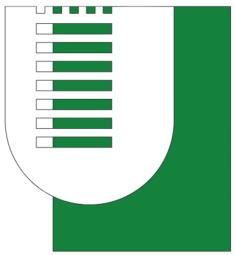
Prodotto tra Matrici con MPI e CUDA

Sistemi di Calcolo Parallelo e Applicazioni

Simone Nicosanti
simone.nicosanti@students.uniroma2.eu

Università degli Studi di Roma Tor Vergata
DICII
Laurea Magistrale in Ingegneria Informatica

Giugno 2024



TOR VERGATA
UNIVERSITY OF ROME

School of Engineering

Indice

Indice

Il Problema

Implementazione di un nucleo di calcolo per il calcolo del prodotto tra matrici, dove

- A, matrice di dimensioni $m * k$
- B, matrice di dimensioni $k * n$
- C, matrice di dimensioni $m * n$

$$C \leftarrow C + AB$$

Il nucleo di calcolo è stato sviluppato in due versioni:

- MPI
- CUDA

Metriche Considerate

FLOPS

$$FLOPS = \frac{2 * m * n * k}{T}$$

T misurato in secondi

SpeedUp

$$SpeedUp = \frac{T_{Seq}}{T_{Par}}$$

Errore Relativo

$$RelativeError = \frac{\|C_{Seq} - C_{Par}\|}{\|C_{Seq}\|}$$

Matrici di Input

Matrici generate:

- In-Memory
- Randomicamente
- Per ogni matrice
  - Tre esecuzioni parallele
  - Esecuzione sequenziale: fino a dimensione $k = 5000$

Due casi principali:

- Matrici Quadrate $m = k = n$
  - Da dimensione 250 a 10000 a passo 250
- Matrici Rettangolari $m, n >> k$
  - m ed n fissi a 10000
  - $k \in [32, 64, 128, 156]$

Indice

Distribuzione ScalaPack

- Processi organizzati su una griglia bidimensionale.
- Matrice divisa lungo le righe e lungo le colonne $\rightarrow$ blocchi distribuiti ai processi
- Parametri da considerare:
  - Dimensioni delle matrici m, k, n
  - Numero di processi P
  - Dimensioni della griglia di processi P_r, P_c

