Prodotto Matrice Sparsa - MultiVettore

Un analisi delle prestazioni di implementazioni in CUDA e OMP

Daniele La Prova, Luca Fiscariello
daniele.laprova@students.uniroma2.eu,luca.fiscariello@students.uniroma2.eu

Universita' degli Studi di Roma Tor Vergata

February 27, 2023

Roadmap

Introduzione

- Il progetto ha come obiettivo quello di implementare un prodotto tra una matrice sparsa e un multivettore confrontando le prestazioni di implementazioni seriali e parallele;
- Le implementazioni parallele sono state realizzate usando le tecnologie `OpenMP` e `CUDA`, in diversi formati di matrice sparsa;

Introduzione: Formati

- `ELLPACK`: Una matrice contiene i valori non nulli, le cui righe e colonne indicizzano un'altra matrice che contiene i valori delle colonne dei NZEs;
- `CSR`: Un array associa a ogni posizione una riga e un puntatore a un array di colonne, indicando da quale posizione di quest'ultimo iniziano le colonne di quella riga. Un ultimo array contiene i NZEs alle stesse posizioni delle colonne;
- `COO`: Gli indici di riga e di colonna e i valori dei NZEs sono mantenuti in tre distinti arrays paralleli;

Formati

- `MatrixMarket`: Decora un puntatore `FILE *` che rappresenta un file contenente una matrice in formato MatrixMarket, offrendo operazioni su di essa e interpretandone il banner.
- `MultiVector`: un formato che memorizza un multivettore in un array denso bidimensionale, in cui ogni riga memorizza un puntatore a un array le cui posizioni indicano le colonne;
- `ArrayDense`: un formato che memorizza una matrice densa in un array unidimensionale. Molto utile per memorizzare il risultato di implementazioni di prodotti tra matrici che coinvolgono l'uso di thread paralleli, ovvero OMP e CUDA, poiché thread safe.

Roadmap

Modello

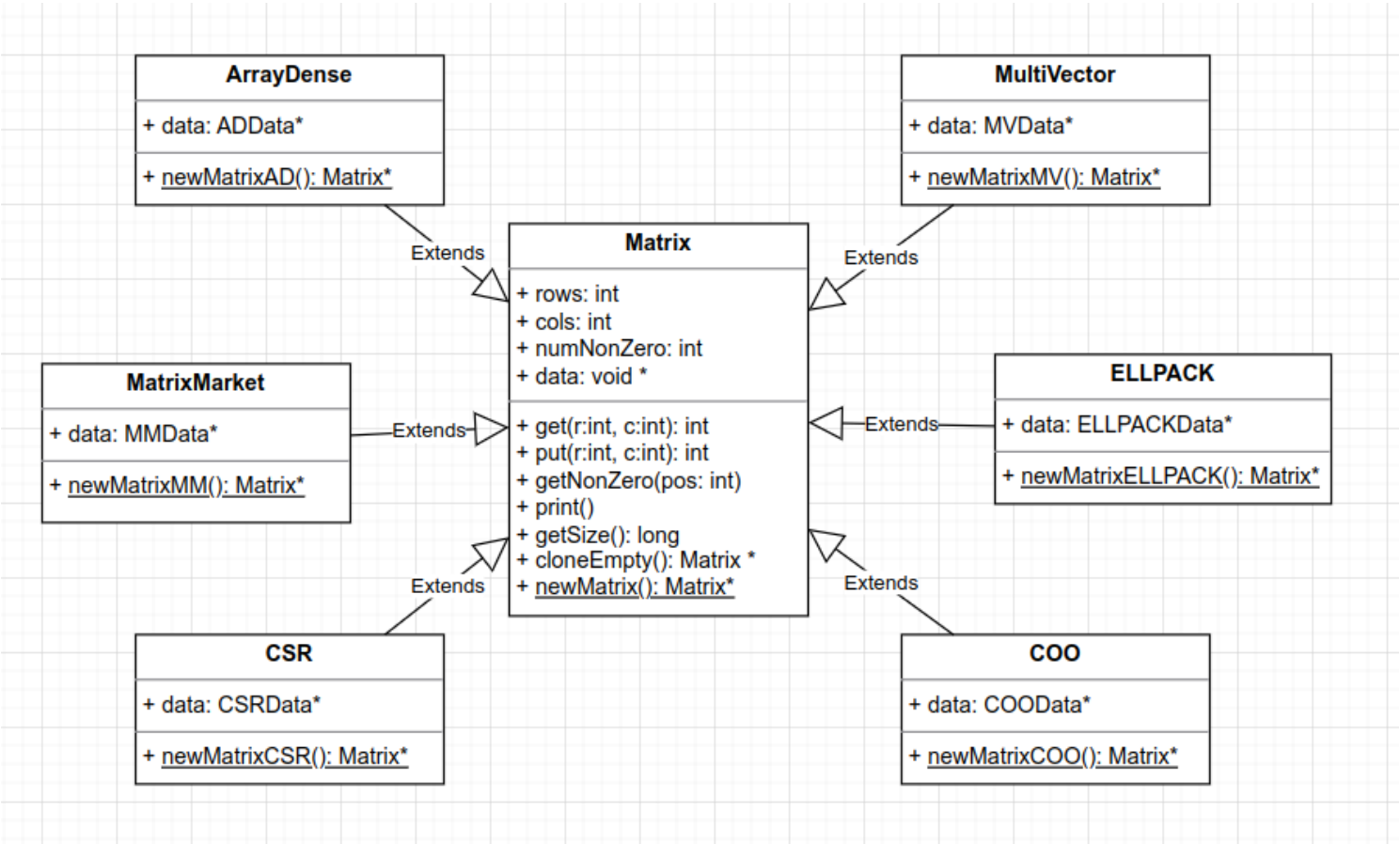

Figure 1: Sono stati applicati i pattern `STATE`, `BUILDER`, `PROTOTYPE`, `MEDIATOR` per modellare le astrazioni chiave

