Kernel Image Processing

Editing di immagini tramite maschere di convoluzione in Multi-Threading C++11

Andrea Spitaleri

A.A. 2019/2020

Applicazione di filtri ad immagini

Analisi del processo di editing

Lo scopo di questo elaborato è mostrare le differenze nei tempi di calcolo, di un software che applica maschere di convoluzione, utilizzando codice Single-Thread e Multi-Thread (tramite std::thread, il Multi-Threading nativo di C++11), analizzando i casi specifici di blurring (sfuocamento) di immagini tramite Box Filter e Gaussian Filter.
Prenderemo in considerazione le differenze nelle dinamiche del processo di editing, e nei tempi richiesti dai vari Thread per completare lo stesso task che porta ad ottenere l'immagine finale.

https://github.com/spita90/Kernel-Image-Processing

L’editing in dettaglio

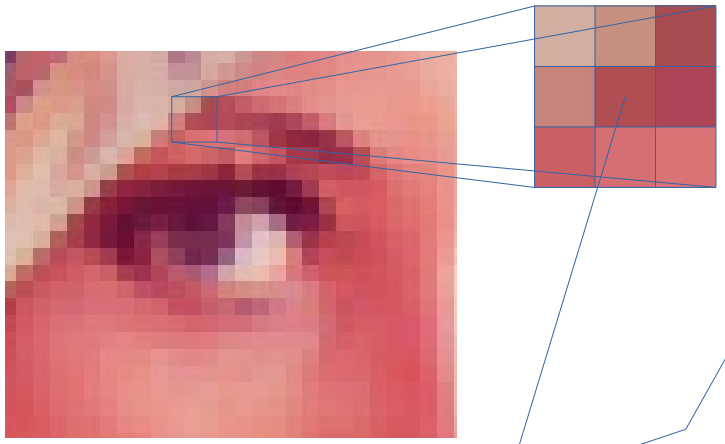
Descrizione degli obiettivi

In questo elaborato vogliamo applicare filtri di sfuocamento a un’immagine, esattamente come accade nei più comuni software di elaborazione grafica. Per questo genere di effetti grafici, in quasi tutti i software si usa una maschera di convoluzione applicata di pixel in pixel, la quale essenzialmente associa a ogni pixel la media (pesata o meno) del colore dei pixel circostanti.

In un approccio Single-Thread il programma elabora tutti i pixel in modo sequenziale, mentre nell’approccio Multi-Thread adottato nel mio elaborato, a ogni unità di calcolo viene associata una porzione dell’immagine iniziale, quindi il calcolo può avvenire in modo parallelo, al fine di avere tempi di computazione complessivamente minori, più compatibili con i tempi che ci si aspetta da un software di questo tipo.

L'editing in dettaglio

Descrizione degli obiettivi



Gaussian Filter

1	2	1
2	4	2
1	2	1

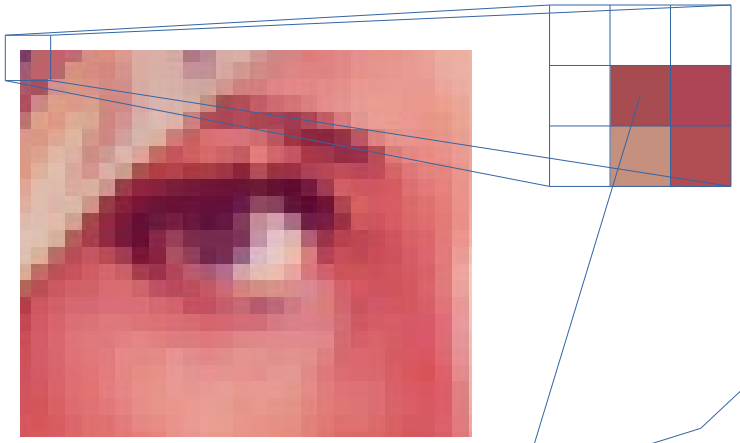
Somma

image[5,14] = {1*image[4,13] + 2*image[4,14] + … + 1*image[6,15]} / 16

(Media del pixel con i pixel circostanti)

L'editing in dettaglio

Descrizione degli obiettivi



Nel caso di calcolo di pixel di confine (angoli o bordi), basta considerare i pixel fuori immagine come contributo nullo, e non considerare i relativi moltiplicatori della maschera nella somma (divisore della media):

Gaussian Filter

1	2	1
2	4	2
1	2	1

Somma

image[0,0] = {1*NULL + 2*NULL + … + 1*image[1,1]} / 9

(Media dei soli pixel circostanti validi)

Il problema nel caso Multi-Thread

Suddivisione dell'immagine

Nel caso Single-Thread, tutti i pixel dell'immagine sono calcolati da una singola unità di calcolo della CPU.
Se vogliamo utilizzare tutte le unità di calcolo disponibili occorre suddividere il carico di lavoro, così da poter sfruttare il calcolo parallelo.
Questa suddivisione introduce delle sottigliezze di cui è importante tener conto.
Prima tra tutte è: come suddividere l'immagine in modo appropriato?