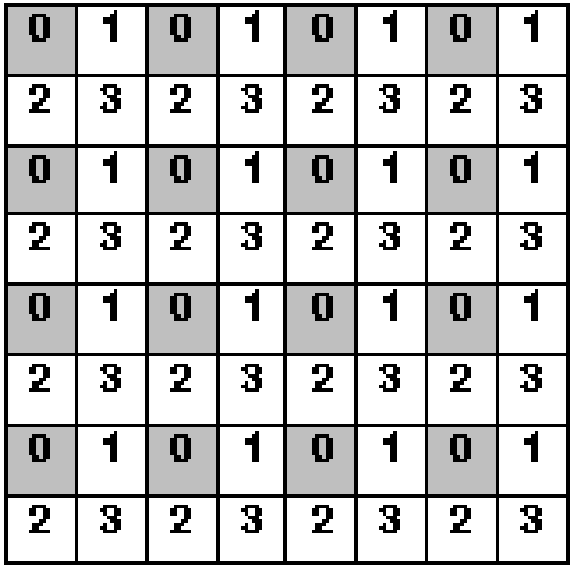


Figura 1: ScalaPack - Block Cyclic Distribution

Possibili distribuzioni

Nelle figure, supponiamo i processi distribuiti su una griglia $2 * 3$

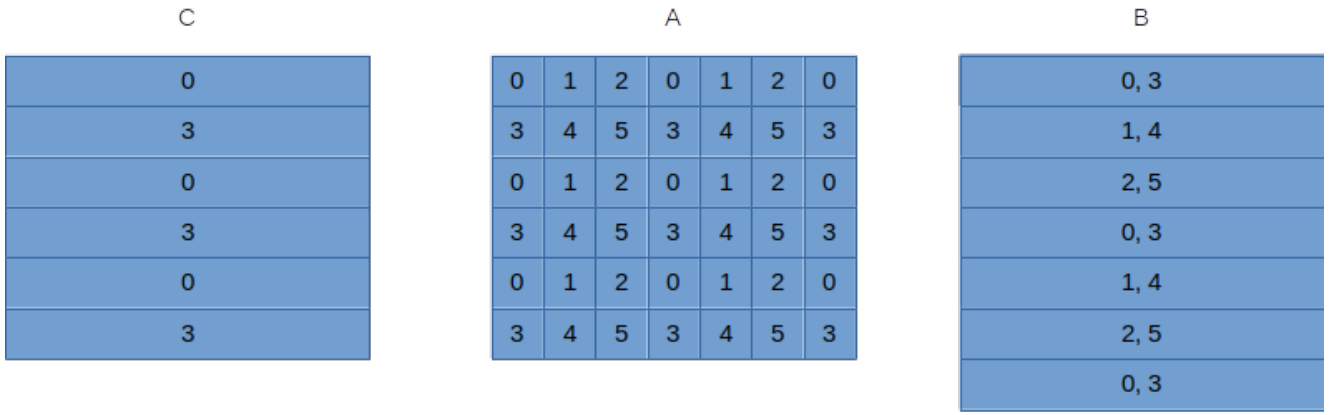


Figura 2: MPI - Distribuzione a Blocchi di A

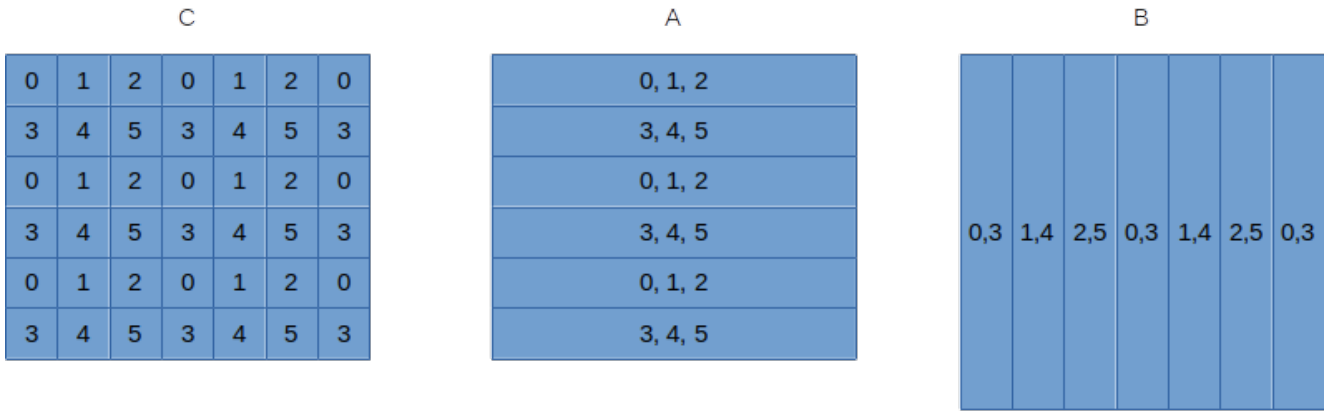


Figura 3: MPI - Distribuzione a Blocchi di C

Distribuzione A VS Distribuzione C

- Distribuzione A
  - Più punti di sincronizzazione (necessario almeno un gruppo di `reduce`)
  - Meno simile al classico prodotto riga-colonna
- Distribuzione C
  - Non sono necessarie reduce intermedie
  - Più simile al prodotto riga-colonna classico

Indice

Uso dei tipi MPI

Vantaggi:

- Riutilizzo
- Ottimizzazione MPI
- Evitare cicli annidati

Svantaggi:

- Utilizzo memoria
  - Supporto creazione
  - Mantenimento

Costruzione:

- `indexed` per dividere le colonne della matrice (necessari 3 tipi)
- `hindexed_block` per dividere le righe in gruppi di `indexed` (necessari 9 tipi)