Modello

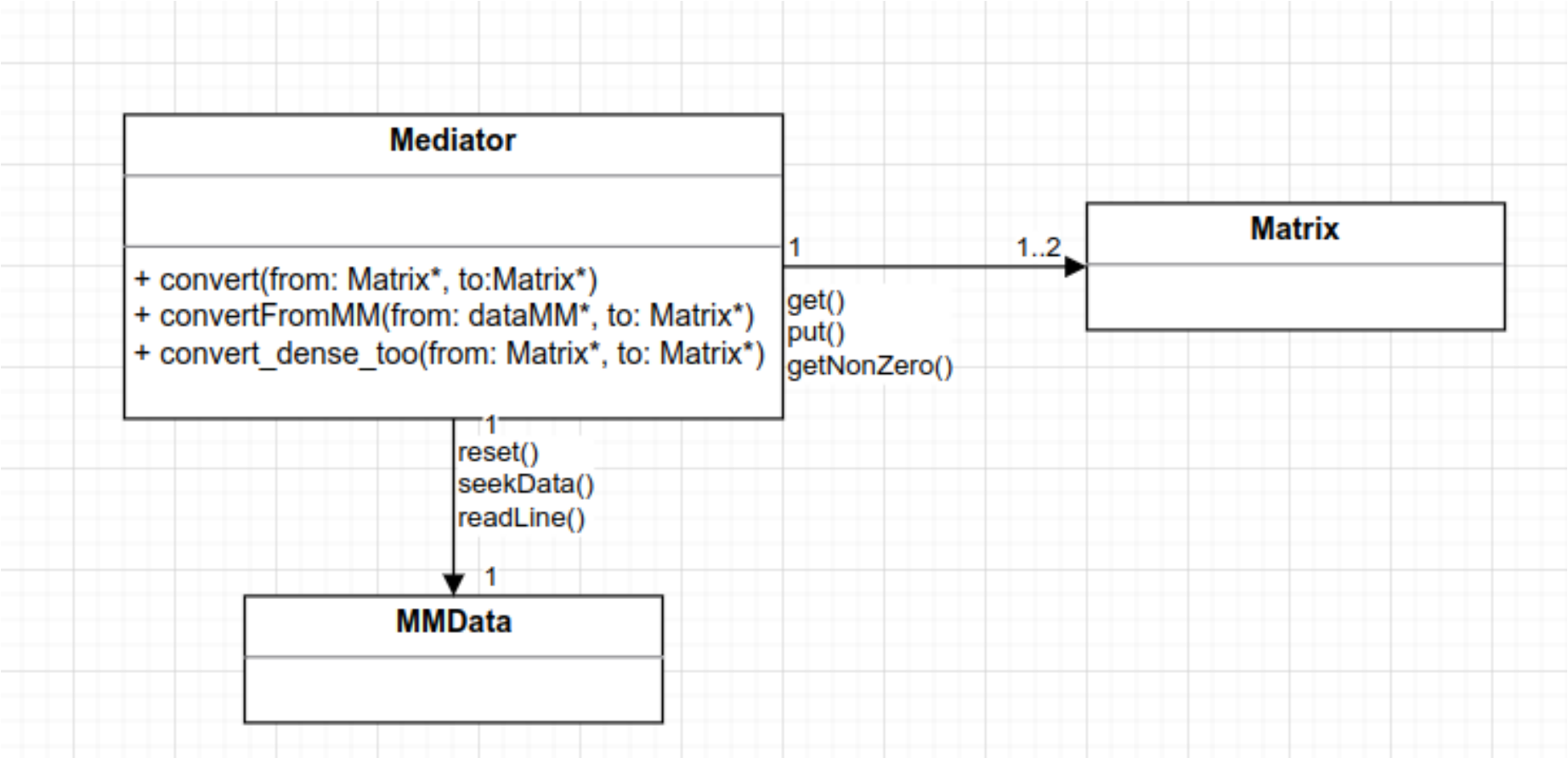

Figure 2: Il componente Mediator espone delle funzioni che permettono di ricopiare gli elementi contenuti in una matrice di un certo formato in un altra nello stesso o altro formato

Implementazione Prodotti

- Per implementare i prodotti tra formati di matrici sparse e multivettori è stato deciso di scrivere un file di interfacce `product.h`, in cui è possibile aggiungere implementazioni di prodotti che devono rispettare tutte la stessa interfaccia;
- Per poter aggiungere un'implementazione di prodotto, è sufficiente scriverne l'implementazione in un file `.c` (oppure `.cu`) dedicato e aggiungerne l'interfaccia tra le altre.

productMatrixMatrixSerial()

- È stato deciso di fornire un'implementazione "ingenua" del prodotto seriale utilizzando i metodi dell'interfaccia `Matrix` per poter ottenere una baseline facilmente senza preoccuparsi del formato delle matrici coinvolte, al costo di una ridotta efficienza.

```
1  //Scorro tutti gli elementi non zeri della prima matrice
2  for(int i =0; i< matrix1->numNonZero; i++){
3      nze = matrix1->getNonZero(matrix1,i);
4      //scorro tutti gli elementi della riga "nze->row" della seconda matrice
5      for(int j =0; j< matrix2->cols; j++){
6          /**
7           * Moltiplico un elemento non nullo della prima matrice per tutti gli
8           * elementi della riga della seconda matrice.
9           * Sommo questo risultato parziale nella matrice risultato.
10          * La posizione in cui sommare questo risultato parziale
11          * è definita dalla riga dell'elemento della prima matrice e dalla
12          * colonna del valore della seconda matrice.
13          */
14         result[nze->row][j] += nze->value * matrix2->get(matrix2,nze->col,j);
15     }
16 }
```

productMatrixMatrixParallelEllpack()

Ottimizzazioni

- Parallelismo senza sincronizzazione.
- Coalescenza.
- Memoria Shared.
- Elusione conflitti tra banchi shared

productMatrixMatrixParallelEllpack()

- La matrice *A_Values* mantiene i valori della matrice sparsa, mentre *A_Cols* mantiene gli indici di colonna dei non zeri;

productMatrixMatrixParallelEllpack()

- La matrice *A_Values* mantiene i valori della matrice sparsa, mentre *A_Cols* mantiene gli indici di colonna dei non zeri;

Idea

Scomporre la matrice *A_Values* in blocchi bidimensionali e associare ciascuno di questi blocchi di dati un blocco di $32 * 32$ thread sempre bidimensionale.

productMatrixMatrixParallelEllpack()

- La matrice *A_Values* mantiene i valori della matrice sparsa, mentre *A_Cols* mantiene gli indici di colonna dei non zeri;

Idea

Scomporre la matrice *A_Values* in blocchi bidimensionali e associare ciascuno di questi blocchi di dati un blocco di $32 * 32$ thread sempre bidimensionale.

- È necessario portare in memoria condivisa ogni sottomatrice così creata insieme ai valori del multivettore che andranno moltiplicati per i NZEs, individuati dai valori di *A_Cols*.
- Funziona bene per matrici grandi poiché le ottimizzazioni dette in precedenza portano benefici se i dati sono acceduti molto frequentemente.

productMatrixMatrixParallelEllpack()

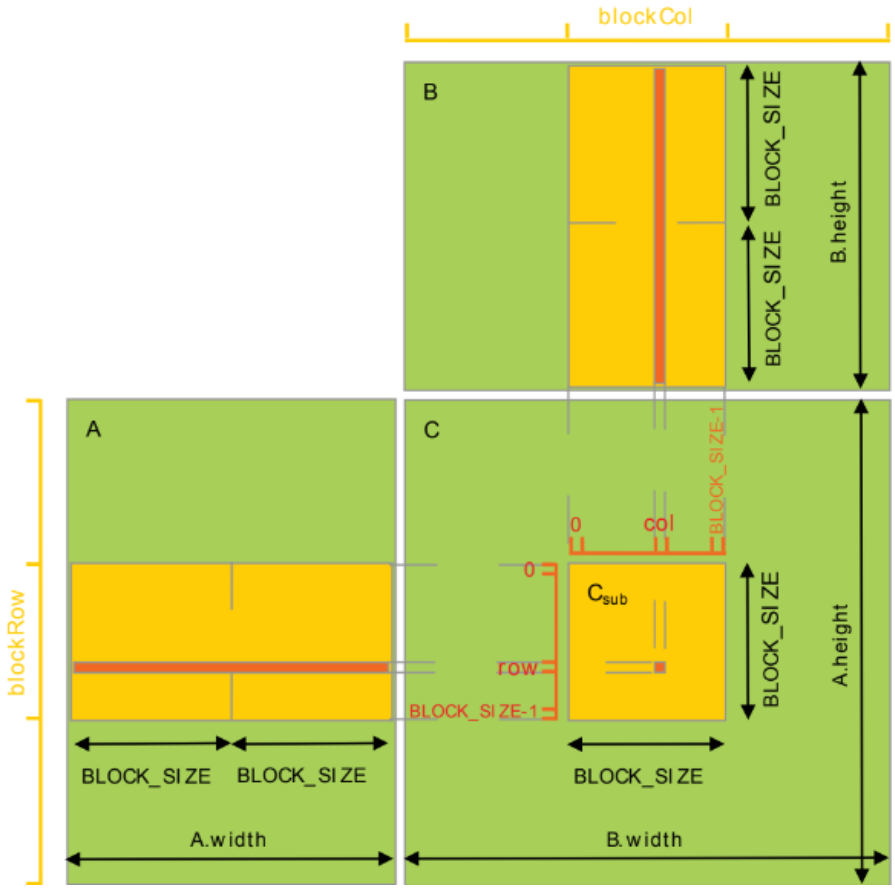

Figure 3: Ogni blocco di thread si prende un blocco di dati dalla matrice *A_Values* e lo moltiplica per il rispettivo blocco di dati del multivettore, ottenendo così una sottomatrice del risultato

productEllpackMultivectorParallelCPU()

Idea

Ad ogni thread è assegnato un elemento della matrice sparsa e uno del multivettore, il quale li moltiplicherà e accumulerà il prodotto nel corrispondente elemento della matrice risultato. In questo modo, ogni thread dovrebbe ricevere un carico di lavoro per quanto più possibile uguale a quello degli altri, poiché le righe potrebbero avere un numero di elementi non zero molto diversi tra loro.

productEllpackMultivectorParallelCPU()

