SpMV - Implementazioni OMP e CUDA

ANDREA DI IORIO

Il progetto consiste in varie implementazioni del nucleo di calcolo $y \leftarrow A\ x$ ovvero il prodotto di una matrice sparsa con un vettore. Questa operazione è essenziale per la soluzione di sistemi lineari sparsi ed è estremamente importante per varie applicazioni di calcolo scientifico.
Nel seguito si farà riferimento a matrici sparse di `M` righe e `N` colonne con in numero massimo di non zeri nelle varie righe pari ad `MAX_NZ_ROW`

Additional Key Words and Phrases: SpMV, CSR 2D partitioning, sparse linear sistems, HPC, scientific computing

Segue una descrizione delle implementazioni realizzate per risolvere il problema in oggetto

1 IMPLEMENTAZIONI OPENMP

1.1 CSR

1.1.1 Partizionamento monodimensionale della matrice.

Semplici implementazioni per SpMV si possono ottenere effettuando una parallelizzazione su (blocchi di) righe della matrice sparsa, delegando ad ogni thread l'accumulazione del prodotto punto tra la riga della matrice assegnatagli e il vettore in ingresso.

```
#pragma omp parallel for schedule(runtime) private(acc,block,startRow,c)
for (ulong b=0;   b<cfg->gridRows;   b++){
    block       = UNIF_REMINDER_DISTRI(b,rowBlock,rowBlockRem);
    startRow    = UNIF_REMINDER_DISTRI_STARTIDX(b,rowBlock,rowBlockRem);
    for (ulong r=startRow;  r<startRow+block;  r++){
        acc = 0;
        #if SIMD_ROWS_REDUCTION == TRUE
        #pragma omp simd reduction(+:acc)
        #endif
        for (ulong j=mat->IRP[r]; j<mat->IRP[r+1]; j++){
           c = mat->JA[j];
           acc += mat->AS[j] * vect[c];
        }
        outVect[r] = acc;
    }
}
```

Nello snippet di codice precedente, appartenente alla funzione `spmvRowsBlocksCSR`, si parallelizza il calcolo di SpMV in blocchi di righe di dimensione pari al numero di righe diviso il valore all'interno del campo `gridRows`.

Questi blocchi hanno una dimensione il più possibile simile per migliorare l'uniformità dei lavori assegnati ai thread e quindi minimizzare il divario nel loro tempo di completamento, nell'ipotesi di avere un carico di lavoro uniforme per ogni blocco di righe. Grazie all'uso delle macro `UNIF_REMINDER_DISTRI` e `UNIF_REMINDER_DISTRI_STARTIDX`, viene ridistribuito il resto della divisione tra il numero di righe della matrice e la variabile `gridRows` tra tutti i thread.

Un altro partizionamento monodimensionale è realizzato dalla funzione `spmvRowsBasicCSR`, dove si suddividono tutte le righe della matrice tra i thread disponibili in base allo scheduling e il chunk configurato nel `#pragma omp parallel for`.

1.1.2 Partizionamento 2D della matrice.

Effettuando un partizionamento bidimensionale della matrice in ingresso è possibile assegnare ad ogni thread un blocco di dati con cui accumulare un prodotto punto parziale tra il vettore e porzioni

di righe.
Successivamente mediante operazioni di riduzione è possibile avere i risulati finali dei prodotti punto e quindi il vettore risultante all'operazione di SpMV.

```
1 #pragma omp parallel for schedule(runtime) private(acc,rowBlock,startRow,c,t_i,t_j
      )
2 for (ulong tileID = 0; tileID < gridSize; tileID++){
3     ///get iteration's indexing variables
4     t_i = tileID/cfg->gridCols;   //i-th row block
5     t_j = tileID%cfg->gridCols;   //j-th col block
6     //get tile row-cols group FAIR sizes
7     rowBlock = UNIF_REMINDER_DISTRI(t_i,_rowBlock,_rowBlockRem);
8     startRow = UNIF_REMINDER_DISTRI_STARTIDX(t_i,_rowBlock,_rowBlockRem);
9
10    for (ulong r=startRow,partOffID;  r<startRow+rowBlock;  r++){
11        partOffID = IDX2D(r,t_j,cfg->gridCols);
12        acc = 0;
13        #if SIMD_ROWS_REDUCTION == TRUE
14        #pragma omp simd reduction(+:acc)
15        #endif
16        //2D CSR tile SpMV using cols partition of row r
17        for (ulong j=offsets[partOffID]; j<offsets[partOffID+1]; j++){
18            c = mat->JA[j];
19            acc += mat->AS[j] * vect[c];
20        }
21        tilesOutTmp[partOffID] = acc;
22    }
23 }
24 for (ulong r=0;  r<mat->M;   r++){
25    for (ulong p=0;   p<cfg->gridCols;   p++){
26        outVect[r] += tilesOutTmp[ IDX2D(r,p,cfg->gridCols) ];
27    }
28 }
```

Nello snippet di codice precedente, appartenete alla funzione `spmvTilesCSR`, si assegna ad ogni thread un blocco della matrice di dimensione pari ad una suddivisone del numero di colonne e di righe rispettivamente per i campi `gridCols` e `gridRows`.
Mediante le variabili `t_i, t_j`, si identifica una partizione CSR delle colonne della matrice ed un blocco di righe al suo interno su cui effettuare i prodotti punti parziali per SpMV, seguiti poi da un operazione di riduzione da riga 23.

1.1.3 metodologie di partizionamento 2D di una matrice CSR.

Il partizionamento bidimensionale di una matrice sparsa in formato CSR è realizzabile mediante una suddivisione delle colonne della matrice che possono essere accedute come sottomatrici CSR salvate separatamente o con riferimenti in una struttura ausiliaria. Quest'ultima opzione è implementata nella funzione `spmvTilesCSR`, dove blocchi della matrice sono accessibili mediante una matrice di indici M x `gridCols` dove l'elemento i,j è l'offset relativo all'inizio della j-esima partizione di colonne nella i-esima riga della matrice CSR.