Figura 4: Tipi indexed

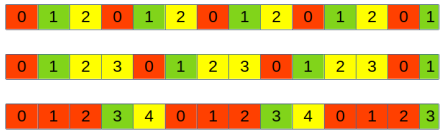


Figura 5: Tipi per processi

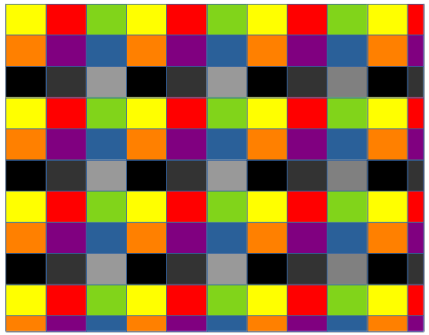


Figura 6: Tipi hindexed

Scatter & Gather

Scatter:

- `send`
  - Unica per processo ricevente specificando
    - Tipo derivato di invio: come i dati sono disposti in memoria del mittente
    - Punto di partenza dei dati nella matrice
- `recv`
  - Unica per processo ricevente specificando
    - Tipo di base (ricezione contigua)
    - Numero di elementi da ricevere

Gather

- `send`
  - Unica per processo mittente specificando
    - Tipo di base: dati contigui in memoria del processo mittente
    - Numero di elementi da inviare
- `recv`
  - Unica per processo mittente specificando
    - Tipo derivato: ricezione con pattern a blocchi
    - Punto di partenza dei dati nella matrice C