Idea

Ad ogni thread è assegnato un elemento della matrice sparsa e uno del multivettore, il quale li moltiplicherà e accumulerà il prodotto nel corrispondente elemento della matrice risultato. In questo modo, ogni thread dovrebbe ricevere un carico di lavoro per quanto più possibile uguale a quello degli altri, poiché le righe potrebbero avere un numero di elementi non zero molto diversi tra loro.

```
1  #pragma omp parallel for default(shared) schedule(static) collapse(3)
2  for (int r1 = 0; r1 < ellpackData ->rowsSubMat; r1++){
3      for (int c2 = 0; c2 < matrix2 -> cols; c2 ++){
4          for (int cSub = 0; cSub < ellpackData ->colsSubMat; cSub++){
5              #pragma omp atomic
6              resultData[r1 * matrix2 ->cols + c2] +=
7              ellpackData ->matValues[r1][cSub] *
8              multivectorData[ellpackData ->matCols[r1][cSub]][c2];
9          }
10     }
11 }
```

productCsrMultivectorParallelGPU()

Idea

Ogni riga della matrice CSR è associata al thread che ha la sua coordinata x uguale all'indice di riga, e per ogni colonna registrata in quella riga si accumula il prodotto tra il valore corrispondente alla colonna e il valore del multivettore che ha come riga la colonna di CSR e come colonna la coordinata y del thread.

productCsrMultivectorParallelGPU()

Idea

Ogni riga della matrice CSR è associata al thread che ha la sua coordinata x uguale all'indice di riga, e per ogni colonna registrata in quella riga si accumula il prodotto tra il valore corrispondente alla colonna e il valore del multivettore che ha come riga la colonna di CSR e come colonna la coordinata y del thread.

```
1 row = threadIdx.x + blockIdx.x * blockDim.x;
2 mvCol = threadIdx.y + blockIdx.y * blockDim.y;
3 startCol = firstColOfRowIndexes[row];
4 endCol = firstColOfRowIndexes[row + 1];
5 for (int c = startCol; c < endCol; c++){
6     partial += values[c] * getPitched(mv, columns[c], mvCol, mv_pitch, double);
7 }
8 getPitched(result, row, mvCol, result_pitch, double) = partial;
```

productCsrMultivectorParallelGPU()

- Tutte le aree di memoria passate come parametri del kernel presentano il type qualifier `__restrict__` che indica a `nvcc` che i puntatori forniti contengono indirizzi diversi tra loro (nessuno tra loro è un alias di un altro tra loro), permettendo al compilatore di apportare riordino dele istruzioni e caricamento di dati in registri per ottimizzare le prestazioni.
- Le aree di memoria che devono ospitare i dati del formato CSR sono allocate utilizzando `cudaMalloc()` e `cudaMallocPitch()`, che permettono di allineare i dati in maniera tale da coalizzare gli accessi in memoria;
- Prima di trasferire il multivettore sul device o il risultato sull'host, è necessario preliminarmente convertirne la rappresentazione sull'host in un ArrayDense;

productCsrMultivectorParallelGPU()

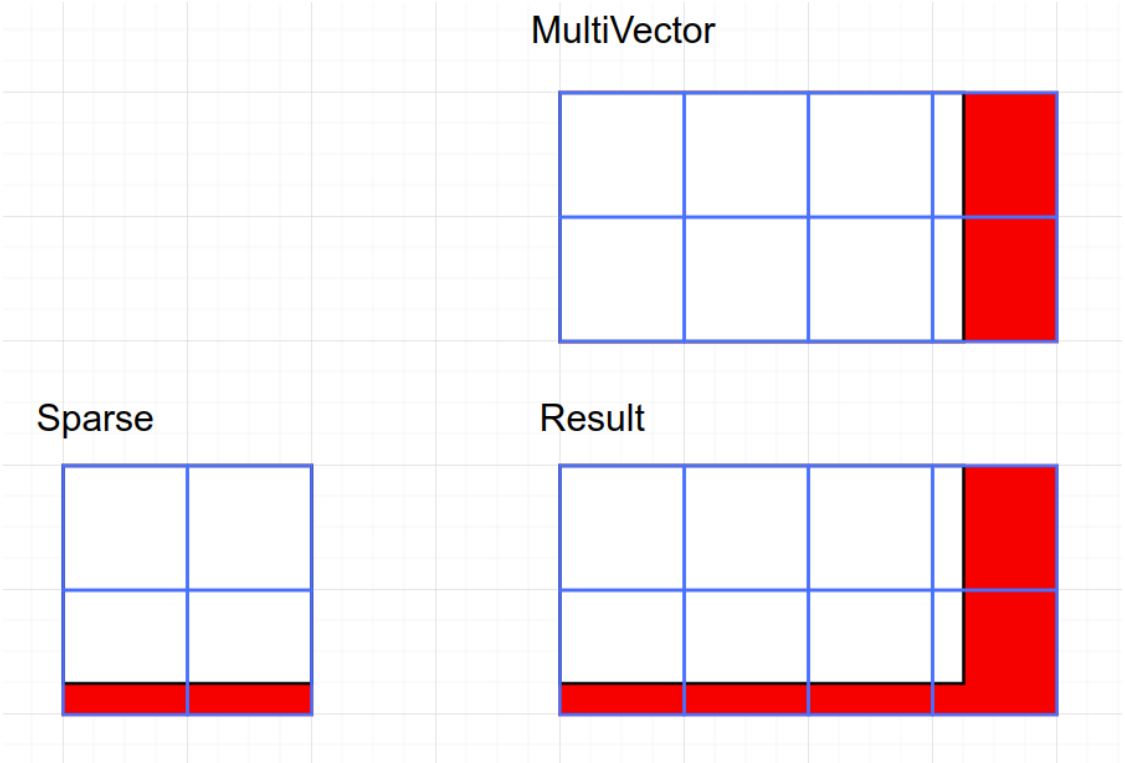

Figure 4: `ajustToWarpSize()` evita la divergenza che si sarebbe tra i thread di un warp se le dimensioni della sottomatrice (in bianco) assegnata a un blocco (in blu) non corrispondono a quelle del blocco stesso allocando memoria extra di padding (in rosso), a causa di una necessaria sanitizzazione degli indici pena l'occorrenza di segfaults.

productCsrMultivectorParallelCPU()

Idea

si scorrono una riga per volta la matrice in formato CSR e ogni elemento della riga viene moltiplicato per l'elemento corrispondente del multivettore.

productCsrMultivectorParallelCPU()

- `pragma omp parallel for`: per parallelizzare il ciclo esterno, che scorre tutte le righe e le colonne della matrice in formato CSR;
- `collapse(2)`: per combinare i due cicli nidificati (quello che scorre le righe e quello che scorre le colonne) in un singolo ciclo parallelo, in modo da evitare l'overhead del parallelismo per ogni ciclo.
- `schedule(static)`: per distribuire il lavoro in modo equo tra i thread.
- `pragma omp simd`: per sfruttare le istruzioni SIMD (Single Instruction Multiple Data) per parallelizzare l'operazione di prodotto scalare tra le righe della matrice CSR e le colonne del multivettore.
- `reduction(+:sum)`: per combinare più velocemente i risultati parziali di ogni thread e ottenere il risultato finale.

Roadmap

Esperimenti

- Per ogni nome di file di matrice su cui si vogliono effettuare gli esperimenti, si effettua la conversione con il Mediator nel formato desiderato;
- Per ogni matrice convertita, si accoppia un Multivettore generato per ogni valore di k (larghezza) desiderato, e si riportano i rispettivi puntatori in due arrays paralleli;
- La funzione `doExperiments()` userà lo stesso indice per accedere alle matrici in questi due arrays paralleli e applicherà ad ogni coppia tutte le implementazioni di prodotto desiderate, per il numero di `TRIALS` desiderati;
- Per ogni trial di un esperimento così effettuato viene generato un oggetto `Sample` che contiene tutte le informazioni di profilazione relative a quel trial;

Esperimenti

- Tutti i Samples così generati sono raccolti in un array, che viene infine convertito in un file csv che può essere elaborato da software statistici;
- I valori dei MultiVettori sono generati mediante l'uso di una variabile aleatoria uniforme(0,1) i cui valori sono prodotti da un generatore di Lehmer, implementato nella libreria presente nella cartella `random`;
- Per poter raccogliere i tempi di esecuzione, è stata utilizzata la funzione `clock_gettime()` prima e dopo aver lanciato il nucleo di calcolo omp, e gli eventi CUDA per la misurazione delle prestazioni dei calcoli su GPU;
- Gli esperimenti sono stato effettuati sul server di dipartimento.

Roofline

Definition (Intensità Aritmetica)

L'intensità aritmetica è definita come il rapporto tra il numero di operazioni effettuate da un nucleo di calcolo e il quantitativo di byte acceduti in memoria.