1.2 ELL

Le righe di una matrice sparsa in formato ELL sono caratterizzate da un insieme di elementi di padding alla fine di ogni riga di elementi non zero così da avere MAX_NNZ_ROWS elementi per ogni riga. I valori di padding inseriti nelle righe sono in valore uguali a zero così da non alterare il risultato finale dell'operazione di SpMV.

1.2.1 Partizionamento monodimensionale della matrice.

Analogamente al caso CSR, nella funzione `spmvRowsBasicCSR`, è possibile effettuare un partizionamento semplice delle sole righe della matrice tra i thread, dove le righe possono essere distribuite automaticamente da openMP, configurando lo scheduling e la dimensione del chunk del costrutto `#pragma omp parallel for` come realizzata dalla funzione `spmvRowsBasicELL`.

Un'altra soluzione, realizzata nella funzione `spmvRowsBlocksELL`, è quella di suddividere le righe in gruppi, ed assegnarle mediante il costrutto `#pragma omp parallel for` ai thread, similmente a come effettuato nella funzione `spmvRowsBlocksCSR`

1.2.2 Partizionamento 2D della matrice.

È possibile effettuare un partizionamento bidimensionale dei dati, assegnando blocchi della matrice ai thread, che accumuleranno prodotti punto parziali, analogamente al caso CSR (1.1.2) e tramite una riduzione finale si avrà il vettore risultante all'operazione di SpMV.
Segue il blocco di codice principale della funzione `spmvTilesELL`, che implementa questa funzione

```
#pragma omp parallel for schedule(runtime) private(acc,...)
for (ulong tileID = 0; tileID < gridSize; tileID++){
    ///get iteration's indexing variables
    t_i = tileID/cfg->gridCols;  //i-th row block
    t_j = tileID%cfg->gridCols;  //j-th col block
    //get tile row-cols group FAIR sizes
    rowBlock = UNIF_REMINDER_DISTRI(t_i,_rowBlock,_rowBlockRem);
    startRow = UNIF_REMINDER_DISTRI_STARTIDX(t_i,_rowBlock,_rowBlockRem);
    colBlock = UNIF_REMINDER_DISTRI(t_j,_colBlock,_colBlockRem);
    startCol = UNIF_REMINDER_DISTRI_STARTIDX(t_j,_colBlock,_colBlockRem);

    for (ulong r=startRow; r<startRow+rowBlock;  r++){
        acc = 0;
        rowPartStart = IDX2D(r,startCol,rMax);
        rowPartEnd   = rowPartStart + colBlock;
        #ifdef ROWLENS
        rowPartEnd   = MIN(rowPartEnd,IDX2D(r,0,rMax) + mat->RL[r]);
        #endif
        #if SIMD_ROWS_REDUCTION == TRUE
        #pragma omp simd reduction(+:acc)
        #endif
        //2D CSR tile SpMV using cols partition of row r
        for (ulong j=rowPartStart; j<rowPartEnd; j++){
            c = mat->JA[j];
            acc += mat->AS[j] * vect[c];
        }
        tilesOutTmp[ IDX2D(r,t_j,cfg->gridCols) ] = acc;
    }
}
for (ulong r=0;  r<mat->M;   r++){
    for (ulong p=0;   p<cfg->gridCols;   p++){
        outVect[r] += tilesOutTmp[ IDX2D(r,p,cfg->gridCols) ];
```

```
33      }
34 }
```

A differenza del caso CSR 1.1.2, il partizionamento 2D è ottenibile semplicemente mediante un indicizzazione differente della matrice, che è appunto salvata in formato matriciale grazie al padding. Definendo la macro `ROWLENS` a compile-time, verrà allocato un vettore contente il numero di non zeri effettivi per ogni riga (simile al vettore IRP per il formato CSR). Questa struttura di supporto diventa particolarmente utile nel caso ELL, dato che permette di interrompere l'accumulazione dei prodotti punti prima di arrivare ai valori di padding (da riga 17).

1.3 Scheduling e chunkSize

La parallelizzazione openMP del codice mediante i vari `#pragma omp parallel for` è configurabile grazie ad uno scheduling `runtime`.
Nel caso di configurare uno scheduling dynamic, la dimensione del chunk verrà adattata al numero totale di iterazioni del for diviso un costante definita in `FAIR_CHUNKS_FOLDING`. Grazie a questo si potrà sovrascrivere il chunkSize pari ad 1 di default di openMP per scheduling dynamic avendo un assegnamento delle iterazioni ragionevole per ogni thread.
Dato il pattern di sparsità variabile della matrice, il carico di lavoro di ogni thread può essere molto variabile con uno scheduling static, soprattutto effettuando un partizionamento dei dati troppo grossolano. Viceversa con uno scheduling dynamic si può avere una migliore distribuzione del carico computazionale in alcuni casi.

2 IMPLEMENTAZIONI CUDA

2.1 partizionamento di 1 riga per thread

Segue una breve descrizione delle implementazioni che assegnano una riga per thread.
La configurazione del kernel è data da un blocco monodimensionale di una dimensione `BLOCKS_1D` una griglia monodimensionale di dimensione tale da coprire tutte le righe della matrice con i blocchi configurati.

2.1.1 CSR.

In `cudaSpMVRowsCSR`, viene assegnato una riga della matrice ad ogni thread e i prodotti punto sono accumulati in maniera simile al caso openMP monodimensionale

2.1.2 ELL.

In `cudaSpMVRowsELL`, viene prima trasposta la matrice e poi assegnata una colonna della matrice risultante ad ogni thread. Successivamente l'accumulazione dei prodotti punto è analoga al caso openMP ELL monodimensionale.
Al fine di migliorare ulteriormente la coalescenza degli accessi in memoria globale sono state usate le funzioni `cudaMallocPitch cudaMemcpy2D`, per aggiungere un ulteriore padding ad ogni riga della matrice per allineare l'inizio di ogni riga ad indirizzi che corrisponderanno agli indirizzi iniziali di transizioni in memoria globale.