Indice

Prestazioni

GFLOPS - Caso Quadrato

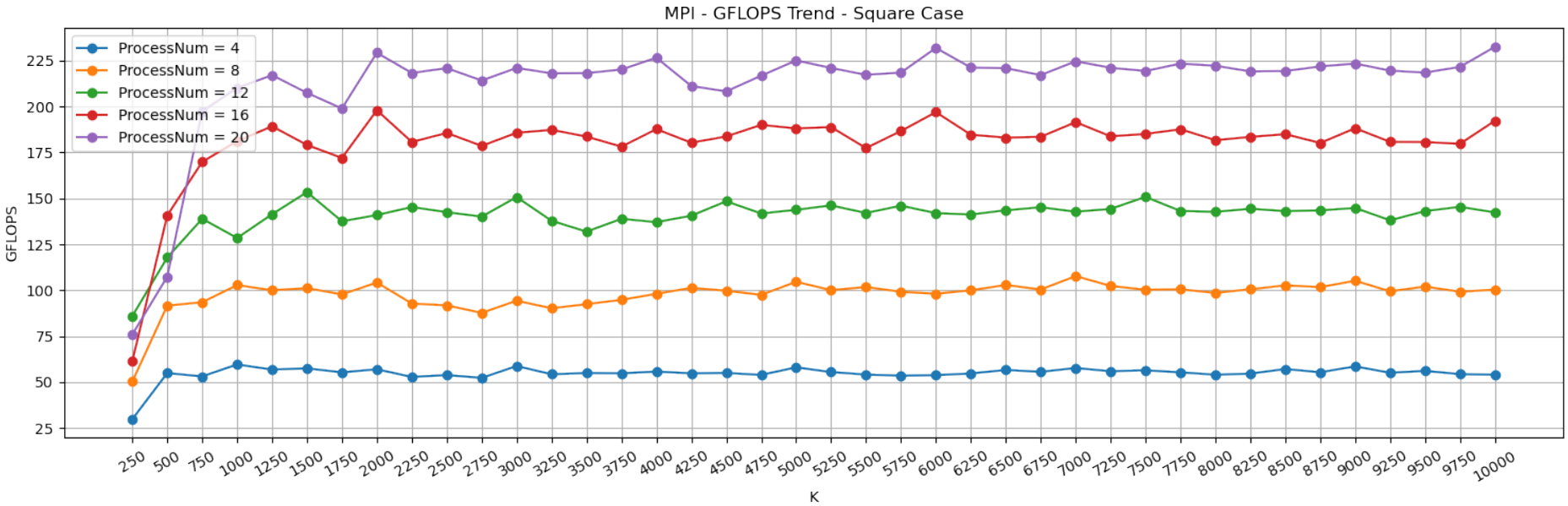


Figura 7: MPI - GLOPS Trend - Quadrato

Prestazioni

SpeedUp - Caso Quadrato

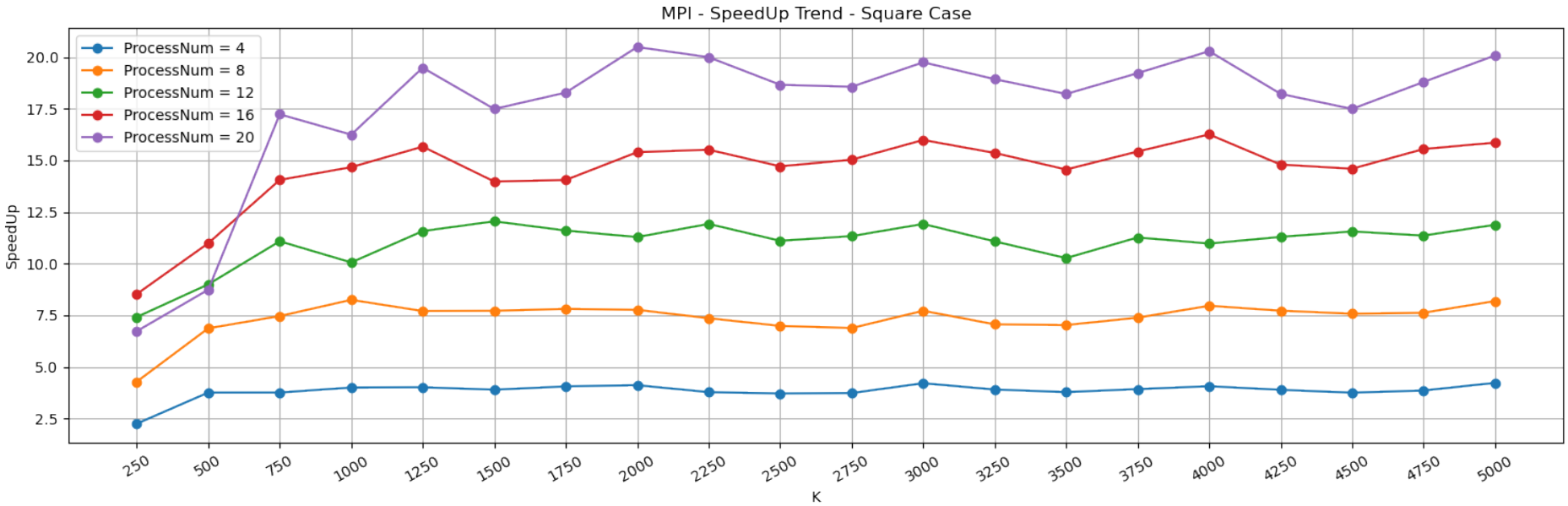


Figura 8: MPI - SpeedUp Trend - Quadrato

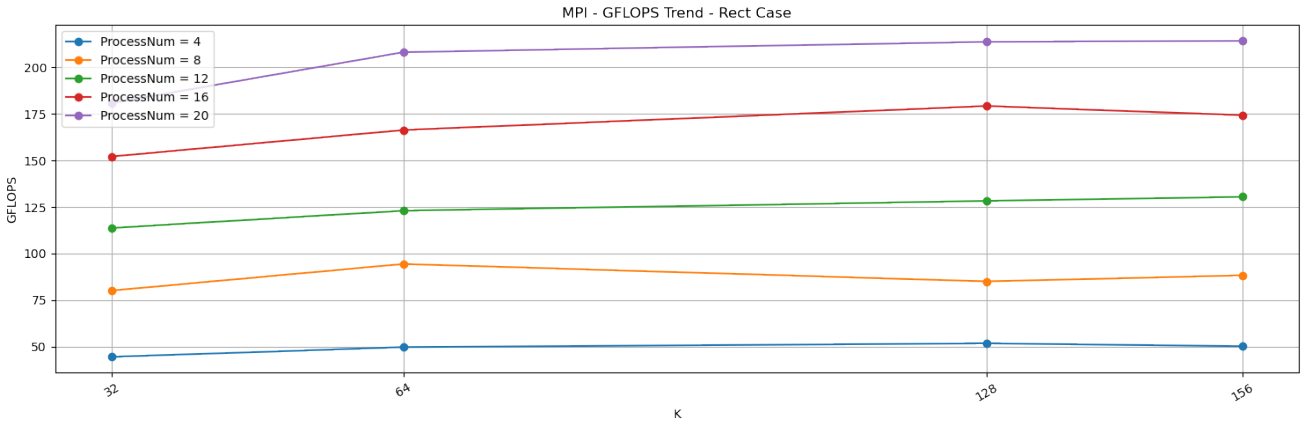


Figura 9: MPI - GLOPS Trend - Rettangolare

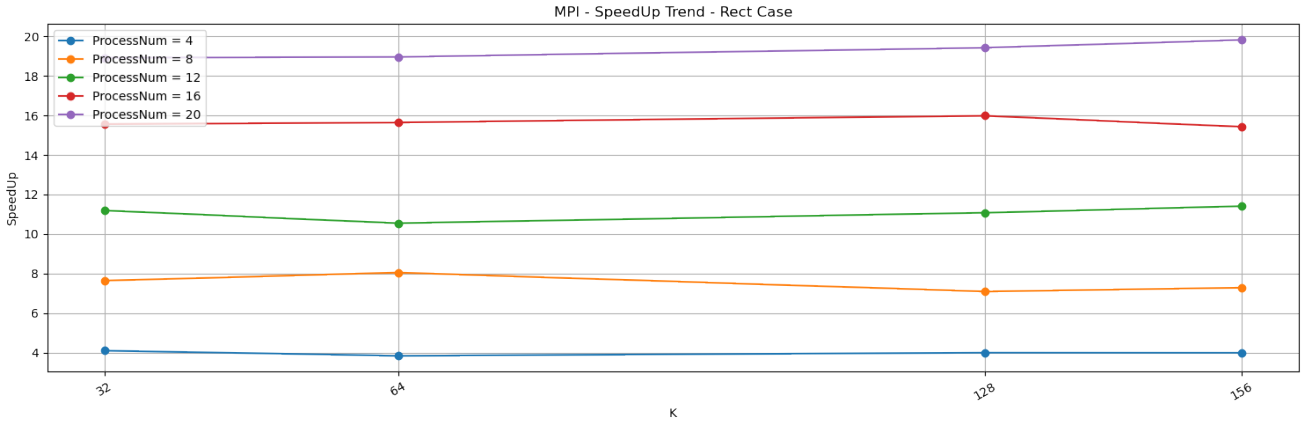


Figura 10: MPI - SpeedUp Trend - Rettangolare

Indice

Introduzione

- Cinque versioni di kernel
  - Raffinamenti successivi
  - Studio delle differenze prestazionali
- Matrici allocate in host
  - Input spostato con
    - `cudaHostRegister`
    - `cudaMallocPitch`
    - `cudaMemcpy2D`

Figura 11: CUDA e NVIDIA

Indice

Kernel 0 e Kernel 1

Caratteristiche:

- Direttamente in memoria globale
- Accumulatore in registro

Dove $div(t) = \frac{t-1}{BLOCK\_SIZE} + 1$

Kernel 0

$$GRID\_DIM = (div(m), div(n))$$

Kernel 1

$$GRID\_DIM = (div(n), div(m))$$

Cambio nella griglia $\rightarrow$ Cambio indicizzazione della riga e della colonna nel thread