$I_{ELLPACK} = \frac{2k*nz}{16nz+4nz} = \frac{k}{10}$

$I_{CSR} = \frac{2k*nz}{16nz+4n+4nz} \approx \frac{k}{11}$

- **nz** : numero di non zero della matrice sparsa, che corrisponde al numero di elementi del multivettore coinvolti nel prodotto.
- **k** : numero di colonne del multivettore.
- **n** : numero di righe della matrice sparsa.

Roofline

Definition (Intensità Aritmetica)

L'intensità aritmetica è definita come il rapporto tra il numero di operazioni effettuate da un nucleo di calcolo e il quantitativo di byte acceduti in memoria.

$I_{ELLPACK} = \frac{2k*nz}{16nz+4nz} = \frac{k}{10}$

$I_{CSR} = \frac{2k*nz}{16nz+4n+4nz} \approx \frac{k}{11}$

Si osservi che ...

Le intensità di entrambi i formati dipendono da K, la cui crescita trasforma il prodotto in un problema **compute-bound** da uno **memory-bound**.

Roofline Ellpack

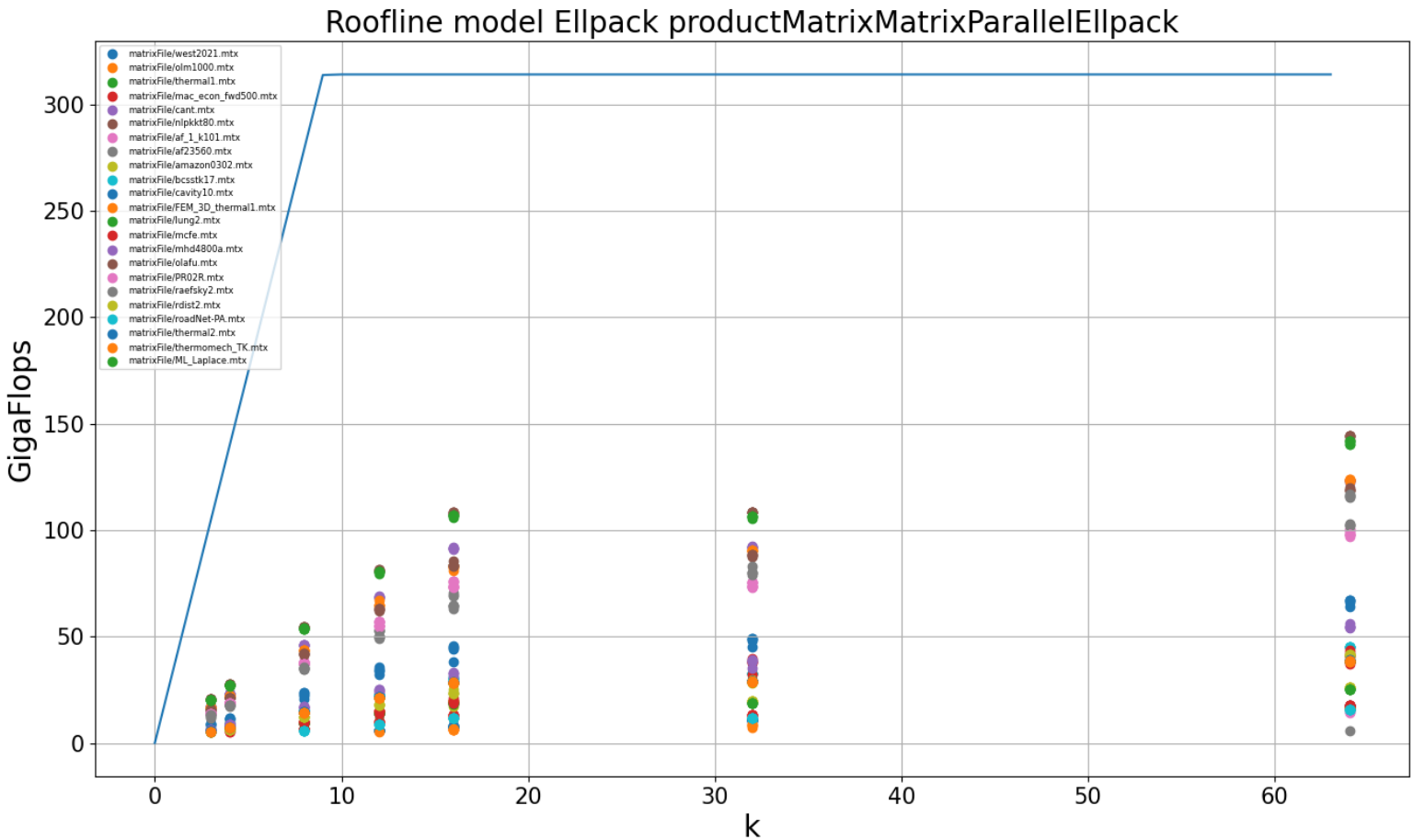

Figure 5

Roofline CSR

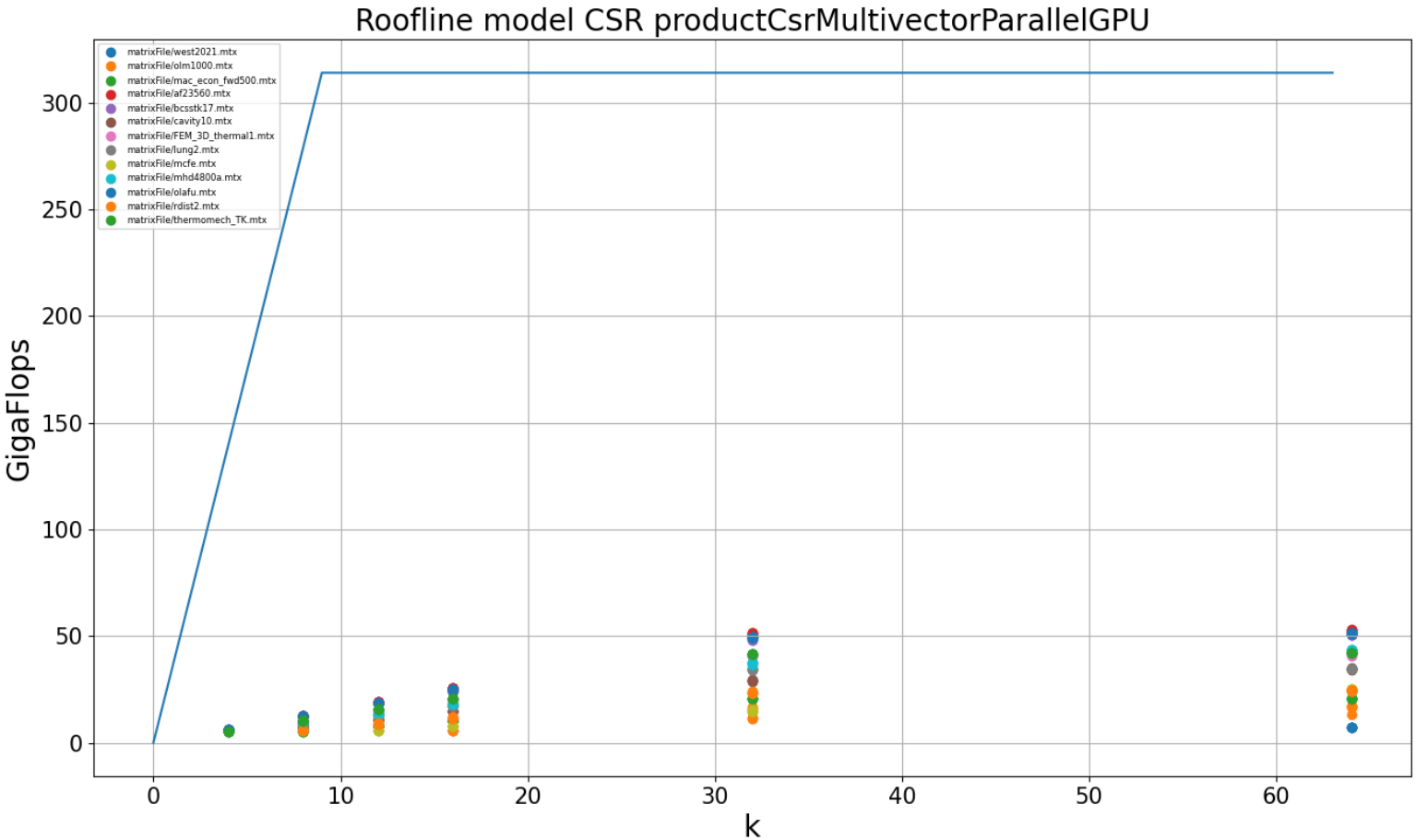

Figure 6

Prestazioni

Idea

Le prestazioni potrebbero dipendere dai seguenti fattori:

- dimensioni della matrice sparsa.
- valore di k.
- padding (per Ellpack).
- varianza dei non zeri nelle righe.

