RealTime ASCII Video Converter

Lorenzo Grillo

Primo paragrafo: Descrizione idea

Secondo paragrafo: Implementazione

Terzo paragrafo: Osservazioni e valori prestazionali

Sommario

L'obiettivo principale di questo progetto è sfruttare la potenza dell'elaborazione multicore per effettuare la conversione in tempo reale di un video in caratteri ASCII, utilizzando le librerie OpenCV e SDL. L'applicazione acquisisce il video e lo suddivide in frame, successivamente elabora ogni singolo frame trasformandolo in un'immagine composta da caratteri ASCII con l’aggiunta dei colori presi direttamente dai pixel del frame.

L'implementazione di un'elaborazione multi core consente al programma di sfruttare appieno le potenzialità di elaborazione parallela dei moderni processori, garantendo prestazioni ottimali durante la conversione del video. L'utilizzo di librerie come OpenCV consente un'elaborazione efficiente del video, facilitando l'acquisizione, l'analisi e la manipolazione dei singoli frame. SDL, invece, permette di creare un'interfaccia grafica per visualizzare il video convertito in tempo reale.

Descrizione idea

Acquisizione del video: All’avvio, l’applicazione legge i dati dal file config per inizializzare le variabili che serviranno per la codifica del video, per attivare la modalità profiler (utile per testare le prestazioni) e per settare il path del video. Una volta conclusa l’inizializzazione si passa all’apertura del video attraverso la libreria OpenCV per acquisire i frame del video. OpenCV è una libreria open source molto diffusa per l'elaborazione delle immagini e la visione artificiale. Offre un'ampia varietà di funzioni e strumenti per acquisire, elaborare, analizzare e visualizzare immagini e video.

Elaborazione dei frame: Per ogni frame del video acquisito, ogni processo calcola la sua porzione da elaborare e spedisce quella di scarto al processo successivo, creando cosi una catena gestita dalle funzioni di Isend (non bloccante) e Receive (bloccante).

Conversione in ASCII art: Terminata la partizione dei pixel, i pixel assegnati ad ogni processo vengono convertiti in testo ASCII. Questo viene fatto mappando i diversi valori di intensità dei pixel dell'immagine ai caratteri ASCII corrispondenti. Ad esempio, i pixel più scuri potrebbero essere associati a caratteri ASCII più "pesanti" o da uno spazio vuoto, mentre i pixel più chiari potrebbero corrispondere a caratteri più leggeri. Per caratteri pesanti si intendono tutti quelli che nella console appaiono con una maggioranza di colore nero. In questa fase viene

inoltre decodificato il colore dei pixel, utile successivamente per fornire un output colorato del frame ricostruito.

Ricostruzione del frame: Terminata la conversione dei vari pixel, il frame viene riassemblato partendo dall'ultimo che invierà tramite un Isend la propria porzione convertita. Gli altri processi seguiranno successivamente in ordine fino a terminare con quello principale che avrà anche il compito di mostrare a schermo il frame finale. I colori precedentemente decodificati dai vari processi invece vengono accoppiati al pixel convertito in un buffer secondario attraverso l'utilizzo di MPI_gather e di un data type custom creato ad hoc per supportare l'invio e ricezione del tipo struct SDL_Color.

Utilizzo di openMPI:

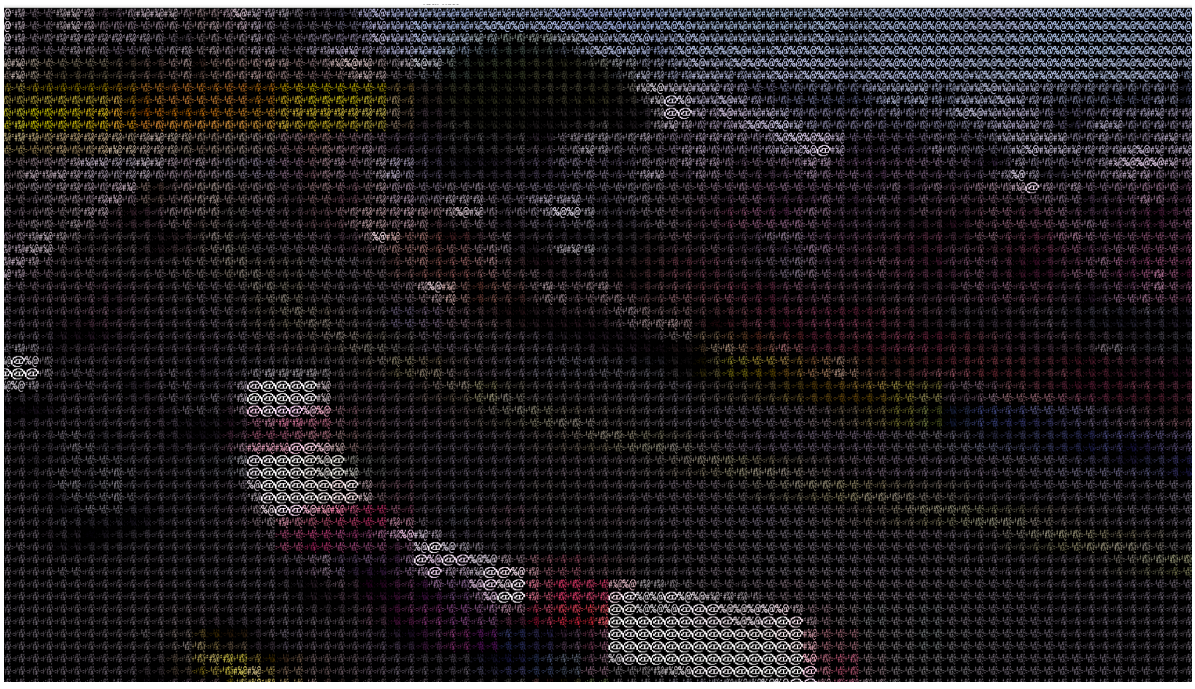
Il programma è stato opportunamente parallelizzato sfruttando la libreria MPI (Message Passing Interface), la quale rappresenta uno standard per la programmazione parallela. Tale approccio consente ai processi di interagire scambiandosi messaggi tra loro. Il fine del programma consiste nell'effettuare la conversione dei pixel in formato ASCII, ottenendo in parallelo anche le informazioni relative ai colori associati. In questa implementazione, si è adottata una topologia cartesiana a una dimensione, poiché, come successivamente si evidenzierà, quella a due dimensioni potrebbe generare un eccessivo overhead.

Mediante la topologia cartesiana, il lavoro è equamente distribuito tra i processi, consentendo un'elaborazione efficiente dei dati. Il processo principale, con un rank eventualmente differente

da 0, assume la responsabilità di acquisire i frame del video e di suddividerli in porzioni appropriate da inviare a ciascun processo. In tal modo, i processi ricevono solo i pixel necessari, evitando di dover gestire l'intera matrice dei pixel.

Inoltre, ciascun processo ha il compito di trasmettere la sua porzione di pixel al processo adiacente. Dopo aver elaborato i pixel in formato ASCII, il processo principale si occupa di "unire" i risultati ottenuti, al fine di poterli visualizzare in modo coerente attraverso l'utilizzo di SDL2, la libreria grafica utilizzata per la gestione dell'interfaccia grafica. I pixel vengono convertiti seguendo questa formula: r+g+b/3, usata per accedere ad un array di char contenenti i rispettivi valori in ASCII.

Osservazioni e valori prestazionali



Esempio esecuzione del programma: da notare come i colori più chiari sono rappresentati da una @).

Al fine di condurre un'analisi delle prestazioni più equa possibile, il programma è sottoposto a iterazioni per un determinato numero di passi, specificamente 10. Si è scelto di mantenere costante la temperatura di partenza della CPU e quella dell'ambiente circostante. Tale accortezza è stata adottata con l'intento di preservare l'integrità delle prestazioni del sistema, evitando che una temperatura elevata nell'ambiente possa condurre al thermal throttling della CPU, con conseguente limitazione delle prestazioni globali.