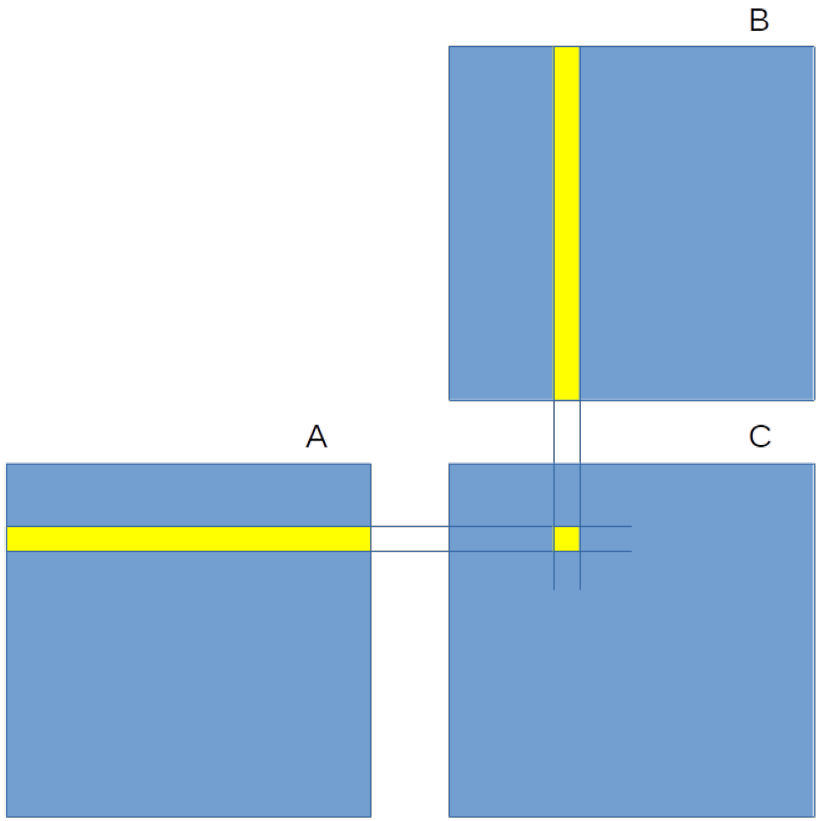


Figura 12: Kernel 0/1 - Prodotto

Kernel 0 VS Kernel 1 - Accesso in Memoria Globale

Consideriamo due thread in stesso warp come $(0,0)$ e $(1,0)$

Kernel 0

$$rowIdx = threadIdx.x + ...$$
$$colIdx = threadIdx.y + ...$$

Kernel 1

$$rowIdx = threadIdx.y + ...$$
$$colIdx = threadIdx.x + ...$$

Accesso

$$A[rowIdx][kIdx] \ ; \ B[kIdx][colIdx]$$

Conclusione: Il Kernel 1 usa il coalesce degli accessi in memoria globale

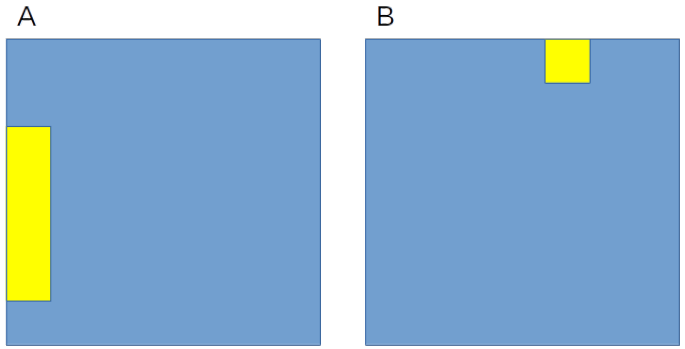


Figura 13: Kernel 0 - Pattern Accesso

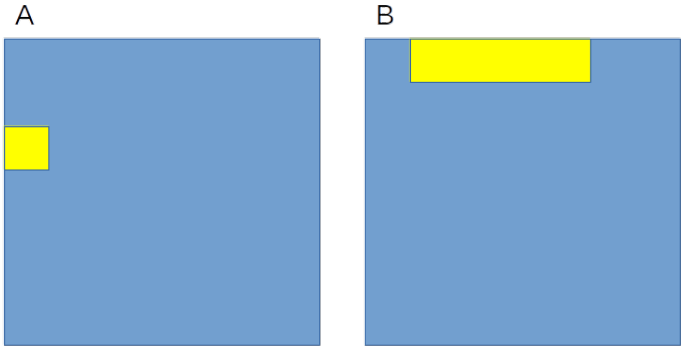


Figura 14: Kernel 1 - Pattern Accesso

Kernel 2

- Porzione dei dati all'interno della shared memory (coalesce)
  - Scorrimento lungo dimensione k
- Un thread calcola un singolo elemento della matrice C
  - Ad ogni iterazione del ciclo in k, calcola il prodotto tra sotto riga e sotto colonna
  - Accumulatore in un registro locale