Prestazioni CUDA

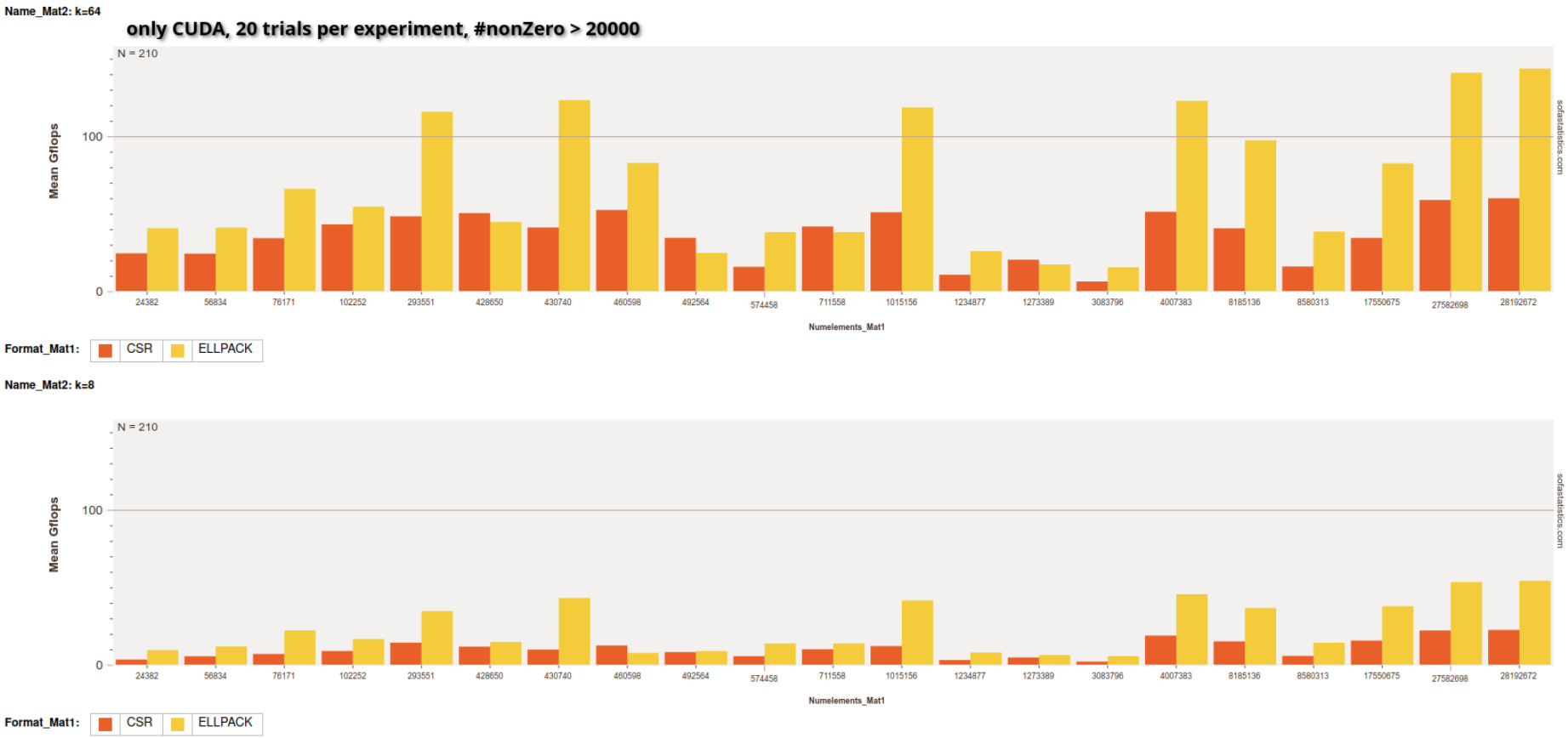

Figure 7: Mostra l'andamento delle prestazioni all'aumentare delle dimensioni della matrice, fissato k=64 e k=8

