Confronto tra versione sequenziale e versioni parallelizzate con CUDA e OpenMP di convoluzioni

Silvia Dani
Indirizzo e-mail
silvia.dani@stud.unifi.it

Niccolò Niccoli
Indirizzo e-mail
niccolo.niccoli@stud.unifi.it

Abstract

Questo esperimento vuole andare confrontare i tempi di esecuzione di una sezione di codice che applica un filtro su di un'immagine utilizzando le convoluzioni (normali e separabili) che viene eseguita in modalità sequenziale, parallelizzata con OpenMP e parallelizzata con CUDA. Successivamente va a misurare lo speed up che si ottiene mediante la parallelizzazione.

1. Introduzione

Questo elaborato verte sul confronto del codice necessario ad eseguire delle convoluzioni (sia separabili che non) su di un'immagine. Il codice viene eseguito in modo sequenziale, parallelo con OpenMP e parallelo con CUDA. Le convoluzioni sulle immagini vengono eseguite in modo tale che venga applicato un filtro su ogni pixel dell'immagine di riferimento.
L'immagine viene suddivisa in tiles e su ognuna di esse viene applicata la maschera. Nelle convoluzioni separabili è necessario essere sicuri che la convoluzione fatta in una direzione abbia terminato prima di eseguirla nell'altro verso.

2. Progettazione

Il metodo per applicare il filtro esamina in sequenza tutti i pixel dell'immagine. Per ciascuno di essi si moltiplica il valore di quest'ultimo e i valori dei pixel confinanti per i valori corrispondenti nel kernel. I risultati vengono poi sommati e il pixel iniziale viene impostato a questo risultato finale. Il filtro che viene applicato nel corso di questi esperimenti è un box blur. I pixel di bordo sono stati gestiti considerando quelli fuori dall'immagine come pixel neri.

3. Parallelizzazione

3.1. Con OpenMP

Nella versione parallelizzata con OpenMP per effettuare la parallelizzazione si è deciso di utilizzare

```
#pragma omp parallel
#pragma omp for
```

per applicare il filtro su ogni chunk sia per la versione separabile che non. La scomposizione in chunk può essere apprezzata nel seguente codice:

```
    #pragma omp parallel default
        (none) shared (src, dst,
        kernel, tileHeight,
        numProcs, kernelSize,
        offset)
#pragma omp for
    for (int threadIdx = 0;
        threadIdx < numProcs;
        threadIdx++) {
          filter(src, dst, kernel,
              kernelSize, offset,
            tileHeight * threadIdx
            , tileHeight * (
            threadIdx + 1));
    }
```

3.2. Con CUDA

Nella versione parallelizzata con CUDA utilizziamo un approccio *tile based* dove tutti i thread di un blocco caricano in memoria condivisa ma solo alcuni eseguono la convoluzione e salvano il risultato nell'immagine di output. Questa strategia viene seguita sia quando viene eseguita la convoluzione con un filtro separabile che non.

Nel caso della convoluzione separabile vengono eseguiti due kernel, uno che riceve come parametro un vettore colonna e uno che riceve un vettore riga.

Le dimensioni del blocco e della griglia dipendono sia dall'immagine su cui deve essere applicato il filtro, sia dal tipo di filtro che viene applicato:

- La dimensione del blocco è definita come

```
#define BLOCK_WIDTH (
    TILE_WIDTH + MASK_WIDTH -
    1)
dim3 dimBlock(BLOCK_WIDTH,
    BLOCK_WIDTH);
```

- La dimensione della griglia è invece definita come

```
dim3 dimGrid((src.cols -
    1) / TILE_WIDTH + 1, (
    src.rows - 1) /
    TILE_WIDTH + 1, src.
    channels());
```

Al fine di migliorare i tempi di accesso alla memoria[1] dove sono contenuti i dati che vengono utilizzati dai kernel è stato scelto di utilizzare le keyword *const* e *__restrict__* così da dire al compilatore che la locazione di memoria a cui punta un certo puntatore non viene mai scritta ma solo letta. Un esempio di questo accorgimento si può vedere tra i parametri della funzione che serve per eseguire le convoluzioni "normali":

```
__global__ void convolution(
    const uchar* __restrict__
    src, int srcWidth, int
    srcHeight, int srcChannels
    ,const float* __restrict__
     convKernel, int
    kernelWidth, int
    kernelHeight, uchar* dst)
```

4. Esperimenti

Gli esperimenti che sono stati svolti su entrambe le versioni parallelizzate e sulla versione sequenziale consistono nel misurare il tempo di esecuzione al variare delle dimensioni del filtro che viene applicato (3x3, 5x5, 9x9, 15x15, 31x31) e delle dimensioni dell'immagine su cui viene applicato il suddetto filtro (512x512, 1024x1024, 2048x2048, 4096x4096, 8192x8192).

