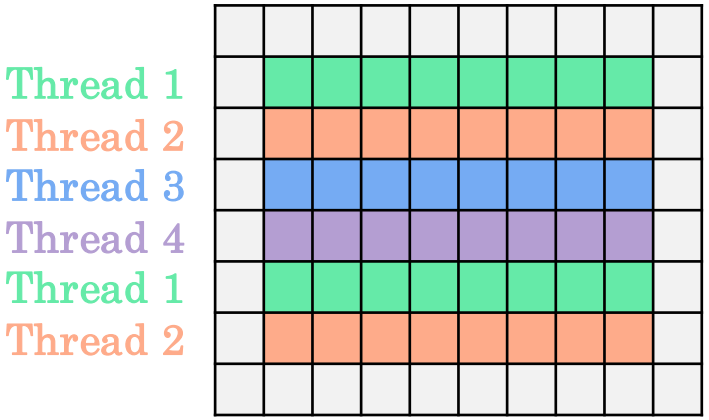


# Soluzione OpenMP

- Ogni processo esegue il calcolo di **rows_per_thread = (int) (N-2) / num_processi** righe della matrice B (più eventualmente una in eccesso).
  - Elementi sulla stessa riga sono assegnati allo stesso thread
  - Le iterazioni sono assegnate ai thread staticamente secondo una politica round-robin

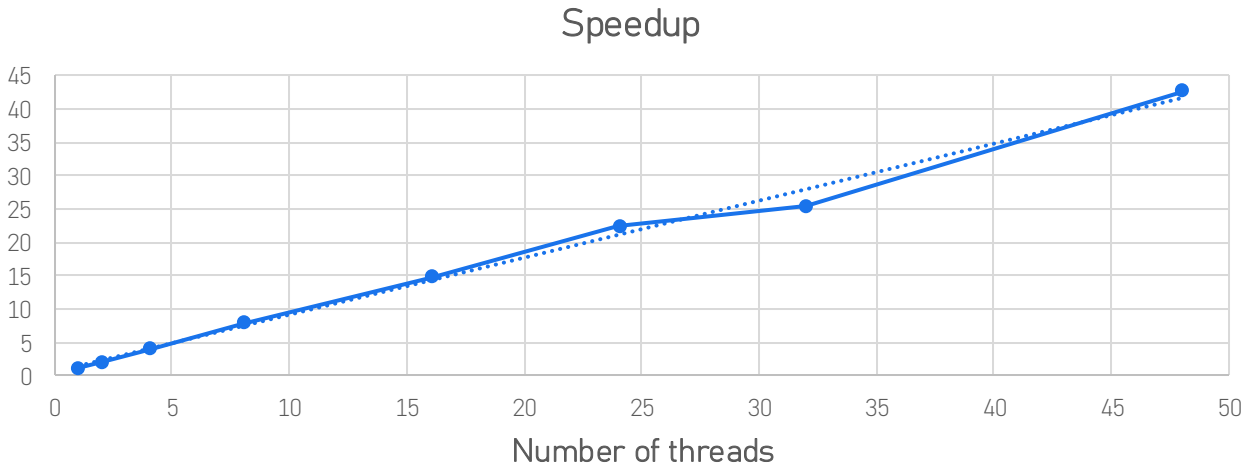
```
start = omp_get_wtime();

#pragma omp parallel for num_threads(p)
            schedule(static, 1) private(somma)
for(int i = 1; i < N-1; i++) {
    for(int j = 1; j < N-1; j++) {

       // Calcolo elemento B[i-1][j-1]

    }
 }

 end = omp_get_wtime();
```

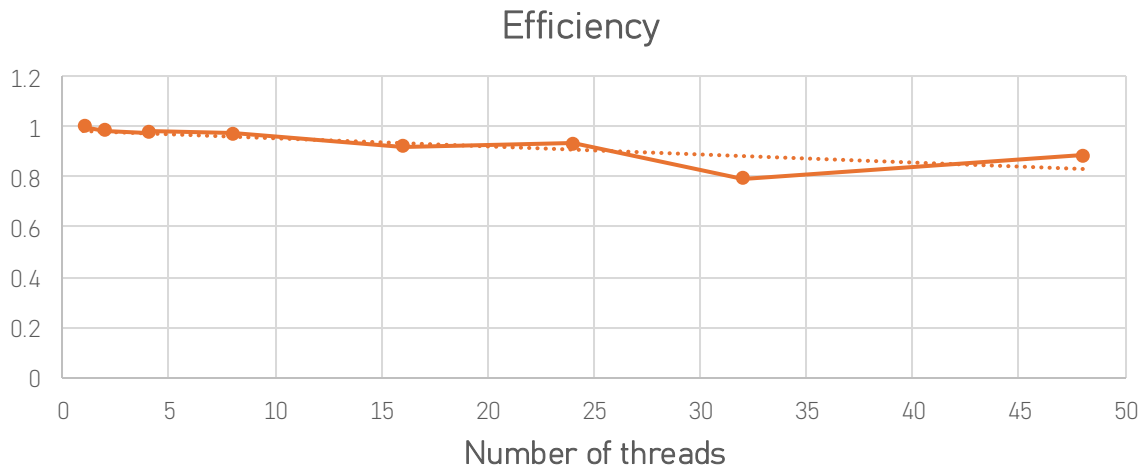


# OpenMP - Strong scalability

| N | Threads | Time (sec) | Speedup | Efficienza |
|---|---|---|---|---|
| 15000 | 1 | 8.194095 | 1 | 1 |
| 15000 | 2 | 4.155008 | 1.972 | 0.986 |
| 15000 | 4 | 2.082428 | 3.935 | 0.983 |
| 15000 | 8 | 1.053214 | 7.780 | 0.972 |
| 15000 | 16 | 0.556058 | 14.736 | 0.921 |
| 15000 | 24 | 0.365762 | 22.403 | 0.933 |
| 15000 | 32 | 0.323172 | 25.355 | 0.792 |
| 15000 | 48 | 0.192347 | 42.600 | 0.887 |