Prestazioni CUDA

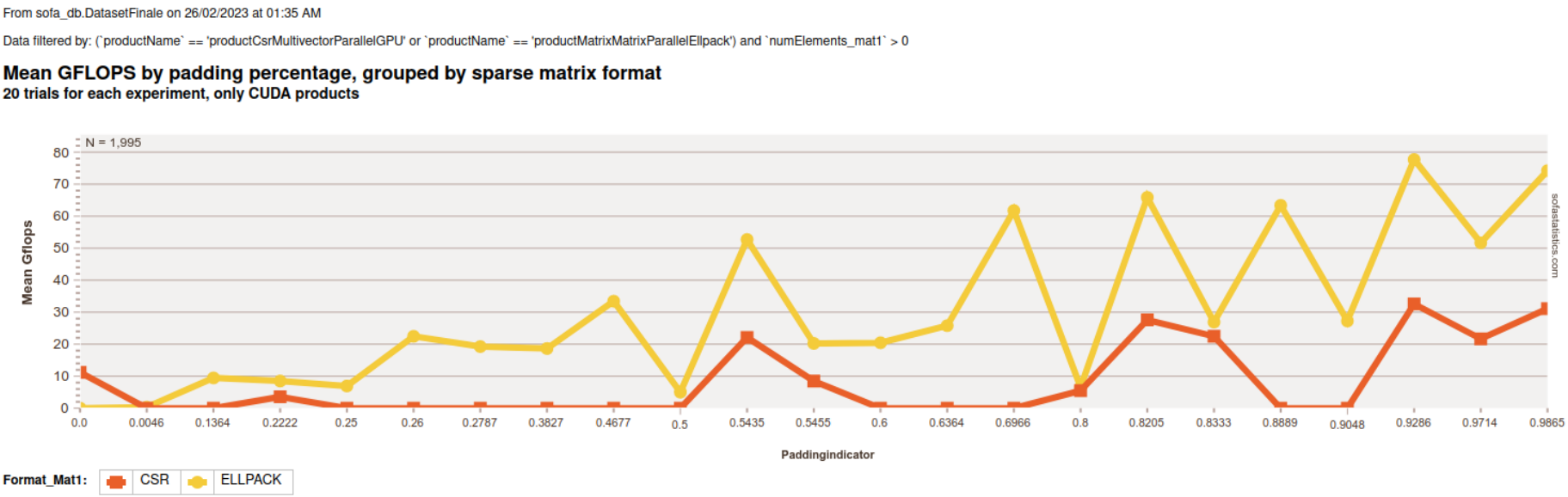

Figure 8: Mostra l'andamento delle prestazioni al variare del padding

Prestazioni CUDA

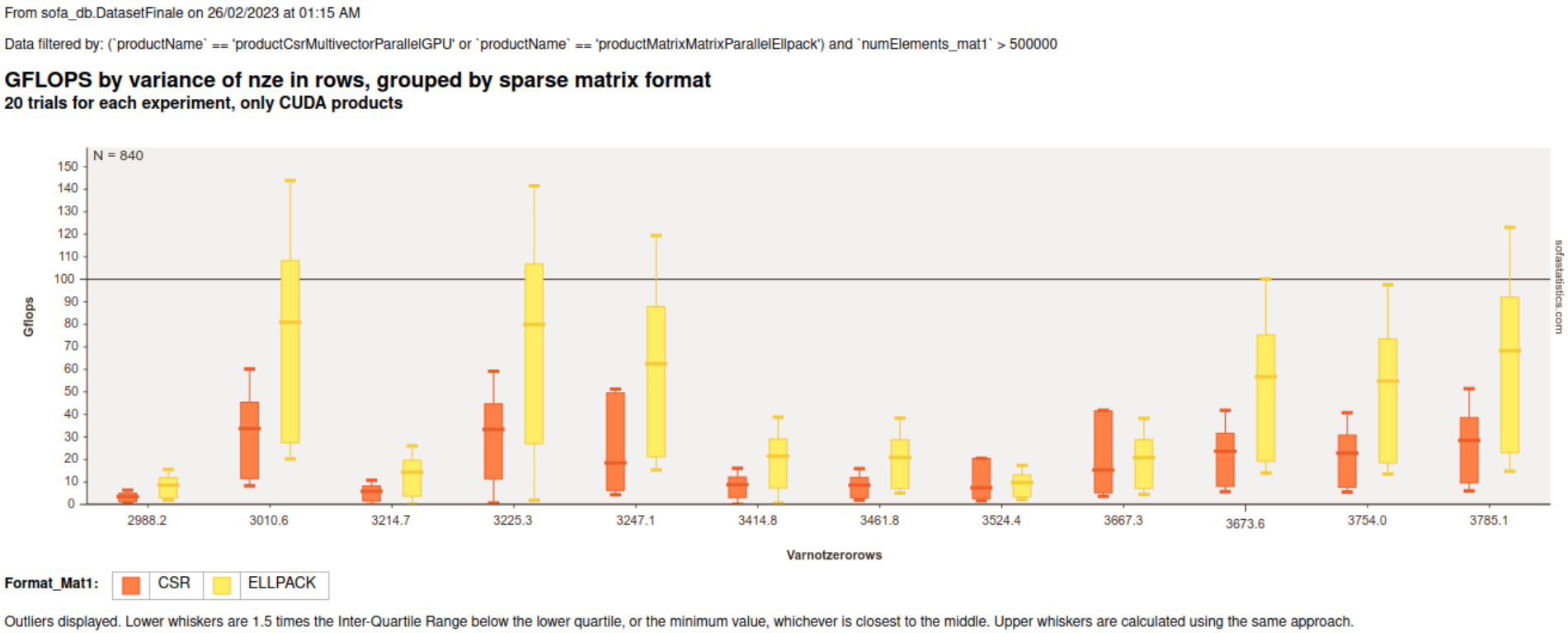

Figure 9: Mostra l'andamento delle prestazioni all'aumentare della varianza

Prestazioni CUDA

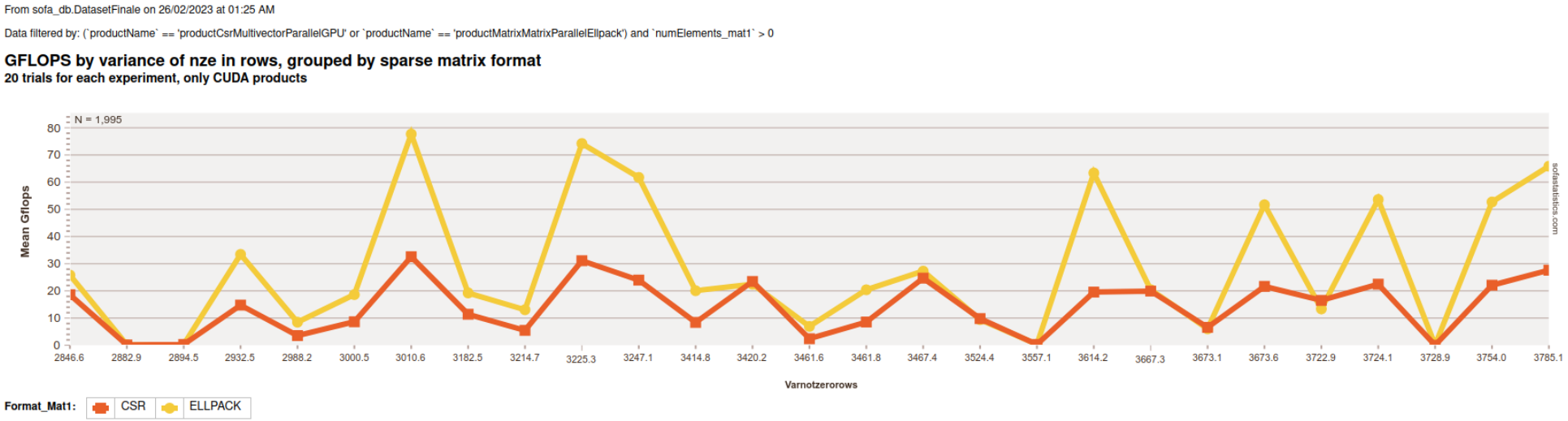

Figure 10: La varianza di elementi non-zero per riga influisce sulle prestazioni di entrambi i formati allo stesso modo, ma con peso in funzione del formato stesso.

Prestazioni OpenMP

Idea

Si valuta la bontà delle implementazioni OpenMP considerando:

- Speedup.
- Incremento delle prestazioni all'aumentare del numero di thread allocati.

Prestazioni OpenMP

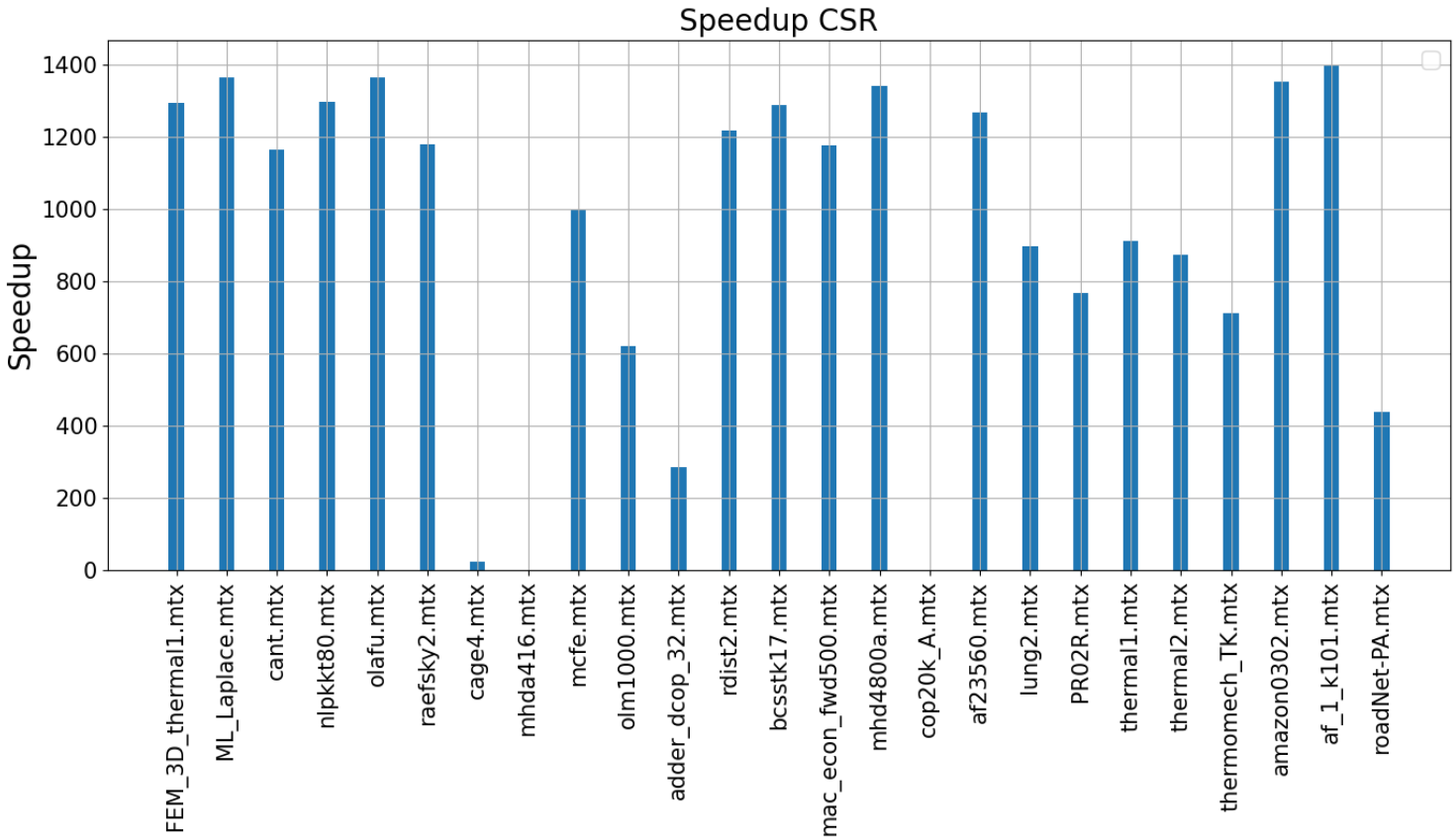

Figure 11: Mostra lo speedup dei GFLOPS ottenuto confrontando le prestazioni con un prodotto seriale