Nella versione parallelizzata con CUDA la variazione della dimensione del filtro comporta anche una variazione della dimensione del tile perché il numero massimo di thread per blocco è 1024 [2] e quindi siccome la dimensione del blocco è definita in base alla dimensione del tile e del filtro, all'aumentare della grandezza del filtro può corrispondere una diminuzione della grandezza del tile. In particolare la larghezza "base" dei tile che è stata scelta è 16 pixel e quindi quando è stato utilizzato il filtro 31x31, è stato necessario ridurre la larghezza del tile è a 2 pixel.

Con OpenMP è stato effettuato anche un esperimento che vede fissate le dimensioni dell'immagine e del filtro e analizza i tempi di esecuzione al variare del numero di thread che vengono utilizzati per parallelizzare il calcolo. I tempi sono stati misurati relativamente al codice parallelizzato utilizzando 2, 4, 8, 16 e 32 thread.

L'hardware su cui sono stati eseguiti questi esperimenti è composto da un processore con 8 core logici e una scheda video Nvidia GeForce GTX 1650 (CC 7.5). Ciascuna versione parallelizzata viene confrontata con la versione sequenziale

[1] https://developer.nvidia.com/blog/cuda-pro-tip-optimize-pointer-aliasing/

[2] https://forums.developer.nvidia.com/t/maximum-number-of-threads-on-thread-block/46392

Figure 1. Immagine originale

Figure 2. Convoluzione sequenziale

Figure 3. Convoluzione sequenziale separabile

Figure 4. Convoluzione con CUDA

Figure 5. Convoluzione con CUDA separabile

Figure 6. Convoluzione con OpenMP

Figure 7. Convoluzione con OpenMP separabile

Figure 8. Esempio di output delle diverse implementazioni.
Come si può vedere gli output sono identici e quindi le implementazioni sono equivalenti.

per mettere in risalto lo speed up che si ottiene. Il codice che confronta la versione CUDA con la versione sequenziale è stato compilato utilizzando nvcc e msvc. Il codice che invece confronta la versione OpenMP con quella sequenziale è stato compilato con gcc. Questo fatto comporta l'ottenimento di tempi di esecuzione lievemente differenti anche per quanto riguarda la versione sequenziale, quindi al fine di confrontare le due strategie di parallelizzazione viene preso in considerazione lo speed up rispetto alla versione sequenziale.

5. Risultati

5.1. OpenMP

Dalle tabelle 2, 4, 6, 8 e 10 (in appendice) si nota che lo speed up massimo relativo alle convoluzioni non separabili che è stato possibile ottenere con OpenMP è 5.493 e che in generale non ci sono grosse differenze dovute all'aumento di dimensioni dell'immagine. Si vede inoltre che all'aumentare del numero dei thread lo speed up aumenta fino ad arrivare a quando il numero dei thread utilizzati è uguale al numero dei thread logici della CPU, a questo punto il tempo di esecuzione smette di decrescere e quindi anche lo speed up resta stabile.

Sempre dalle stesse tabelle ma guardando i dati relativi alle convoluzioni separabili si può vedere che il ragionamento fatto per il caso precedente vale ancora; tuttavia lo speed up massimo che è stato registrato è 2.990.

5.2. CUDA

Dalle tabelle 1, 3, 5, 7 e 9 si può invece vedere che lo speed up ottenuto con CUDA è nettamente più alto di quello ottenuto utilizzando OpenMP. Parallelizzando utilizzando la scheda video è possibile arrivare a impiegare fino a 162.74 volte meno rispetto a eseguire le stesse operazioni sulla CPU in modo sequenziale (tabella 7).

Ponendo attenzione sulla tabella 9 si può notare che in questo caso lo speed up è molto minore rispetto a tutti gli altri casi. Questo è dovuto alla riduzione della grandezza del tile (in tutti gli altri

esperimenti la larghezza del tile era 16, in questo è stata impostata a 2).

6. Conclusioni

Dai risultati degli esperimenti è possibile concludere che la parallelizzazione dell'operazione di applicazione di un filtro mediante delle convoluzioni comporta dei vantaggi sensibili dal punto di vista del tempo di esecuzione. Questi vantaggi sono maggiori se l'algoritmo viene parallelizzato utilizzando CUDA.