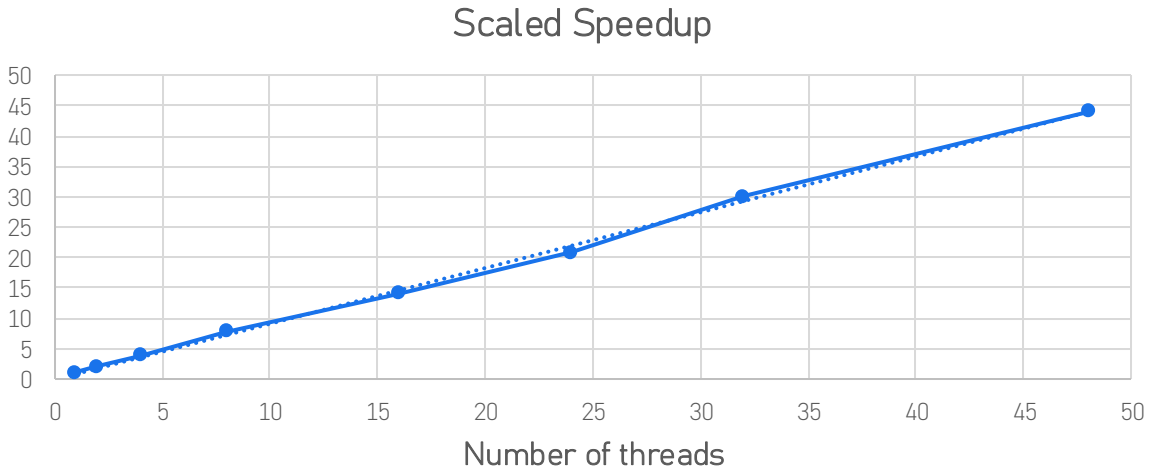
Il massimo livello di parallelismo è limitato dal numero di core disponibili in un singolo nodo di Galileo, ovvero 48.

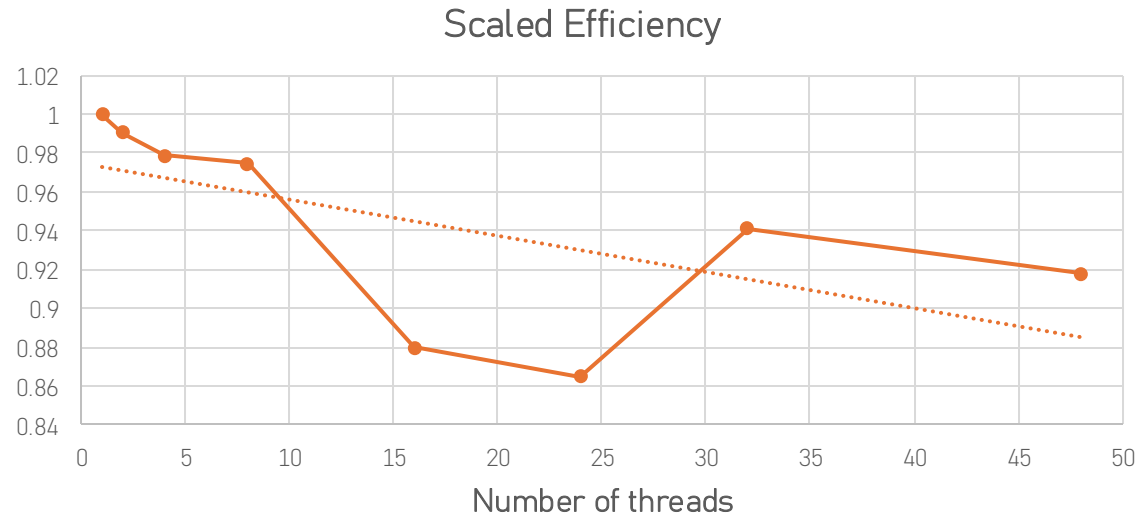


# OpenMP – Weak scalability

| N | Threads | Time (sec) | Speedup scalato | Efficienza scalata |
|---|---|---|---|---|
| 4000 | 1 | 0.583581 | 1 | 1 |
| 5656 | 2 | 0.589055 | 1.981 | 0.991 |
| 8000 | 4 | 0.596188 | 3.915 | 0.979 |
| 11312 | 8 | 0.598762 | 7.797 | 0.974 |
| 16000 | 16 | 0.663255 | 14.078 | 0.879 |
| 19596 | 24 | 0.674644 | 20.760 | 0.865 |
| 22624 | 32 | 0.620064 | 30.117 | 0.941 |
| 27712 | 48 | 0.635782 | 44.059 | 0.917 |




** Tutti i dati riportati fanno riferimento a simulazioni svolte sull'infrastruttura HPC Galileo100 di Cineca