Il problema nel caso Multi-Thread

Suddivisione dell'immagine in blocchi di dimensione fissa

Un approccio possibile può essere quello di suddividere l'immagine in blocchi di grandezza fissata:

Ciò introduce però alcuni possibili punti deboli:

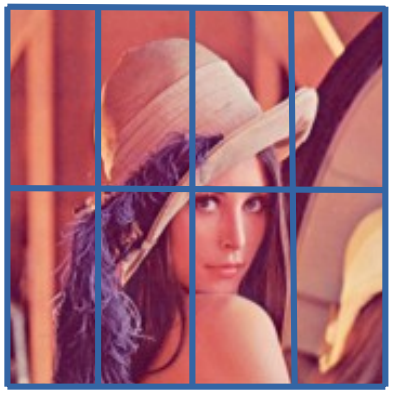
- L'immagine potrebbe non ricadere esattamente in tutti i blocchi: alcuni (quelli a destra e in basso) sprecherebbero un po' di tempo a fare lavoro inutile.
- A seconda delle dimensioni dell'immagine e dei blocchi potremmo avere molto overhead a causa della schedulazione dei thread sui vari core (come nel caso in figura).

Vorremmo avere un approccio di suddivisione che non presenti questi problemi.

Il problema nel caso Multi-Thread

Suddivisione dell'immagine in blocchi di dimensione dinamica

Ha più senso un approccio che attribuisce dimensione dinamica ai blocchi:



Vediamo il caso in cui la CPU ha 8 core logici:
La figura rappresenta una suddivisione all'apparenza ragionevole.

Cosa succede però se l'immagine non è quadrata come quella in figura, ma è in modalità landscape o portrait?
Quante suddivisioni orizzontali e verticali conviene fare?

Il problema nel caso Multi-Thread

Suddivisione dell'immagine in blocchi di dimensione dinamica

Immagine 500x300

Nel caso con 8 core con immagine landscape, notiamo che con suddivisione solo verticale ogni thread ha bisogno, oltre che del blocco su cui lavora, di un bordo di pixel (per il calcolo del colore ai confini) fatto da due segmenti da 500/8 pixel e due segmenti da 300 pixel. Quindi la parte in più, di bordo, è per ogni thread, di circa 725 pixel.

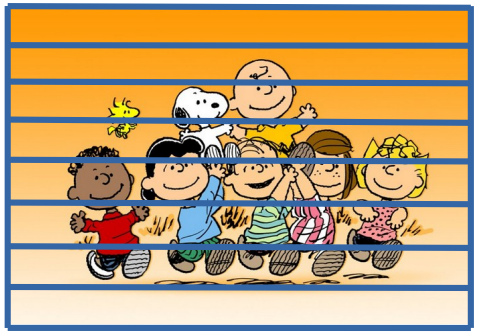
Ovviamente per il primo e l'ultimo thread 300+2*(500/8) pixel sono nulli, in quanto fuori dall'immagine, e solo 300 sono di pixel "veri". Per i thread intermedi invece solo 2*(500/8) pixel sono nulli.

Se non contiamo i pixel nulli, si hanno complessivamente per gli 8 thread:
6(2*300)+2(300) = 4200 calcoli di pixel in più.

Il problema nel caso Multi-Thread

Suddivisione dell'immagine in blocchi di dimensione dinamica

Immagine 500x300



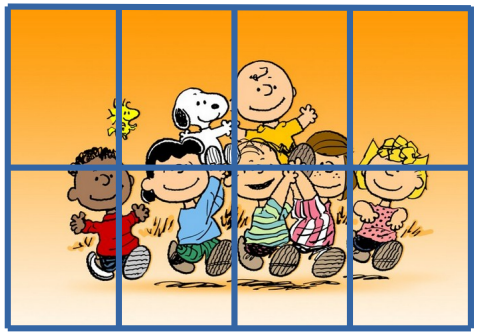
Con suddivisione solo orizzontale ogni thread ha bisogno, oltre che del blocco su cui lavora, di un bordo di pixel (per il calcolo del colore ai confini) fatto da due segmenti da 500 pixel e due segmenti da 300/8 pixel. Quindi la parte in più, di bordo, è per ogni thread, di circa 1075 pixel.