La trasposizione della matrice è utile a favorire la coalescenza degli accessi in memoria globale dei thread, dato che ogni thread utilizzerà un valore adiacente al thread a lui precedente nel suo warp

2.2 partizionamento di 1 riga per warp

Per matrici con righe grandi può essere conveniente assegnare un intero warp ad ogni riga. Per facilitare l'indicizzamento dei thread e warp rispetto alla matrice per avere accessi coalizzati in memoria globale ad ogni iterazione ho utilizzato un blocco 2D di dimensioni 32x`BLOCKS_2D_WARP_R`

e una griglia monodimensionale di dimensione tale da coprire tutte le righe con la componente y dei blocchi. In questo modo ogni thread può ricavare il suo indice nel warp nella componente x del suo blocco, mentre può calcolare analogamente al caso precedente la riga a lui assegnatagli nella componente y.
Segue uno snippet di codice relativo alla funzione `cudaSpMVWarpsPerRowELLNTrasposed` dove si assegna ad ogni riga un warp.

```
1 //1D GRID of 2D BLOCKS: [warpIdx,rowIdx]
2 uint tWarpIdx = threadIdx.x;
3 uint tRow = threadIdx.y + blockIdx.y*blockDim.y;
4 ......
5 for(ulong c=tWarpIdx,asIdx=IDX2D(tRow,c,m->pitchAS),jaIdx=IDX2D(tRow,c,m->pitchJA)
      ;
6     c<rLen;  c+=warpSize,asIdx+=warpSize,jaIdx+=warpSize){
7         outEntry += m->AS[asIdx] * v[m->JA[jaIdx]];
8 }
9 outEntry = reduceWarpRegs(outEntry);     //in-reg&intra warp tree reduction
10 if (!tWarpIdx)  outV[tRow] = outEntry;  //0th thread will see the reducted out
```

nel ciclo a riga 5, ogni thread accumula un prodotto punto parziale rispetto alla riga a lui assegnata, accedendo elementi separati dalla variabile definita dal runtime CUDA `warpSize`.
Successivamente si eseguirà una riduzione dei prodotti punto parziali locali ad ogni warp con una somma per avere nel primo thread del warp il valore risultante.

2.3 compilazione

I sorgenti per le implementazioni openMP e CUDA sono accumunati da dei file contenti funzioni `main` che andranno a chiamare una implementazione piuttosto che un'altra. Al fine di avere un unico main che vada a chiamare una implementazione CUDA o openMP, mantendo la possibilità di compilare il codice openMP con `gcc` invece che tramite `nvcc`, ho sfruttato la macro `__CUDACC__`, esportata da `nvcc`, per separare le parti CUDA del codice con degli `#ifdef __CUDACC__`.

3 TESTING

La verifica delle varie implementazioni è realizzata confrontando i vettori risultanti con quelli ottenuti da un'implementazione seriale, mantendo come margine d'errore $7 \cdot 10^{-4}$. Il vettore denso utilizzato è costituito da valori casuali in modulo inferiori a $3 \cdot 10^{-5}$, questo per ridurre i valori risultanti dei vari prodotti punto parziali, riducendo anche l'errore relativo delle operazioni floating-point.

4 ANALISI DELLE PERFORMANCE

Segue una analisi delle performance delle varie implementazioni per il problema in oggetto, ottenuta mediante un analisi dei tempi medi di esecuzione ottenuti tramite uno script di parsing dei log dell'applicazione.
Tutti i tempi di esecuzione utilizzati sono ottenute da medie di 25 iterazioni dell'applicazione in una determinata configurazione.

Nell'analisi delle prestazioni ottenute ho confrontato, oltre che le diverse implementazioni, anche varie configurazioni specifiche alle implementazioni CUDA e openMP. Tutti i log comprendenti le varie configurazioni sono nella cartella `test/log` e tutti i fogli di calcolo usati per effettuare i

confronti sono nella cartella `test/tables`.

Le matrici che ho utilizzato per testare la mia applicazione provengono dalla SuiteSparse Matrix Collection, una raccolta di matrici sparse provenienti da varie applicazioni. Le matrici che ho selezionato per i test sono:
delaunay n12 raefsky1 caidaRouterLevel thermal2 FEM 3D thermal2 west2021 olm1000 consph cavity12 coPapersDBLP delaunay n17 thermomech TK cop20k A delaunay n16 cage4 citationCiteseer mc2depi raefsky2 olafu rdist2 channel-500x100x100-b050 auto asia osm 598a adder dcop 32 delaunay n15 mhd4800a belgium osm chesapeake af 1 k101 lung2 cavity15 adder dcop 31 coPapersCiteseer cavity13 cavity10 hvdc1 roadNet-PA nlpkkt80 PR02R ML Laplace cavity11 mac econ fwd500 mcfe delaunay n14 bcsstk17 mhda416 144 dc3 FEM 3D thermal1 thermal1 coAuthorsCiteseer coAuthorsDBLP Cube Coup dt0 adaptive cant af23560 pdb1HYS dc2 amazon0302 delaunay n13 dc1 webbaseM cavity14

4.1 Limitazioni con implementazioni ELLPACK

Ho implementato il parsing delle matrici mediante un primo parsing dal formato sorgente `MatrixMarket` alla rappresentazione sparsa più simile al formato: COO.
Successivamente vengono effettuate conversione dal formato COO al formato CSR o ELL.