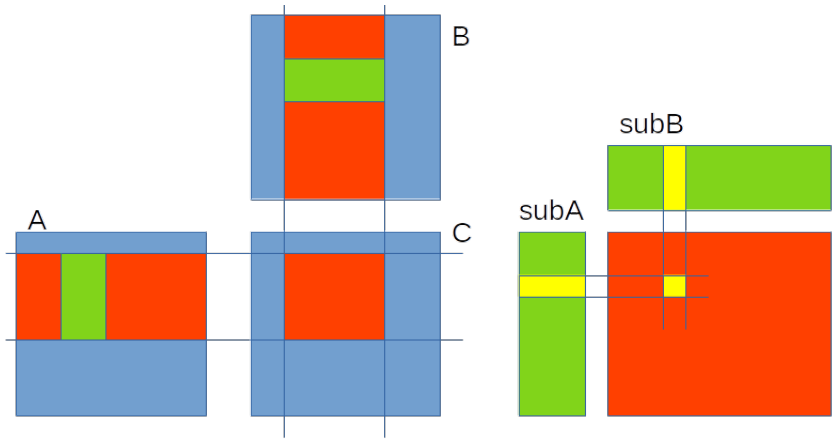


Figura 15: Kernel 2 - Prodotto

Kernel 3 e Kernel 4

- Porzione dei dati all'interno della shared memory (coalesce)
  - Scorrimento lungo la dimensione k
- Un thread calcola una tile di C
  - Accumulatore come una matrice
  - Blocco di A diviso in tile di size TA; blocco di B è diviso in tile di size TB
  - Thread moltiplica una tile di B per una tile di A $\rightarrow$ tile di C di dimensione $TA * TB$
- Caching in registri di colonne di subA e righe di subB

Differenza: In Kernel 4 subA è caricata trasposta in shared memory (accesso contiguo)