Ovviamente per il primo e l'ultimo thread 500+2*(300/8) pixel sono nulli, in quanto fuori dall'immagine, e solo 500 sono di pixel "veri". Per i thread intermedi invece solo 2*(300/8) pixel sono nulli.
Se non contiamo i pixel nulli, si hanno complessivamente per gli 8 thread: 6(2*500)+2(500) = 7000 calcoli di pixel in più.
Questo è molto peggiore del caso precedente per immagini landscape, ma sarà inverso nel caso di immagini portrait!
Cerchiamo quindi un modo di suddividere l'immagine che sia universalmente buono.

Il problema nel caso Multi-Thread

Suddivisione dell'immagine in blocchi di dimensione dinamica

Immagine 500x300



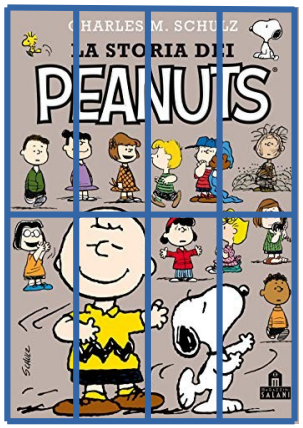
Suddivisione mista: ogni thread ha bisogno, oltre che del blocco su cui lavora, di un bordo di pixel (per il calcolo del colore ai confini) fatto da due segmenti da 500/4 pixel e due segmenti da 300/2 pixel. Quindi la parte in più, di bordo, è per ogni thread, di circa 550 pixel.

Ovviamente per i thread agli angoli metà pixel sono nulli, in quanto fuori dall'immagine, e solo l'altra metà sono pixel "veri". Per i thread "interni" invece solo 500/4 pixel sono nulli.
Se non contiamo i pixel nulli, si hanno complessivamente per gli 8 thread:
4(500/4+300/2)+4(500/4+2*(300/2)) = 2800 calcoli di pixel in più.
Adesso sì che si inizia a migliorare!

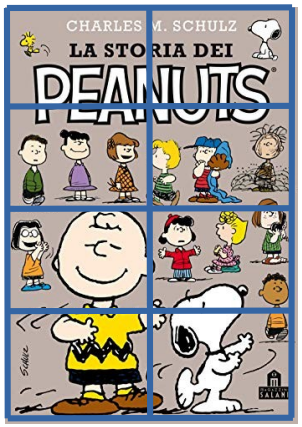
Il problema nel caso Multi-Thread

Suddivisione dell'immagine in blocchi di dimensione dinamica

Immagine 300x500 conviene dividere maggiormente sul lato corto o su quello lungo?



Se non contiamo i pixel nulli, si hanno complessivamente per gli 8 thread:
4(500/2+300/4)+
4(300/4+2*(500/2))
= 3600 calcoli di pixel in più.



Se non contiamo i pixel nulli, si hanno complessivamente per gli 8 thread:
4(500/4+300/2)+
4(500/4+2*(300/2))
= 2800 calcoli di pixel in più.
È il caso della slide precedente.

Risulta dunque chiaro che è sempre meglio fare più suddivisioni sul lato lungo che sul lato corto. A ragione di ciò la mia implementazione del software riconosce preventivamente se l'immagine è landscape o portrait.

Il problema nel caso Multi-Thread

Suddivisione al variare dei thread

Adesso che abbiamo visto che è più conveniente suddividere maggiormente il lato più lungo, rispetto al lato più corto, vediamo come questo risultato varia al variare del numero di thread.

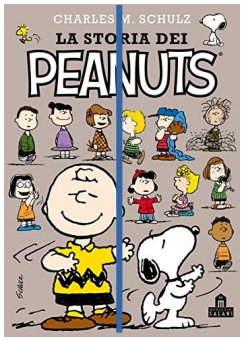
Il problema nel caso Multi-Thread

Suddivisione al variare dei thread

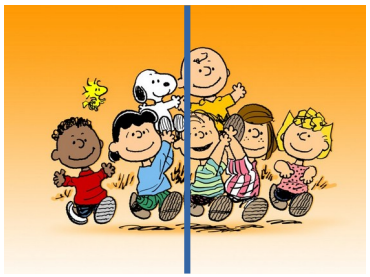
2 Thread

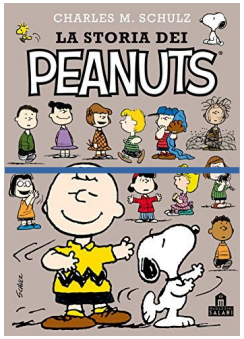
500x200

200x500



Divisione sul lato corto.
In entrambi i casi per ogni blocco ho 2*500=1000 calcoli di pixel in più.
Come avevamo già visto non conviene dividere per il lato corto.