La rappresentazione in formato ELL di alcune matrici, a causa del padding, potrebbe richiedere uno spazio in memoria eccessivo. Per questo motivo ho implementato un controllo iniziale nella conversione COO - ELL, che impedisce la conversione di matrici che abbiano complessivamente tra entry e padding per le matrici AS e JA superiori a $6 \cdot 2^{27}$ (che occuperebbero uno spazio in memoria superiore a circa 6 GB).
Di seguito le matrici che eccedono questa soglia impostata, che nelle tabelle successive avranno uno spazio vuoto al posto del valore per le colonne relative alle implementazioni ELL.

coPapersDBLP coPapersCiteseer dc3 dc2 dc1 webbase-1M

4.2 Implementazioni OpenMP

4.2.1 Configurazioni analizzate. Le implementazioni confrontate sono:

- OMP CSR 1: partizionamento 1D delle righe della matrice in blocchi gestiti dal costrutto `#pragma omp parallel for`
- OMP CSR 2: partizionamento 1D delle righe della matrice in blocchi di dimensione configurata dal parametro `gridRows`, come descritto in 1.1.1
- OMP CSR 3: partizionamento 2D della matrice, con blocchi di dimensione pari a una suddivisione delle righe e delle colonne della matrice, rispettivamente per i parametri `gridRows` `gridCols`. Dove l'accesso alle partizioni delle colonne è realizzato mediante riferimenti una struttura di supporto come descritto in 1.1.2
- OMP CSR 4: partizionamento 2D, come sopra, dove l'accesso alle partizioni delle colonne della matrice è realizzato mediante una allocazione delle partizioni come matrici CSR separate, come descritto in 1.1.3
- OMP ELL 0: partizionamento delle righe della matrice gestito dal costrutto `#pragma omp parallel for` ,come descritto in 1.2.1

- OMP ELL 1: partizionamento delle righe della matrice in blocchi di dimensione configurata dal parametro `gridRows`, come descritto in 1.2.1
- OMP ELL 2: partizionamento 2D della matrice, con blocchi di dimensione pari a una suddivisione delle righe e delle colonne della matrice, rispettivamente per i parametri `gridRows` `gridCols`. Descritto in 1.2.2

Sono state confrontate inoltre,

- riduzioni SIMD nelle fasi di prodotto punto (parziale)
- l'utilizzo del vettore ausiliario `ROWLENS` per determinare la fine di ogni riga,
- l'utilizzo di varie configurazioni di griglia di parallelizzazione con i parametri `gridRows` `gridCols` in valore pari ad: 5x8 10x4 4x10 14x3 13x3
- l'utilizzo di scheduling STATIC e DYNAMIC con un adattamento della dimensione del chunk come descritto in 1.3, con parametro `FAIR_CHUNKS_FOLDING` pari ad 4.

Seguono confronti con tutte le matrici in varie configurazioni di griglia di parallelizzazione

4.2.2 Riduzioni SIMD. Le riduzioni SIMD danno un miglioramento nei tempi di esecuzione nel 42% dei casi, con un miglioramento di almeno 0.003 secondi solo nel 26.41% dei casi, principalmente nelle implementazioni con un partizionamento bidimensionale della matrice.
Questo risultato complessivamente negativo è probabilmente dovuto al fatto che, anche se i valori floating point usati sono contigui, l'accesso del vettore denso con indicizzamento non contiguo impedisce una effettiva ottimizzazione del codice mediante OpenMP.

4.2.3 Vettore ausiliario ROWLENS. L'uso di una precomputazione della lunghezza delle righe da un vantaggio nel 61.86% dei casi, nella maggior parte dei casi per le implementazioni ELL.

4.2.4 configurazioni di griglia di parallelizzazione. Questa analisi ha portato risultati diversi in base all'implementazione e alla matrice in uso, questo è probabilmente dovuto al fatto che il partizionamento (bidimensionale) della matrice può dare benefici in configurazioni diverse in base al pattern di sparsità dei non zeri della matrice.

4.2.5 scheduling Static vs Dynamic. Confrontando la griglia di parallelizzazione 10x4, ho identificato che uno scheduling Static ha dato risultati migliori dello scheduling Dynamic, con partizionamento `FAIR_CHUNKS_FOLDING` pari ad 4 nel 64.88% dei casi, con un miglioramento di almeno 0.0003 secondi nel 26.74%

Segue un Confronto delle varie implementazioni nella configurazione senza riduzioni SIMD, con il vettore ausiliario ROWLENS e una griglia di parallelizzazione pari ad 10x4:

source	OMP CSR 1	OMP CSR 2	OMP CSR 3	OMP CSR 4	OMP ELL 0	OMP ELL 1	OMP ELL 2	MIN TIME	BEST GFLOPS	BEST IMPL
west2021.mtx	6.17E-05	1.67E-05	6.90E-05	2.19E-04	1.68E-05	1.84E-05	3.83E-05	**1.67E-05**	0.878133193685514	**OMP CSR 2**
webbase-1M.mtx	2.75E-03	3.26E-03	3.39E-02	6.63E-02				**2.75E-03**	0.00534600932826	**OMP CSR 1**
thermomech_TK.mtx	1.54E-04	2.67E-04	4.29E-03	1.04E-02	1.40E-04	4.60E-04	1.13E-03	**1.40E-04**	0.105135451436873	**OMP ELL 0**
thermal2.mtx	5.35E-03	5.68E-03	5.38E-02	1.27E-01	7.76E-03	6.07E-03	3.04E-02	**5.35E-03**	0.002748465505506	**OMP CSR 1**
thermal1.mtx	1.30E-04	1.68E-04	2.51E-03	8.12E-03	9.96E-05	2.18E-04	8.14E-04	**9.96E-05**	0.147602498221965	**OMP ELL 0**
roadNet-PA.mtx	1.97E-03	2.47E-03	3.68E-02	9.43E-02	3.95E-03	3.77E-03	2.08E-02	**1.97E-03**	0.007459438731256	**OMP CSR 1**
rdist2.mtx	7.47E-05	3.04E-05	1.24E-04	4.22E-04	2.28E-05	3.42E-05	5.55E-05	**2.28E-05**	0.643823328154202	**OMP ELL 0**
raefsky2.mtx	8.88E-05	8.19E-05	3.16E-04	2.88E-03	4.14E-05	8.66E-05	6.20E-05	**4.14E-05**	149.985317981609	**OMP ELL 0**
raefsky1.mtx	3.21E-04	8.64E-05	3.15E-04	2.85E-03	4.34E-05	9.15E-05	6.82E-05	**4.34E-05**	143.127032467868	**OMP ELL 0**
PR02R.mtx	4.38E-03	3.97E-03	1.73E-02	5.72E-02	6.48E-03	6.76E-03	1.76E-02	**3.97E-03**	1.56598728163795	**OMP CSR 2**
pdb1HYS.mtx	1.80E-03	1.73E-03	8.20E-03	3.25E-02	1.77E-03	1.98E-03	4.03E-03	**1.73E-03**	3.5892896975693	**OMP CSR 2**
olm1000.mtx	6.34E-05	1.76E-05	4.17E-05	1.88E-04	2.27E-05	1.77E-05	2.69E-05	**1.76E-05**	81.0591626695738	**OMP CSR 2**
olafu.mtx	7.63E-04	2.35E-04	1.21E-03	4.80E-03	1.35E-04	3.26E-04	6.80E-04	**1.35E-04**	10.509208284687	**OMP ELL 0**
nlpkkt80.mtx	1.57E-02	1.47E-02	8.34E-02	2.33E-01	9.09E-03	1.04E-02	3.75E-02	**9.09E-03**	0.156634217153054	**OMP ELL 0**
ML_Laplace.mtx	1.40E-02	1.32E-02	5.21E-02	1.98E-01	1.32E-02	1.32E-02	3.66E-02	**1.32E-02**	0.108196160451787	**OMP CSR 2**
mhda416.mtx	6.23E-05	1.94E-05	3.37E-05	1.31E-04	1.96E-05	1.86E-05	2.40E-05	**1.86E-05**	76.4435534814165	**OMP ELL 1**
mhd4800a.mtx	7.44E-05	4.49E-05	1.95E-04	1.21E-03	2.80E-05	4.77E-05	6.87E-05	**2.80E-05**	50.8518327679223	**OMP ELL 0**
mcfe.mtx	7.71E-05	2.06E-05	5.35E-05	3.27E-04	3.47E-05	2.18E-05	2.49E-05	**2.06E-05**	69.1056764496077	**OMP CSR 2**
mc2depi.mtx	9.80E-04	1.07E-03	1.79E-02	3.57E-02	5.68E-04	9.95E-04	6.90E-03	**5.68E-04**	30.2215532395936	**OMP ELL 0**
mac_econ_fwd500.mtx	9.78E-04	3.96E-04	5.80E-03	1.86E-02	2.85E-03	3.75E-03	6.75E-03	**3.96E-04**	43.3407257387291	**OMP CSR 2**
lung2.mtx	1.15E-04	1.38E-04	2.59E-03	1.49E-02	5.68E-04	1.83E-04	1.03E-03	**1.15E-04**	148.887126678715	**OMP CSR 1**
hvdc1.mtx	8.66E-05	7.38E-05	8.00E-04	2.91E-03	4.75E-05	1.50E-04	2.98E-04	**4.75E-05**	361.003797974834	**OMP ELL 0**
FEM_3D_thermal2.mtx	1.64E-03	1.82E-03	1.10E-02	3.17E-02	1.36E-03	1.92E-03	6.25E-03	**1.36E-03**	12.6607442711484	**OMP ELL 0**
FEM_3D_thermal1.mtx	4.67E-04	1.12E-04	6.60E-04	5.16E-03	5.71E-05	1.23E-04	2.16E-04	**5.71E-05**	300.614222142653	**OMP ELL 0**
delaunay_n17.mtx	7.62E-04	3.29E-04	5.28E-03	1.96E-02	3.40E-04	8.70E-04	2.69E-03	**3.29E-04**	52.2147750252088	**OMP CSR 2**
delaunay_n16.mtx	5.12E-04	1.46E-04	2.55E-03	9.93E-03	1.09E-04	2.11E-04	7.78E-04	**1.09E-04**	10.5400786206074	**OMP ELL 0**
delaunay_n15.mtx	1.19E-04	6.85E-05	1.25E-03	3.85E-03	4.84E-05	9.71E-05	3.34E-04	**4.84E-05**	23.7234547344451	**OMP ELL 0**
delaunay_n14.mtx	7.08E-05	3.90E-05	6.42E-04	3.15E-03	2.59E-05	4.72E-05	1.57E-04	**2.59E-05**	44.3726945463941	**OMP ELL 0**
delaunay_n13.mtx	6.40E-05	2.49E-05	3.37E-04	1.56E-03	1.88E-05	2.76E-05	9.14E-05	**1.88E-05**	61.0782129409157	**OMP ELL 0**
delaunay_n12.mtx	6.73E-05	2.04E-05	1.96E-04	8.26E-04	1.69E-05	2.06E-05	6.12E-05	**1.69E-05**	68.1406040937171	**OMP ELL 0**
dc3.mtx	1.06E-03	5.08E-04	3.08E-03	1.02E-02				**5.08E-04**	2.26360671912813	**OMP CSR 2**
dc2.mtx	4.70E-04	4.89E-04	3.08E-03	1.04E-02				**4.70E-04**	2.44358812171958	**OMP CSR 1**