Nonostante lo speed up che si ottiene con le convoluzioni separabili sia minore di quello che si ottiene nell'altro caso, siccome l'algoritmo è in generale meno dispendioso conviene utilizzare filtri separabili.

7. Appendice

Dimensioni immagine (in pixel)	Sequenziale	Parallelo	Speed up	Sequenziale separabile	Parallelo separabile	Speed up (separabile)
512x512	0.025167	0.000443	56.77193774	0.012808	0.00066	19.4091529
1024x1024	0.096339	0.001486	64.83135935	0.054289	0.002197	24.71483201
2048x2028	0.42395	0.00559	75.8462144	0.225709	0.007158	31.53064931
4096x4096	1.75098	0.022824	76.71560574	0.842219	0.02763	30.48215882
8192x8192	6.95061	0.130942	53.08159338	3.8581	0.108906	35.42596368

Table 1. Tempi di esecuzione al variare della dimensione dell'immagine. Dimensioni del filtro applicato: 3x3. Parallelizzato utilizzando CUDA.

Dimensioni immagine (in pixel)	Sequenziale	Parallelo (con 2 core)	Parallelo (con 4 core)	Parallelo (con 8 core)	Parallelo (con 16 core)	Parallelo (con 32 core)	Speed up
512x512	0.024554	0.012162	0.00943	0.009517	0.008919	0.008121	3.023631
1024x1024	0.063774	0.035766	0.031287	0.022352	0.020893	0.020317	3.138958
2048x2028	0.258799	0.153329	0.118376	0.077398	0.086738	0.070213	3.685908
4096x4096	1.14128	0.872203	0.377732	0.261865	0.262361	0.281657	4.358276
8192x8192	4.28233	2.53015	1.69968	1.06588	1.04616	1.0474	4.09338
	Sequenziale separabile	Parallelo separabile (con 2 core)	Parallelo separabile (con 4 core)	Parallelo separabile (con 8 core)	Parallelo separabile (con 16 core)	Parallelo separabile (con 32 core)	Speed up (separabile)
512x512	0.010371	0.013587	0.009448	0.00988	0.010791	0.0073513	1.410798
1024x1024	0.03561	0.038553	0.032698	0.025284	0.030678	0.0307885	1.408399
2048x2028	0.185038	0.120044	0.1176	0.102303	0.102696	0.116131	1.808725
4096x4096	0.624931	0.516772	0.438976	0.406002	0.433012	0.423674	1.539231
8192x8192	2.55475	2.09156	1.8677	1.65729	1.67239	1.62134	1.575703

Table 2. Tempi di esecuzione al variare della dimensione dell'immagine. Dimensioni del filtro applicato: 3x3. Lo speed up è calcolato come tempo sequenziale corrispondente ad una certa immagine / minimo dei tempi ottenuti mediante parallelizzazione, quindi il numero presente in tabella rappresenta l'accelerazione massima che è stata ottenuta. Parallelizzato utilizzando OpenMP.

Dimensioni immagine (in pixel)	Sequenziale	Parallelo	Speed up	Sequenziale separabile	Parallelo separabile	Speed up (separabile)
512x512	0.052131	0.000604	86.29565	0.021837	0.000763	28.60506
1024x1024	0.208695	0.001898	109.9494	0.082454	0.002138	38.56211
2048x2028	0.825305	0.007358	112.1719	0.342456	0.007503	45.64254
4096x4096	3.8867	0.028793	134.99	1.22595	0.028869	42.46538
8192x8192	15.0385	0.198526	75.75078	5.31706	0.116713	45.55671

Table 3. Tempi di esecuzione al variare della dimensione dell'immagine. Dimensioni del filtro applicato: 5x5. Parallelizzato utilizzando CUDA.

Dimensioni immagine (in pixel)	Sequenziale	Parallelo (con 2 core)	Parallelo (con 4 core)	Parallelo (con 8 core)	Parallelo (con 16 core)	Parallelo (con 32 core)	Speed up
512x512	0.077703	0.046092	0.039929	0.023196	0.016459	0.017227	4.720958
1024x1024	0.215367	0.17825	0.113897	0.075726	0.066313	0.052693	4.087242
2048x2028	0.991745	0.398764	0.357353	0.262344	0.234855	0.302302	4.222797
4096x4096	3.3993	2.18228	1.0928	0.812338	0.717458	0.711125	4.780172
8192x8192	11.6312	6.59423	4.55312	2.77202	2.8027	2.77641	4.195929
	Sequenziale separabile	Parallelo separabile (con 2 core)	Parallelo separabile (con 4 core)	Parallelo separabile (con 8 core)	Parallelo separabile (con 16 core)	Parallelo separabile (con 32 core)	Speed up (separabile)
512x512	0.02716	0.017961	0.018639	0.015127	0.009085	0.011529	2.989587
1024x1024	0.057603	0.086481	0.076177	0.047029	0.055089	0.048041	1.224834
2048x2028	0.224713	0.168857	0.224692	0.178984	0.212035	0.194221	1.330789
4096x4096	0.855417	0.795006	0.652497	0.682463	0.698951	0.681255	1.31099
8192x8192	3.38813	2.85981	2.4687	2.28358	2.27848	2.22955	1.519647