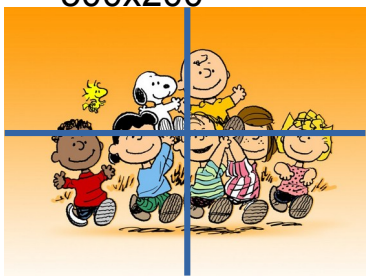
Divisione sul lato lungo.
In entrambi i casi per ogni blocco ho 2*200=400 calcoli di pixel in più.

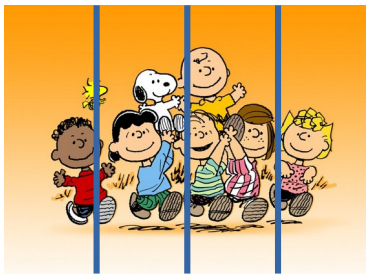
Il problema nel caso Multi-Thread

Suddivisione al variare dei thread

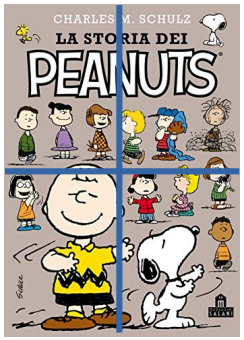
4 Thread

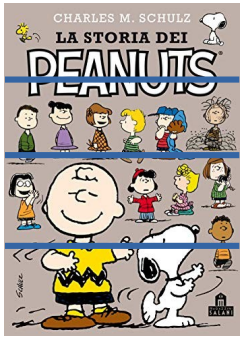
500x200





200x500





Dunque smettiamo di dividere sul lato corto.
Divisione su entrambi i lati.
Ho 4*(250+100) = 1400 calcoli in più

Divisione sul lato lungo.
Ho 2*200+
2(2*200)=
1200 calcoli di pixel in più

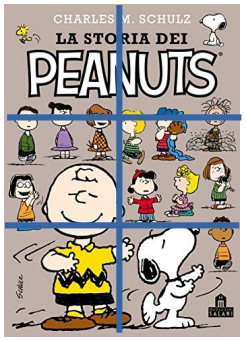
Il problema nel caso Multi-Thread

Suddivisione al variare dei thread

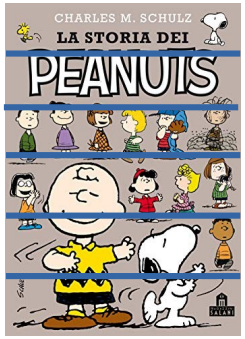
6 Thread

500x200

200x500



Divisione su entrambi i lati.
Ho 4*(100+167) +
2*(2*100+167) = 1802 calcoli in più



Divisione sul lato lungo.
Ho 2*200+
4*(2*200)=
2000 calcoli di pixel in più

Dai 6 thread in su sembrerebbe che generalmente non convenga più dividere solo per il lato lungo.

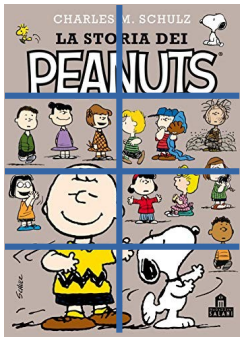
Il problema nel caso Multi-Thread

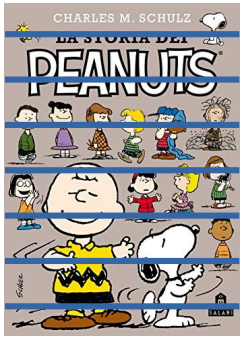
Suddivisione al variare dei thread

8 Thread

500x200

200x500





Divisione su entrambi i lati.
Ho 4*(100+125) +
4*(2*100+125) = 2200 calcoli in più

Divisione sul lato lungo.
Ho 2*200+
6*(2*200)=
2800 calcoli di pixel in più

Ciò conferma il fatto che dai 6 thread in più, generalmente conviene suddividere anche sul lato corto. Ok, ma quante divisioni conviene fare sul lato corto? Si può notare che risulta peggiorativo suddividere il lato corto in più di due parti, poiché ciò creerebbe blocchi centrali con nessun lato costituito da pixel nulli!