dc1.mtx	4.71E-04	4.96E-04	3.12E-03	1.04E-02				**4.71E-04**	13.0871174183911	**OMP CSR 1**
Cube_Coup_dt0.mtx	6.66E-02	6.32E-02	2.43E-01	8.41E-01	4.94E-02	4.90E-02	2.08E-01	**4.90E-02**	0.125892373271085	**OMP ELL 1**
coPapersDBLP.mtx	1.62E-02	1.79E-02	8.90E-02	2.25E-01				**1.62E-02**	0.37990228326028	**OMP CSR 1**
coPapersCiteseer.mtx	1.68E-02	2.24E-02	8.37E-02	2.39E-01				**1.68E-02**	0.366850421239713	**OMP CSR 1**
cop20k_A.mtx	1.26E-03	1.80E-03	1.19E-02	2.75E-02	2.89E-03	3.72E-03	5.52E-03	**1.26E-03**	4.87650354969374	**OMP CSR 1**
consph.mtx	2.71E-03	2.33E-03	1.42E-02	4.27E-02	2.73E-03	2.67E-03	8.13E-03	**2.33E-03**	2.65122936972395	**OMP CSR 2**
coAuthorsDBLP.mtx	1.27E-03	2.13E-03	1.95E-02	4.07E-02	3.77E-03	4.33E-03	7.68E-03	**1.27E-03**	4.87219670188605	**OMP CSR 1**
coAuthorsCiteseer.mtx	9.23E-04	1.44E-03	1.49E-02	3.30E-02	2.66E-03	4.60E-03	9.44E-03	**9.23E-04**	0.123332124938168	**OMP CSR 1**
citationCiteseer.mtx	9.14E-04	2.09E-03	2.12E-02	4.36E-02	5.99E-03	4.90E-03	8.20E-03	**9.14E-04**	0.124566588284715	**OMP CSR 1**
chesapeake.mtx	2.03E-04	1.63E-05	1.69E-05	2.84E-05	1.54E-05	1.45E-05	1.57E-05	**1.45E-05**	7.87572909008036	**OMP ELL 1**
channel-500x100x100-b050.mtx	4.88E-02	5.10E-02	2.24E-01	7.80E-01	2.81E-02	2.86E-02	1.16E-01	**2.81E-02**	0.004054507322597	**OMP ELL 0**
cavity15.mtx	6.83E-05	3.20E-05	1.29E-04	8.29E-04	2.13E-05	3.46E-05	4.81E-05	**2.13E-05**	5.33601005457475	**OMP ELL 0**
cavity14.mtx	6.91E-05	3.08E-05	1.96E-04	8.55E-04	2.22E-05	3.41E-05	5.05E-05	**2.22E-05**	5.121345686786	**OMP ELL 0**
cavity13.mtx	6.98E-05	3.54E-05	1.32E-04	8.30E-04	2.29E-05	3.77E-05	5.03E-05	**2.29E-05**	4.97545437266096	**OMP ELL 0**
cavity12.mtx	6.97E-05	3.20E-05	1.37E-04	8.38E-04	2.16E-05	3.52E-05	4.80E-05	**2.16E-05**	27.1997626412264	**OMP ELL 0**
cavity11.mtx	7.29E-05	3.25E-05	1.33E-04	8.44E-04	2.08E-05	3.51E-05	5.02E-05	**2.08E-05**	28.2882891543101	**OMP ELL 0**
cavity10.mtx	2.52E-04	3.74E-05	1.35E-04	8.33E-04	2.24E-05	3.97E-05	5.01E-05	**2.24E-05**	26.2755108874132	**OMP ELL 0**
cant.mtx	1.27E-03	1.76E-03	8.77E-03	3.17E-02	1.11E-03	1.71E-03	4.21E-03	**1.11E-03**	0.532001323332411	**OMP ELL 0**
caidaRouterLevel.mtx	6.31E-04	1.76E-03	1.08E-02	2.20E-02	1.65E-03	3.74E-03	4.89E-03	**6.31E-04**	0.932336120167429	**OMP CSR 1**
cage4.mtx	7.70E-05	1.59E-05	1.66E-05	2.30E-05	1.46E-05	1.49E-05	1.54E-05	**1.46E-05**	40.3003531875208	**OMP ELL 0**
belgium_osm.mtx	2.96E-03	2.97E-03	4.69E-02	1.14E-01	5.11E-03	5.39E-03	2.24E-02	**2.96E-03**	0.199045949003421	**OMP CSR 1**
bcsstk17.mtx	1.04E-04	1.11E-04	5.93E-04	2.29E-03	8.33E-05	2.24E-04	2.46E-04	**8.33E-05**	7.06526188617782	**OMP ELL 0**
auto.mtx	3.74E-03	4.77E-03	3.61E-02	7.40E-02	4.78E-03	7.99E-03	1.73E-02	**3.74E-03**	0.157560889159216	**OMP CSR 1**
asia_osm.mtx	3.09E-02	3.16E-02	3.53E-01	9.32E-01	6.80E-02	6.57E-02	1.93E-01	**3.09E-02**	0.019038272118108	**OMP CSR 1**
amazon0302.mtx	3.03E-04	7.36E-04	1.09E-02	3.15E-02	2.88E-04	9.58E-04	3.38E-03	**2.88E-04**	2.04262018117846	**OMP ELL 0**
af23560.mtx	1.02E-04	1.10E-04	7.75E-04	6.32E-03	5.61E-05	1.57E-04	2.41E-04	**5.61E-05**	10.485272922594	**OMP ELL 0**
af_1_k101.mtx	9.15E-03	8.23E-03	4.56E-02	1.33E-01	8.57E-03	8.42E-03	3.15E-02	**8.23E-03**	0.071496616935657	**OMP CSR 2**
adder_dcop_32.mtx	6.62E-05	1.98E-05	9.37E-05	2.57E-04	2.08E-05	2.13E-05	3.61E-05	**1.98E-05**	29.7925136028647	**OMP CSR 2**
adder_dcop_31.mtx	6.77E-05	1.94E-05	9.24E-05	2.58E-04	2.10E-05	2.00E-05	3.58E-05	**1.94E-05**	843.083584915183	**OMP CSR 2**
adaptive.mtx	2.97E-02	3.02E-02	2.32E-01	6.06E-01	2.10E-02	2.24E-02	1.08E-01	**2.10E-02**	0.780947301483107	**OMP ELL 0**
598a.mtx	3.44E-04	8.48E-04	1.04E-02	2.26E-02	8.11E-04	1.44E-03	3.43E-03	**3.44E-04**	47.6309925092271	**OMP CSR 1**
144.mtx	9.48E-04	1.16E-03	1.22E-02	2.82E-02	1.42E-03	1.67E-03	5.21E-03	**9.48E-04**	17.2656623428345	**OMP CSR 1**