Prestazioni OpenMP

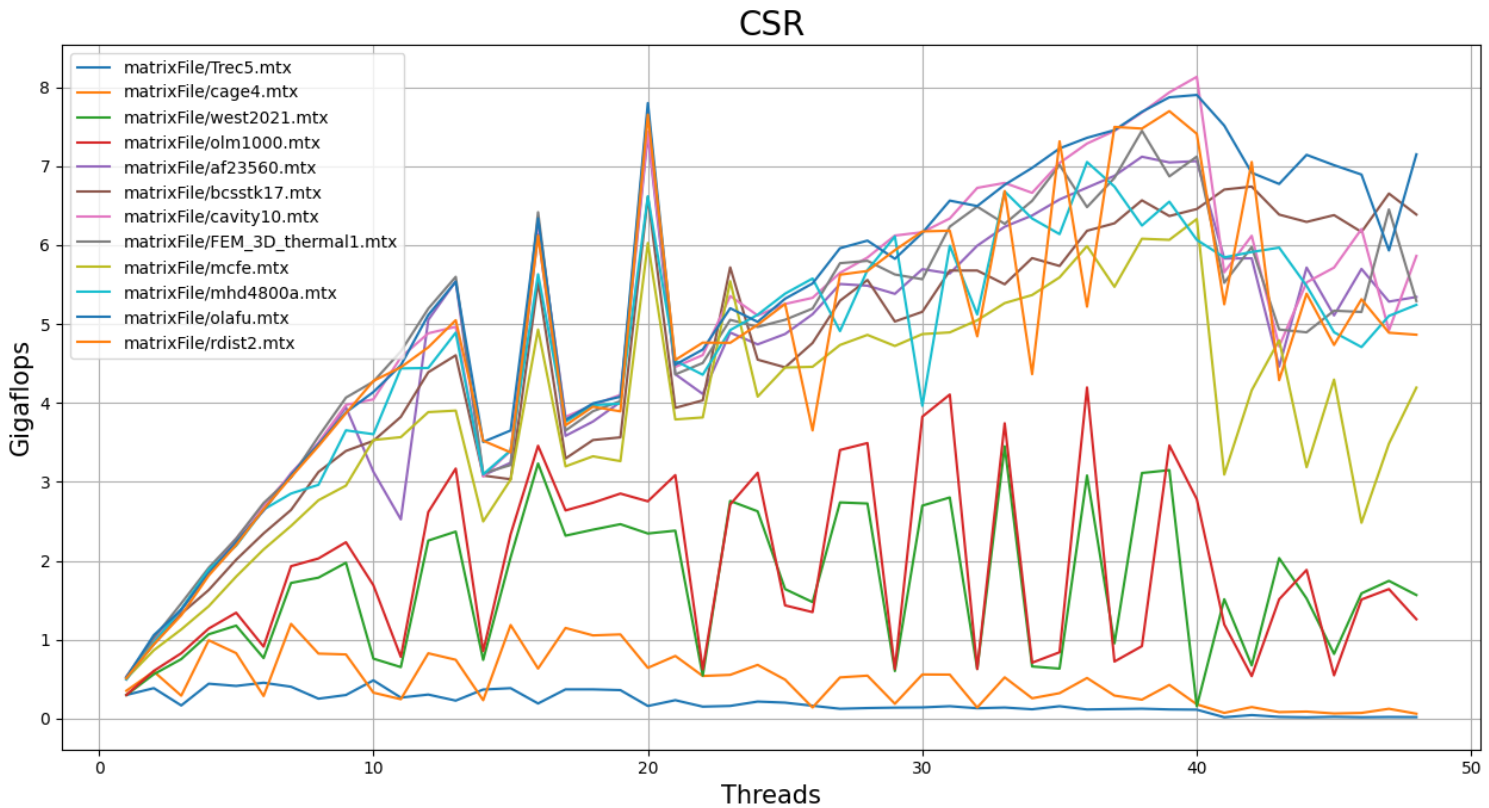

Figure 12: Mostra l'andamento dei GFLOPS all'aumentare dei thread allocati.

Prestazioni OpenMP

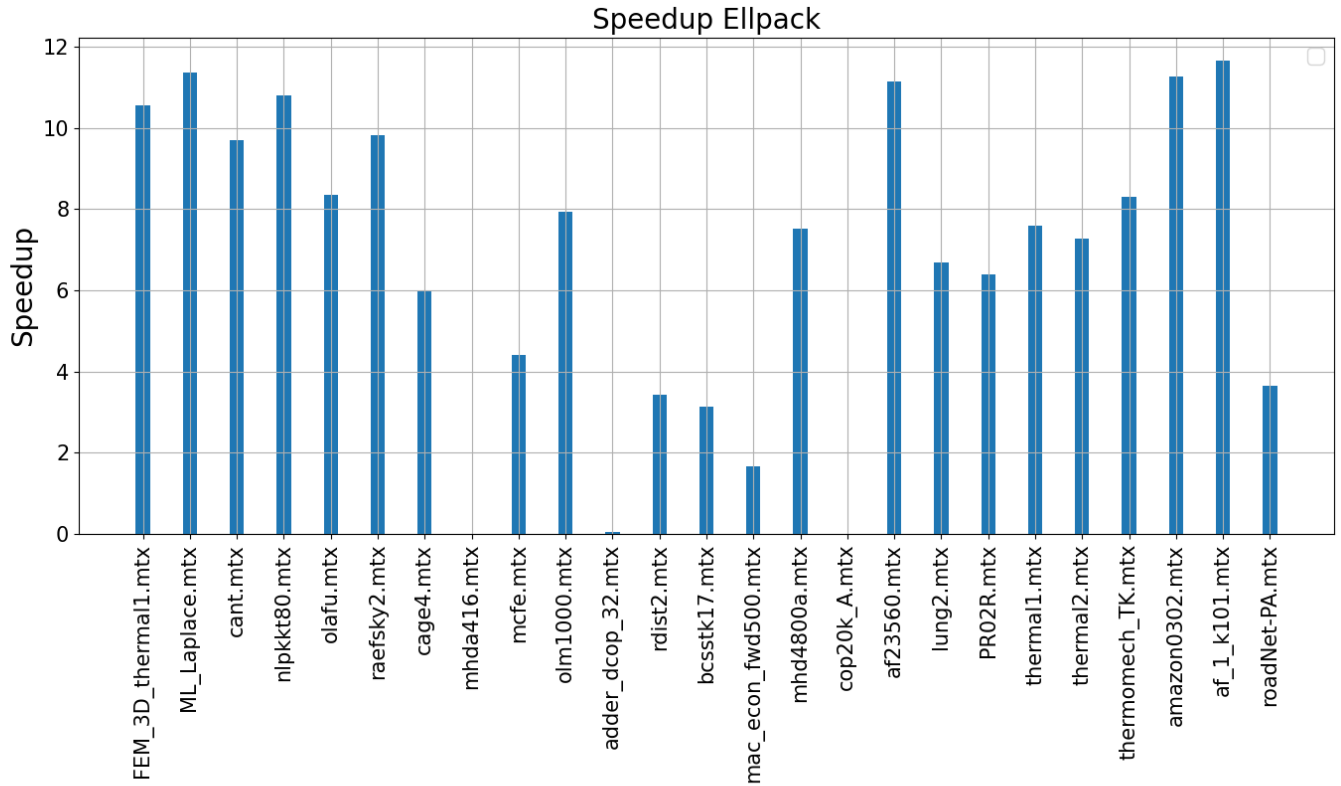

Figure 13: Mostra lo speedup dei GFLOPS ottenuto confrontando le prestazioni con un prodotto seriale