Analisi prestazionale dell'editing

Confronto tra approccio Single-Thread e Multi-Thread

Sfruttando i timer temporali integrati nella GUI del programma possiamo analizzare lo speedup al variare di alcuni parametri, tra cui numero di passaggi di blurring, risoluzione dell'immagine, e numero di thread.

I seguenti benchmark sono eseguiti su:
Laptop Xiaomi Mi Notebook Pro
CPU Intel Core i7-8550U (8 core logici a 1.8 GHz - fino a 4 GHz in Turbo Boost)
RAM 16 GB DDR4
S.O. Manjaro Linux 18.1.4 (basato su Arch Linux) con Kernel Linux 5.4

Verranno effettuate 5 prove per ogni casistica, e i tempi riportati nelle tabelle a seguire sono i risultati medi.

Analisi prestazionale dell'editing

Immagine con risoluzione 100x100

BOX	*Risoluzione*					
	100x100					
	Passaggi					
	10		50		100	
Thread	*Tempi (ms)*					
1 (Single)	34		137		238	
	Tempi (ms)	*Speedup*	*Tempi (ms)*	*Speedup*	*Tempi (ms)*	*Speedup*
2	13	2,62	54	2,54	86	2,77
8	14	2,43	60	2,28	103	2,31
16	15	2,27	63	2,17	104	2,29
64	25	1,36	75	1,83	136	1,75
100	37	0,92	92	1,49	228	1,04

Possiamo notare che per piccolissime immagini si hanno migliori risultati, per quanto piccoli, con 2 thread. Leggermente peggio con 8 thread, e via via peggiorando all'aumentare del numero di thread.
Questo è comprensibile e totalmente ragionevole, in quanto l'overhead che si ha a causa dei numerosi thread diventa sempre maggiore, fino al punto di rendere totalmente inappropriato un approccio Multi-Threaded.
Si nota inoltre che nel caso di piccolissime immagini l'aumento del numero di passaggi non incide sullo speedup.

Analisi prestazionale dell'editing

Immagine con risoluzione 640x480

BOX	*Risoluzione*					
	640x480					
	Passaggi					
	10		50		100	
Thread	*Tempi (ms)*					
1 (Single)	751		3663		7310	
	Tempi (ms)	*Speedup*	*Tempi (ms)*	*Speedup*	*Tempi (ms)*	*Speedup*
2	385	1,95	1891	1,94	3688	1,98
8	212	3,54	1100	3,33	1970	3,71
16	236	3,18	1080	3,39	2014	3,63
64	228	3,29	1073	3,41	2047	3,57
100	224	3,35	1111	3,30	2081	3,51

Per immagini un po' più grandi la situazione cambia molto, infatti non solo si hanno valori di speedup maggiori, ma questi si ottengono per un numero di thread più grande. Infatti, per queste risoluzioni, per pochissimi thread si hanno valori di speedup abbastanza bassi.

Si nota inoltre che nel caso di immagini con questa risoluzione, l'aumento del numero di passaggi sembra incidere molto poco sullo speedup.

Analisi prestazionale dell'editing

Immagine con risoluzione 1920x1080

BOX	*Risoluzione*					
	1920x1080					
	Passaggi					
	10		50		100	
Thread	*Tempi (ms)*					
1 (Single)	5097		26321		51381	
	Tempi (ms)	*Speedup*	*Tempi (ms)*	*Speedup*	*Tempi (ms)*	*Speedup*
2	2606	1,96	15005	1,75	30079	1,71
8	1326	3,84	9229	2,85	19883	2,58
16	1341	3,80	8128	3,24	20185	2,55
64	1324	3,85	9510	2,77	19077	2,69
100	1328	3,84	10255	2,57	20100	2,56

Per immagini ad alta risoluzione si ha un andamento dello speedup in funzione del numero di thread simile al caso di immagini di media risoluzione.
Sembra quindi che all'aumentare della risoluzione dell'immagine, il numero di thread da usare per avere speedup maggiori tende al numero di core della CPU.

Si nota inoltre che nel caso di immagini ad alta risoluzione, l'aumento del numero di passaggi sembra incidere notevolmente sullo speedup.

Analisi prestazionale dell'editing

Considerazioni finali

Abbiamo potuto notare che per l'editing di immagini sempre più grandi si rende sempre più necessario un processore con un numero di core sempre maggiore, in quanto si tendono ad avere speedup maggiori per un numero di thread sempre più tendenti al numero di core disponibili.

Ciò è perfettamente in linea con la realtà, in quanto chi fa image editing per professione sa bene che deve dotarsi di pc con molti core.

Andrea Spitaleri