4.3 Implementazioni CUDA

4.3.1 Configurazioni analizzate. Le implementazioni confrontate sono:

- CUDA CSR 0: assegnamento di una riga CSR per thread
- CUDA CSR 1: assegnamento di una riga CSR per warp
- CUDA ELL 0: trasposizione della matrice ELL e assegnamento di una colonna per thread, come descritto in 2.1.2
- CUDA ELL 1: assegnamento di una riga per thread, senza trasposizione
- CUDA ELL 2: assegnamento di una riga per warp senza trasposizione, come descritto in 2.2

Sono state confrontate inoltre:

- l'utilizzo del vettore ausiliario `ROWLENS` per determinare la fine di ogni riga,
- le seguenti configurazioni di blocchi per le implementazioni con assegnazione di 1 riga per thread:
  192 256 384
- le seguenti configurazioni di blocchi per le implementazioni con assegnazione di 1 riga per warp:
  32x8 32x16 32x32

Seguono confronti con tutte le matrici in tutte le configurazioni dei blocchi

4.3.2 Vettore ausiliario ROWLENS. L'uso del vettore ausiliario ha creato uno svantaggio nel 55.96% dei casi, senza dare significativi cambiamenti ai tempi di esecuzione.
Questo è probabilmente dovuto a questioni di divergenza tra i thread all'interno dello stesso warp.

4.3.3 Configurazioni dei blocchi per il lancio dei kernel. Le performance migliori sono state ottenute rispettivamente dalle configurazioni:

- 192 e 32x8 nel 58.09% dei casi
- 384 e 32x16 nel 34.32% dei casi
- 256 e 32x32 nel 6.60% dei casi

Segue un'analisi delle performance delle varie implementazioni CUDA, senza il vettore ausiliario ROWLENS e con la configurazione di blocchi migliore:

source	NNZ	TimeAvg CSR 0	TimeAvg CSR 1	timeAvg ELL 0	timeAvg ELL 1	TimeAvg ELL 2	GFLOPS MAX	BEST IMPLEMENTATION
delaunay_n12.mtx	24528	9.20E-06	1.39E-05	8.44E-06	1.08E-05	1.38E-05	5.81	GFLOPS ELL 0
raefsky1.mtx	294276	4.24E-05	1.26E-05	2.28E-05	3.92E-05	1.72E-05	46.56	GFLOPS CSR 1
caidaRouterLevel.mtx	1218132	2.33E-04	7.49E-04	8.19E-03	8.00E-02	1.66E-03	10.47	GFLOPS CSR 0
thermal2.mtx	8580313	8.66E-04	2.11E-03	6.52E-04	2.13E-03	2.11E-03	26.34	GFLOPS ELL 0
FEM_3D_thermal2.mtx	3489300	5.26E-04	2.66E-04	1.78E-04	4.48E-04	2.64E-04	39.14	GFLOPS ELL 0
west2021.mtx	7353	8.96E-06	1.02E-05	7.44E-06	9.96E-06	1.07E-05	1.98	GFLOPS ELL 0
olm1000.mtx	3996	7.68E-06	8.76E-06	7.64E-06	8.24E-06	8.52E-06	1.05	GFLOPS ELL 0
consph.mtx	6010480	1.30E-03	1.88E-04	2.88E-04	1.25E-03	1.91E-04	63.97	GFLOPS CSR 1
cavity12.mtx	76367	1.59E-05	1.16E-05	1.25E-05	2.43E-05	1.31E-05	13.17	GFLOPS CSR 1
coPapersDBLP.mtx	30491458	6.78E-03	8.46E-04				72.11	GFLOPS CSR 1
delaunay_n17.mtx	786352	7.55E-05	2.36E-04	1.08E-04	1.96E-04	2.34E-04	20.84	GFLOPS CSR 0
thermomech_TK.mtx	711558	7.22E-05	1.87E-04	5.55E-05	1.29E-04	2.36E-04	25.65	GFLOPS ELL 0
cop20k_A.mtx	2624331	4.18E-04	2.74E-04	4.22E-04	2.38E-03	2.52E-04	20.83	GFLOPS ELL 2
delaunay_n16.mtx	393150	4.14E-05	1.13E-04	5.62E-05	1.01E-04	1.13E-04	19.01	GFLOPS CSR 0
cage4.mtx	49	7.16E-06	6.40E-06	6.88E-06	6.28E-06	6.32E-06	0.02	GFLOPS ELL 1
citationCiteseer.mtx	2313294	4.28E-04	4.73E-04	1.42E-02	1.47E-01	2.56E-03	10.82	GFLOPS CSR 0
mc2depi.mtx	2100225	1.34E-04	8.54E-04	1.19E-04	2.32E-04	9.08E-04	35.23	GFLOPS ELL 0
raefsky2.mtx	294276	4.30E-05	1.27E-05	2.29E-05	3.95E-05	1.69E-05	46.42	GFLOPS CSR 1
olafu.mtx	1015156	1.18E-04	3.76E-05	6.53E-05	1.08E-04	4.29E-05	53.94	GFLOPS CSR 1
rdist2.mtx	56934	1.39E-05	1.22E-05	1.24E-05	2.40E-05	1.44E-05	9.3	GFLOPS CSR 1
channel-500x100x100-b050	85362744	1.26E-02	8.21E-03	3.86E-03	9.04E-03	6.90E-03	44.25	GFLOPS ELL 0
auto.mtx	6629222	1.03E-03	7.78E-04	7.58E-04	2.86E-03	8.13E-04	17.48	GFLOPS ELL 0
asia_osm.mtx	25423206	2.13E-03	2.04E-02	5.32E-03	1.14E-02	2.04E-02	23.87	GFLOPS CSR 0
598a.mtx	1483868	2.01E-04	1.68E-04	1.33E-04	3.82E-04	1.67E-04	22.31	GFLOPS ELL 0
adder_dcop_32.mtx	11246	1.02E-04	9.44E-06	1.35E-04	3.54E-04	3.04E-05	2.38	GFLOPS CSR 1
delaunay_n15.mtx	196548	1.37E-05	5.44E-05	3.26E-05	6.22E-05	5.41E-05	28.65	GFLOPS CSR 0
mhd4800a.mtx	102252	1.34E-05	1.30E-05	8.80E-06	1.99E-05	1.46E-05	23.24	GFLOPS ELL 0
belgium_osm.mtx	3099940	2.87E-04	2.48E-03	7.22E-04	2.07E-03	2.17E-03	21.59	GFLOPS CSR 0
chesapeake.mtx	340	8.72E-06	6.32E-06	7.68E-06	7.96E-06	6.52E-06	0.11	GFLOPS CSR 1
af_1_k101.mtx	17550675	3.05E-03	8.23E-04	7.55E-04	2.14E-03	9.12E-04	46.47	GFLOPS ELL 0
lung2.mtx	492564	4.28E-05	1.66E-04	4.86E-05	5.78E-05	1.65E-04	23.02	GFLOPS CSR 0
cavity15.mtx	76367	1.38E-05	1.04E-05	1.09E-05	2.08E-05	1.10E-05	14.74	GFLOPS CSR 1
adder_dcop_31.mtx	11246	1.02E-04	9.00E-06	1.35E-04	3.58E-04	3.04E-05	2.5	GFLOPS CSR 1
coPapersCiteseer.mtx	32073440	7.52E-03	9.43E-04				68.02	GFLOPS CSR 1
cavity13.mtx	76367	1.60E-05	1.18E-05	1.25E-05	2.43E-05	1.28E-05	12.99	GFLOPS CSR 1
cavity10.mtx	76367	1.61E-05	1.16E-05	1.28E-05	2.44E-05	1.29E-05	13.12	GFLOPS CSR 1
hvdc1.mtx	159981	2.80E-05	5.05E-05	1.86E-04	5.63E-04	8.46E-05	11.41	GFLOPS CSR 0
roadNet-PA.mtx	3083796	2.20E-04	1.88E-03	4.71E-04	9.77E-04	1.87E-03	27.97	GFLOPS CSR 0
nlpkkt80.mtx	28704672	3.91E-03	1.81E-03	1.35E-03	2.05E-03	1.52E-03	42.45	GFLOPS ELL 0
PR02R.mtx	8185136	1.67E-03	3.12E-04	6.17E-04	3.20E-03	2.99E-04	54.75	GFLOPS ELL 2
ML_Laplace.mtx	27689972	5.61E-03	6.68E-04	1.17E-03	7.79E-03	6.83E-04	82.94	GFLOPS CSR 1
cavity11.mtx	76367	1.41E-05	1.04E-05	1.08E-05	2.07E-05	1.10E-05	14.63	GFLOPS CSR 1
mac_econ_fwd500.mtx	1273389	1.14E-04	3.49E-04	3.85E-04	6.68E-04	4.54E-04	22.29	GFLOPS CSR 0
mcfe.mtx	24382	1.84E-05	8.04E-06	1.33E-05	2.96E-05	8.48E-06	6.07	GFLOPS CSR 1
delaunay_n14.mtx	98244	1.11E-05	3.58E-05	1.62E-05	2.47E-05	3.55E-05	17.73	GFLOPS CSR 0
bcsstk17.mtx	428650	4.38E-05	2.81E-05	7.36E-05	1.78E-04	3.71E-05	30.53	GFLOPS CSR 1
mhda416.mtx	8562	1.22E-05	7.84E-06	8.92E-06	1.56E-05	7.84E-06	2.18	GFLOPS CSR 1
144.mtx	2148786	3.03E-04	2.60E-04	1.74E-04	4.18E-04	2.58E-04	24.7	GFLOPS ELL 0
dc3.mtx	766396	1.29E-02	1.76E-04				8.73	GFLOPS CSR 1
FEM_3D_thermal1.mtx	430740	4.74E-05	3.21E-05	2.70E-05	3.64E-05	3.22E-05	31.91	GFLOPS ELL 0
thermal1.mtx	574458	6.16E-05	1.27E-04	4.83E-05	1.40E-04	1.27E-04	23.78	GFLOPS ELL 0
coAuthorsCiteseer.mtx	1628268	3.26E-04	4.03E-04	1.23E-02	1.30E-01	2.15E-03	10	GFLOPS CSR 0
coAuthorsDBLP.mtx	1955352	2.34E-04	4.37E-04	4.08E-03	3.90E-02	9.78E-04	16.7	GFLOPS CSR 0
Cube_Coup_dt0.mtx	127206144	1.75E-02	3.09E-03	6.15E-03	4.03E-02	3.87E-03	82.43	GFLOPS CSR 1
adaptive.mtx	27248640	2.09E-03	1.16E-02	1.95E-03	3.60E-03	1.09E-02	27.96	GFLOPS ELL 0
cant.mtx	4007383	8.80E-04	1.36E-04	2.08E-04	8.22E-04	1.45E-04	59.04	GFLOPS CSR 1
af23560.mtx	484256	5.96E-05	4.90E-05	2.79E-05	4.71E-05	4.86E-05	34.74	GFLOPS ELL 0
pdb1HYS.mtx	4344765	1.04E-03	8.92E-05	3.06E-04	8.16E-04	1.21E-04	97.42	GFLOPS CSR 1
dc2.mtx	766396	1.22E-02	1.76E-04				8.71	GFLOPS CSR 1
amazon0302.mtx	1234877	9.28E-05	3.84E-04	7.61E-05	2.61E-04	3.83E-04	32.45	GFLOPS ELL 0
delaunay_n13.mtx	49094	8.40E-06	1.80E-05	7.44E-06	1.18E-05	1.80E-05	13.2	GFLOPS ELL 0
dc1.mtx	766396	1.31E-02	1.76E-04				8.69	GFLOPS CSR 1
webbaseM.mtx	3105536	7.87E-04	1.72E-03				7.89	GFLOPS CSR 0
cavity14.mtx	76367	1.61E-05	1.19E-05	1.24E-05	2.44E-05	1.28E-05	12.86	GFLOPS CSR 1

SUMMARY		
% CSR 0 BETTER	% ELL0 BETTER	MAX GFLOPS
23.44%	31.25%	81.06
% CSR1 BETTER	% ELL1 BETTER	
40.32%	1.61%	
	% ELL2 BETTER	
	3.33%	

Fig. 2. performance implementazioni CUDA