Table 4. Tempi di esecuzione al variare della dimensione dell'immagine. Dimensioni del filtro applicato: 5x5. Lo speed up è calcolato come tempo sequenziale corrispondente ad una certa immagine / minimo dei tempi ottenuti mediante parallelizzazione, quindi il numero presente in tabella rappresenta l'accelerazione massima che è stata ottenuta. Parallelizzato utilizzando OpenMP.

Dimensioni immagine (in pixel)	Sequenziale	Parallelo	Speed up	Sequenziale separabile	Parallelo separabile	Speed up (separabile)
512x512	0.138503	0.001211	114.3802	0.025986	0.000924	28.12588
1024x1024	0.553139	0.004477	123.5402	0.102715	0.003139	32.71804
2048x2028	2.24156	0.017806	125.8879	0.412621	0.01168	35.32593
4096x4096	9.84972	0.069944	140.8227	2.32176	0.046119	50.34314
8192x8192	38.3746	0.266941	143.7569	7.53318	0.161333	46.69336

Table 5. Tempi di esecuzione al variare della dimensione dell'immagine. Dimensioni del filtro applicato: 9x9. Parallelizzato utilizzando CUDA.

Dimensioni immagine (in pixel)	Sequenziale	Parallelo (con 2 core)	Parallelo (con 4 core)	Parallelo (con 8 core)	Parallelo (con 16 core)	Parallelo (con 32 core)	Speed up
512x512	0.149582	0.096061	0.063565	0.041363	0.046355	0.039287	3.807417
1024x1024	0.573956	0.407016	0.196007	0.134121	0.131435	0.15275	4.366843
2048x2028	2.11917	1.33022	0.778621	0.557927	0.638481	0.528573	4.009229
4096x4096	9.31113	5.16807	2.98314	2.19273	2.14576	2.13313	4.365008
8192x8192	38.9233	20.7499	13.1166	8.41557	8.41296	8.54193	4.626588
	Sequenziale separabile	Parallelo separabile (con 2 core)	Parallelo separabile (con 4 core)	Parallelo separabile (con 8 core)	Parallelo separabile (con 16 core)	Parallelo separabile (con 32 core)	Speed up (separabile)
512x512	0.021072	0.032217	0.023124	0.020352	0.021049	0.019642	1.072756
1024x1024	0.075107	0.0701	0.059737	0.056074	0.090946	0.051017	1.472192
2048x2028	0.316916	0.295014	0.24614	0.235778	0.252615	0.222572	1.423881
4096x4096	1.42621	1.12629	0.973969	1.00642	1.08688	1.02447	1.464328
8192x8192	6.02858	4.5576	3.99046	3.49393	3.505	3.50085	1.725444

Table 6. Tempi di esecuzione al variare della dimensione dell'immagine. Dimensioni del filtro applicato: 9x9. Lo speed up è calcolato come tempo sequenziale corrispondente ad una certa immagine / minimo dei tempi ottenuti mediante parallelizzazione, quindi il numero presente in tabella rappresenta l'accelerazione massima che è stata ottenuta. Parallelizzato utilizzando OpenMP.

Dimensioni immagine (in pixel)	Sequenziale	Parallelo	Speed up	Sequenziale separabile	Parallelo separabile	Speed up (separabile)
512x512	0.339669	0.002712	125.2559	0.041986	0.001093	38.39985
1024x1024	1.38293	0.01047	132.085	0.171274	0.003948	43.37918
2048x2028	5.44511	0.041135	132.3717	0.688943	0.014724	46.7908
4096x4096	23.0781	0.329958	69.94254	2.90478	0.047319	61.3877
8192x8192	92.7028	0.56964	162.7393	13.1883	0.188492	69.96743

Table 7. Tempi di esecuzione al variare della dimensione dell'immagine. Dimensioni del filtro applicato: 15x15. Parallelizzato utilizzando CUDA.