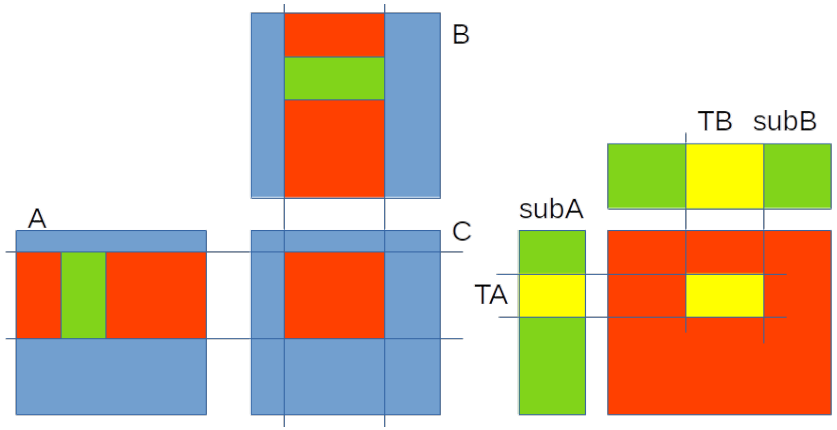


Figura 16: Kernel 3/4 - Prodotto

Indice

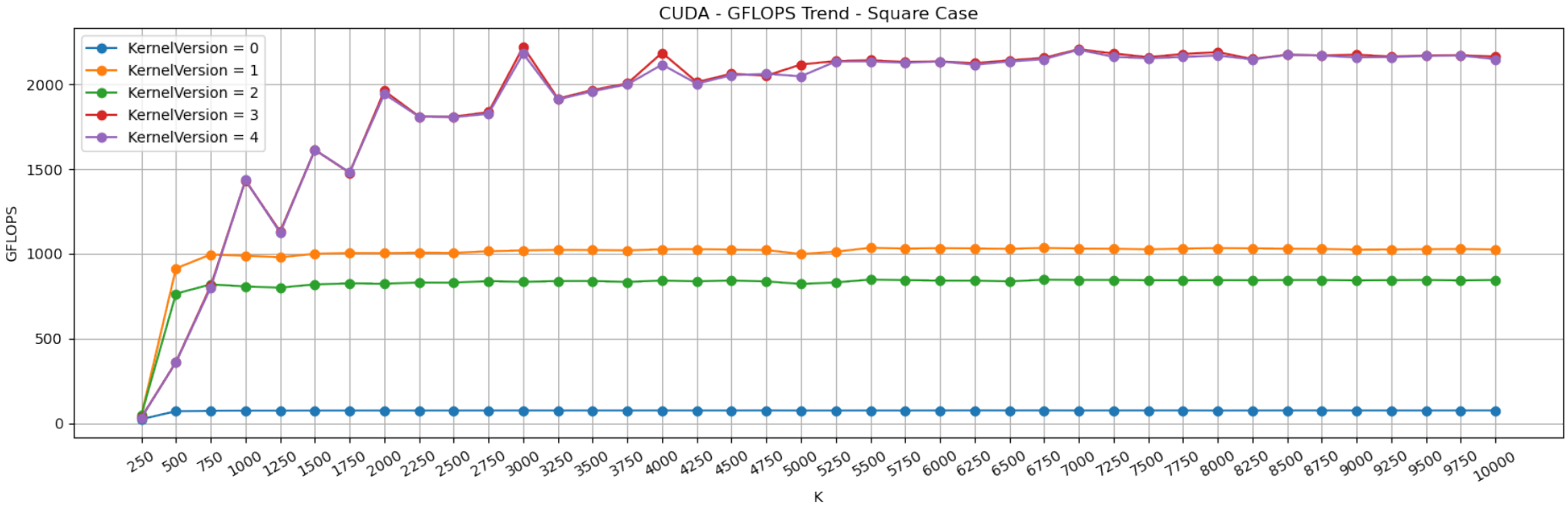


Figura 17: CUDA - GFLOPS Trend - Quadrato

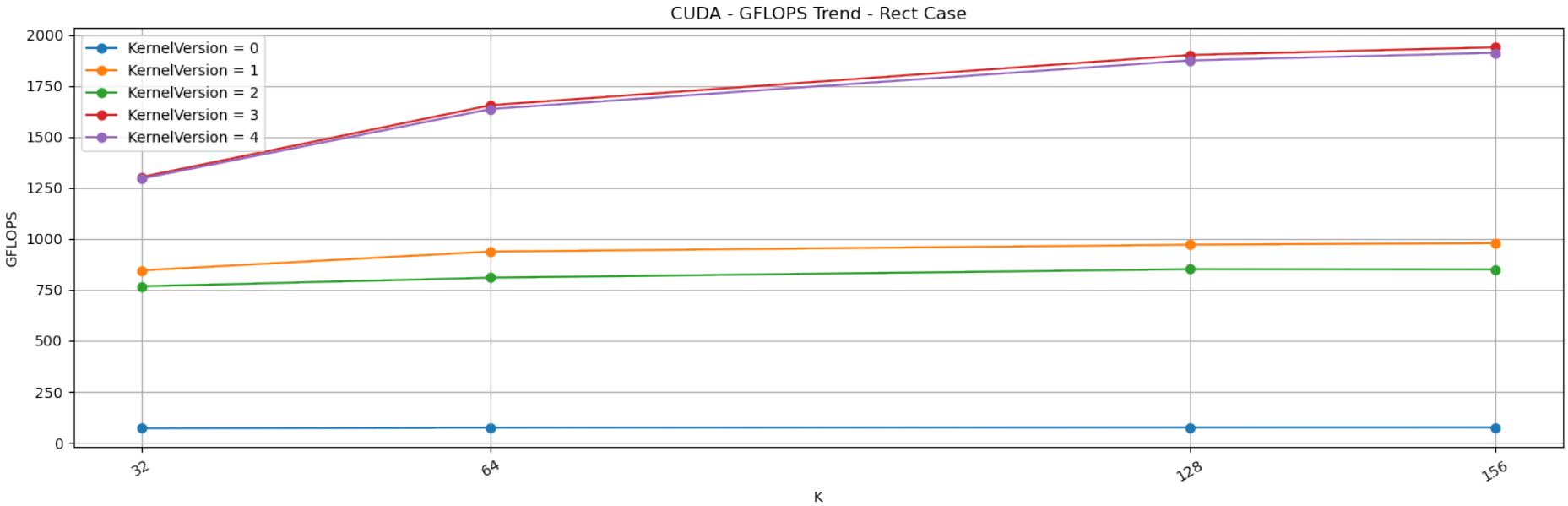


Figura 18: CUDA - GFLOPS Trend - Rettangolare

GRAZIE PER L'ATTENZIONE!