(Esecuzione senza gui, tempi sono in ms)

Run	-O1		-O3	
	-np 2	-np 1	-np 2	-np 1
1	101	55	37	43
2	108	67	33	36
3	97	38	32	42
4	95	34	32	53
5	91	34	31	27
6	92	35	31	26
7	104	38	32	27
8	91	38	33	26
9	127	39	32	31
10	101	38	34	34

(Esecuzione con gui, tempi in ms)

Run	-O1		-O3	
	-np 2	-np 1	-np 2	-np 1
1	894	352	366	303
2	749	362	378	281
3	721	368	379	283
4	687	372	397	282
5	686	368	414	275
6	708	417	423	291
7	667	415	539	328
8	670	450	500	331
9	661	493	483	361
10	673	504	497	369

PC usato per i test:

Processore: Intel Core i5 (in questo caso fanless, senza ventola).

Memoria RAM: 8GB DDR4-2133 MHz, non rimovibile.

GPU: Integrata (Intel).

Il video in input ha una risoluzione di 640x360 pixel e con un tempo ideale di rendering per frame di 42 millisecondi.

NB: La risoluzione del video va moltiplicata per un valore che si aggira tra 3 e 4 in quanto altrimenti i caratteri risulterebbero troppo piccoli da poterli visualizzare (quindi la risoluzione finale è 2560x1440).

Osservando la tabella si nota un significativo miglioramento delle prestazioni nella versione single core rispetto a quella multi-core. Le ragioni di tale disparità possono essere diverse, ma un'analisi delle prestazioni con l'opzione -O3 rivela che la versione priva di GUI multi-core risulta essere paragonabile o addirittura più veloce in alcune circostanze. L'overhead persistente nella versione multi-core è probabilmente attribuibile alle convenzioni del linguaggio C, che si attenuano una volta che il codice è stato compilato con l'opzione -O3 portando l'applicazione ad avere prestazioni simili o maggiori, si ipotizza un elevato utilizzo dello stack e quindi di istruzioni push/pop "inutili". Inoltre, il thermal throttling svolge un ruolo rilevante nelle prestazioni globali, poiché può causare una notevole riduzione del frame rate al fine di mantenere la CPU all'interno del suo range di temperatura di lavoro ideale. Un'altra ipotesi da considerare è che inizialmente il programma in multi-core possa scalare bene, mostrando buone prestazioni. Tuttavia, con il passare del tempo, la CPU potrebbe richiedere un maggiore utilizzo delle risorse per mantenere tali performance elevate. Ciò potrebbe portare la CPU a entrare in thermal throttling, causando una riduzione drastica delle prestazioni globali.

Il thermal throttling è una misura di sicurezza implementata per evitare il surriscaldamento della CPU e danni hardware. Quando la CPU raggiunge una temperatura critica, il sistema riduce la frequenza di clock e il voltaggio per ridurre il calore generato e prevenire danni. Questa situazione può avere un impatto negativo sulle prestazioni globali del programma, poiché la CPU opererà a una frequenza inferiore e quindi l'esecuzione delle operazioni richiederà più tempo. Particolare da notare è che il mio laptop è una versione fanless (senza raffreddamento attivo, ventole), il che aggrava significativamente il problema sopra menzionato.

Inoltre dai test effettuati con la scatter di MPI, si è riscontrato un problema di inefficiente allocazione di memoria nel momento in cui è necessario assegnare la stessa porzione di buffer al processo principale. Tale problematica assume una rilevanza particolare con video ad alta risoluzione, in cui la matrice di pixel e caratteri convertiti risulta essere notevolmente estesa. Al fine di risolvere questa problematica, è stata adottata un'efficiente soluzione di condivisione della memoria, mediante l'impiego delle operazioni Recv (bloccanti) e Isend (non bloccanti) di MPI. Tale strategia consente a ciascun processo di allocare esclusivamente la porzione di buffer necessaria, evitando di trasmettere e allocare segmenti inutilizzati di pixel.

L’analisi delle prestazioni del programma infine mostra che la versione single-core ha prestazioni migliori rispetto a quella multi-core sul laptop testato. Il thermal throttling e l’overhead, accentuato dal laptop fanless, è un fattore critico che riduce drasticamente le prestazioni globali.