Dimensioni immagine (in pixel)	Sequenziale	Parallelo (con 2 core)	Parallelo (con 4 core)	Parallelo (con 8 core)	Parallelo (con 16 core)	Parallelo (con 32 core)	Speed up
512x512	0.489327	0.212081	0.109087	0.089073	0.090514	0.102022	5.49355
1024x1024	1.60606	0.77134	0.514782	0.347877	0.337485	0.346278	4.758908
2048x2028	6.88598	3.32387	2.14654	1.35976	1.56195	1.52382	5.064114
4096x4096	27.374	12.9038	8.09761	5.49599	5.60898	5.58547	4.980722
8192x8192	108.483	52.1395	33.6056	22.3668	22.4296	22.3928	4.85018
	Sequenziale separabile	Parallelo separabile (con 2 core)	Parallelo separabile (con 4 core)	Parallelo separabile (con 8 core)	Parallelo separabile (con 16 core)	Parallelo separabile (con 32 core)	Speed up (separabile)
512x512	0.041271	0.025473	0.021012	0.029447	0.02824	0.032923	1.964182
1024x1024	0.111036	0.101519	0.086217	0.086176	0.099068	0.078371	1.416807
2048x2028	0.508626	0.432386	0.358672	0.409506	0.392169	0.386957	1.418081
4096x4096	2.24651	1.68336	1.51734	1.51052	1.37706	1.36204	1.649372
8192x8192	8.65534	7.1119	5.84008	5.51389	5.63912	5.82052	1.569734

Table 8. Tempi di esecuzione al variare della dimensione dell'immagine. Dimensioni del filtro applicato: 15x15. Lo speed up è calcolato come tempo sequenziale corrispondente ad una certa immagine / minimo dei tempi ottenuti mediante parallelizzazione, quindi il numero presente in tabella rappresenta l'accelerazione massima che è stata ottenuta. Parallelizzato utilizzando OpenMP.

Dimensioni immagine (in pixel)	Sequenziale	Parallelo	Speed up	Sequenziale separabile	Parallelo separabile	Speed up (separabile)
512x512	1.29859	0.203361	6.385639	0.068066	0.02711	2.510736
1024x1024	5.20358	0.681825	7.631841	0.294446	0.103219	2.852634
2048x2028	21.0456	2.98823	7.042831	1.16747	0.41218	2.832428
4096x4096	88.2998	11.189	7.891661	5.11679	1.64945	3.102119
8192x8192	377.616	43.4623	8.688357	22.6498	6.60364	3.429896

Table 9. Tempi di esecuzione al variare della dimensione dell'immagine. Dimensioni del filtro applicato: 31x31. Parallelizzato utilizzando CUDA.

Dimensioni immagine (in pixel)	Sequenziale	Parallelo (con 2 core)	Parallelo (con 4 core)	Parallelo (con 8 core)	Parallelo (con 16 core)	Parallelo (con 32 core)	Speed up
512x512	1.51648	0.916039	0.548664	0.372969	0.365656	0.372714	4.147286
1024x1024	6.50583	3.58167	2.20842	1.48031	1.58477	1.62815	4.394911
2048x2028	25.3499	14.09	8.97724	5.94973	6.06774	6.02742	4.260681
4096x4096	103.349	58.2731	35.0424	24.0742	24.0902	24.113	4.292936
8192x8192	422.102	234.597	140.372	96.932	98.2376	99.3121	4.35462
	Sequenziale separabile	Parallelo separabile (con 2 core)	Parallelo separabile (con 4 core)	Parallelo separabile (con 8 core)	Parallelo separabile (con 16 core)	Parallelo separabile (con 32 core)	Speed up (separabile)
512x512	0.073939	0.055026	0.054307	0.053754	0.058424	0.054522	1.375523
1024x1024	0.291094	0.223583	0.231131	0.228416	0.223228	0.235806	1.304021
2048x2028	1.1653	0.91966	0.831892	0.864753	0.868192	0.777478	1.498821
4096x4096	4.93529	3.91212	3.2408	3.10853	3.08712	3.07073	1.607204
8192x8192	20.3757	16.0752	13.471	12.3525	12.7555	13.0807	1.64952

Table 10. Tempi di esecuzione al variare della dimensione dell'immagine. Dimensioni del filtro applicato: 31x31. Lo speed up è calcolato come tempo sequenziale corrispondente ad una certa immagine / minimo dei tempi ottenuti mediante parallelizzazione, quindi il numero presente in tabella rappresenta l'accelerazione massima che è stata ottenuta. Parallelizzato utilizzando OpenMP.