Prestazioni OpenMP

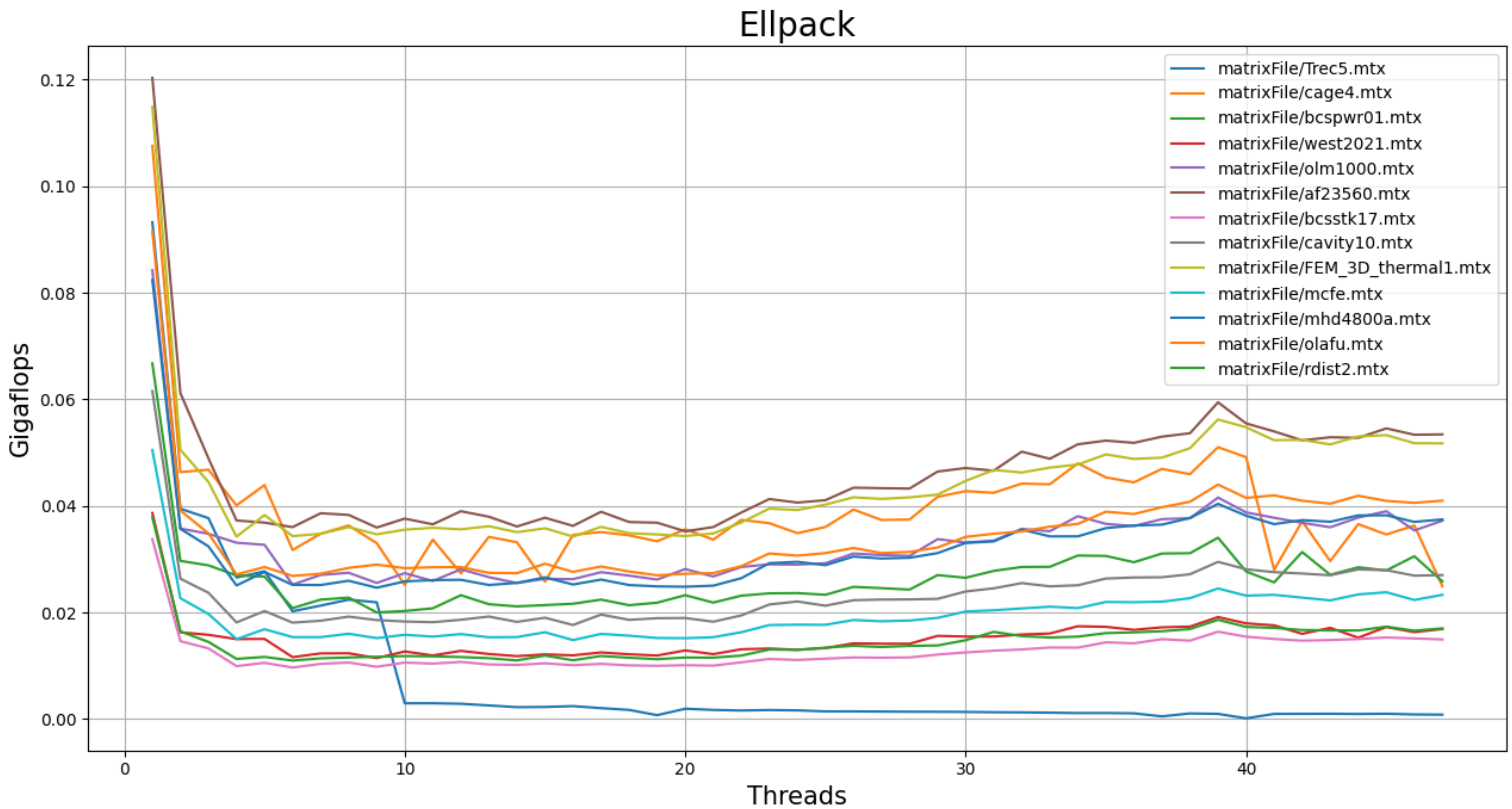

Figure 14: Mostra l'andamento dei GFLOPS all'aumentare dei thread allocati.

Roadmap

Conclusioni

Per concludere

- Come osservato dal modello Roofline ci sono ancora margini di miglioramento delle prestazioni per i prodotti CUDA.
- Le prestazioni dipendono molto delle dimensioni della matrice sparsa e dal numero di colonne del multivettore.
- Per Ellpack il padding influenza notevolmente le prestazioni, la varianza dei non zero nelle righe incide molto poco.
- L'implementazione del prodotto CUDA per Ellpack registra prestazioni leggermente migliori. Ci sono però particolari matrici in cui il prodotto per CSR si comporta meglio.

Conclusioni

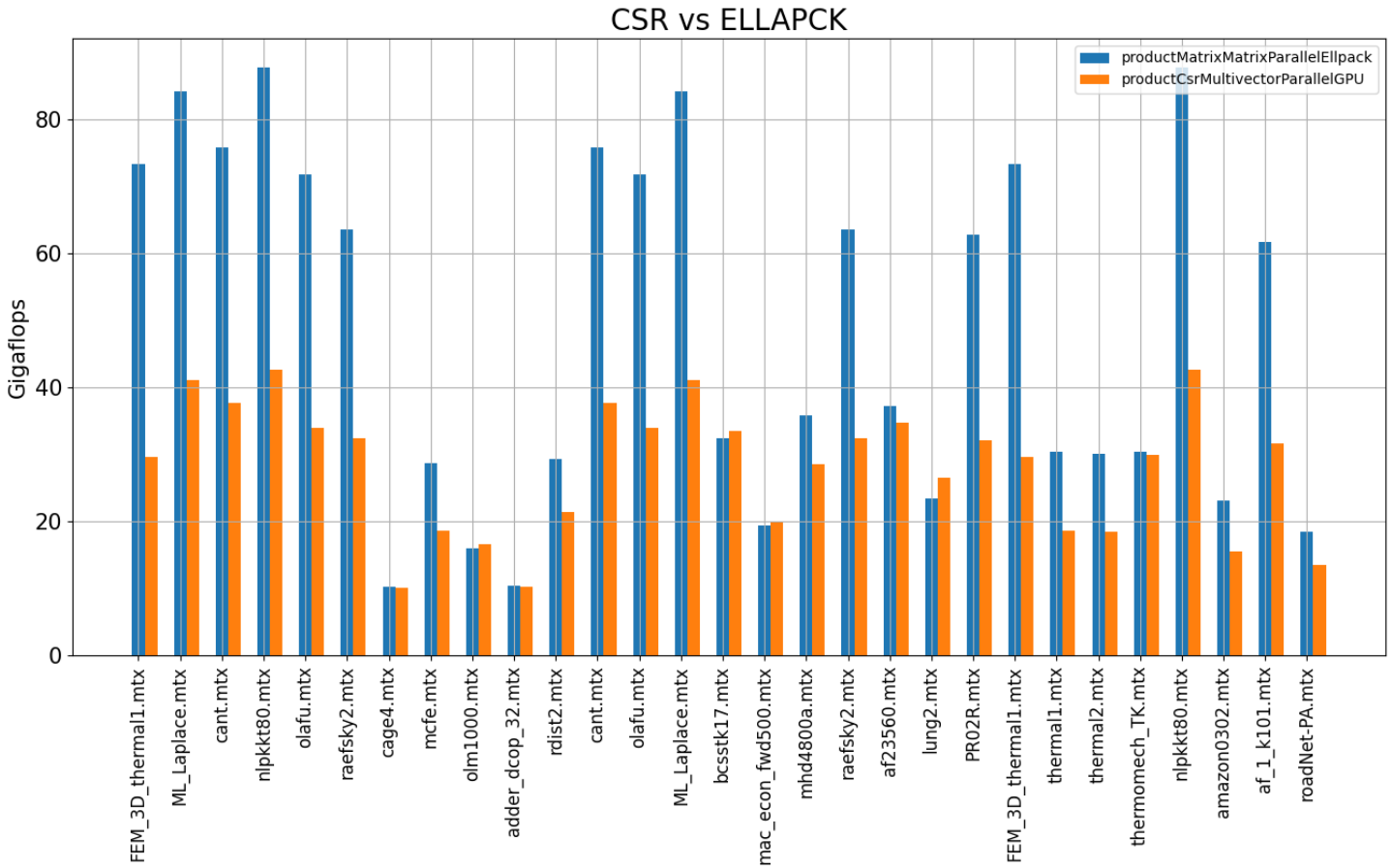

Figure 15: Mostra il confronto tra le prestazioni medie dei prodotto in CUDA per i formati CSR e Ellpack

Domande?

- Sorgenti: https://github.com/scpa-2023/matrix-multiVector-product;
- Consultare README per dettagli build, e la relazione per ulteriori approfondimenti;
- Contattaci a:
  - daniele.laprova@students.uniroma2.eu
  - luca.fiscariello@students.uniroma2.eu
- Fatto con