Elaborato Parallel Computing: DES Password Decrypt

Lorenzo Bucci
lorenzo.bucci@stud.unifi.it

Abstract

Il progetto ha lo scopo di creare un programma che permetta di crackare una password codificata mediante l'algoritmo DES, a partire da un insieme di password note. Il crack è stato implementato in due versioni: una parallela su GPU e l'altra sequenziale su CPU. Il fine ultimo del progetto è dimostrare la maggiore velocità nell'eseguire il programma su GPU e il calcolo dello speed-up.

1. Introduzione

Questo progetto si pone come obiettivo la scrittura e l'esecuzione di codice che permetta di "decriptare" una password codificata con l'algoritmo DES.

In realtà, essendo la codifica DES unidirezionale, è possibile solo criptare ma non decriptare. Per questo è necessario codificare una alla volta le password da provare e confrontare la codifica ottenuta con la password criptata che si intende crackare. Quando i due valori saranno uguali, avremo scoperto la password.

1.1. Algoritmo DES

Data una sequenza di bit qualsiasi *a* e una chiave a 64 bit, l'algoritmo DES codifica la sequenza iniziale *a* restituendo una nuova sequenza *b* di lunghezza multipla di 64 bit.

Le dimensioni di *b* variano in base alla lunghezza di *a*: *b* è pari a 64 bit quando *a* ha lunghezza compresa tra 1 e 64 bit, *b* è pari a 128 bit quando *a* ha lunghezza compresa tra 65 e 128 bit… e così via.

Per semplicità in questo progetto sono state considerate solo password formate da 8 caratteri (64 bit) nell'insieme [A-Za-z0-9./], e con la chiave a 64 bit identica alla sequenza di caratteri da codificare. Da questa semplificazione ne consegue che la sequenza codificata restituita dall'algoritmo DES sarà sempre lunga 64 bit.

1.2. Attacco a dizionario

Dal momento che provare tutte le possibili combinazioni di password a 8 caratteri nell'insieme [A-Za-z0-9./] (attacco brute-force) prevedrebbe $64^8 = 2^{48} = 281.474.976.710.656$ tentativi, si è deciso di semplificare il problema assumendo che la password da crackare sia inclusa in un set di password note al cracker (attacco a dizionario).

Questa semplificazione, talvolta poco realistica, velocizza notevolmente l'esecuzione del crack: in questo progetto non è determinante scoprire le password criptate quanto confrontare le velocità con cui esse vengono crackate al variare di alcuni parametri e delle implementazioni utilizzate.

1.3. Framework e librerie

Il crack delle password usando un attacco a dizionario o un attacco brute-force è un problema imbarazzantemente parallelo: ogni tentativo è completamente indipendente dagli altri e dall'ordine con cui essi vengono eseguiti. Non c'è bisogno di far dialogare i thread né di attuare meccanismi di sincronia.

Data la natura del problema, uno dei framework più adatti e performanti per gestire problemi imbarazzantemente paralleli è CUDA, che per questo motivo è stato scelto per lo sviluppo della parte parallela di questo progetto. La parte sequenziale invece è stata sviluppata in C/C++.

L'intero progetto è basato su l'utilizzo di una libreria C preesistente che implementa l'algoritmo di codifica DES descritto in questa pagina web: http://page.math.tu-berlin.de/~kant/teaching/hess/krypto-ws2006/des.htm

2. Struttura del progetto

Il progetto è così strutturato: il primo passo consiste nella generazione di un set di *n* password casuali secondo i criteri descritti in 1.1. Il set appena generato formerà il dizionario di password noto al cracker. Successivamente vengono scelte dal set la prima, l'ultima e la password centrale. Queste tre password vengono criptate con l'algoritmo DES. Le password codificate ottenute saranno l'unico elemento a disposizione del crack, oltre al dizionario.

A questo punto parte la fase di crack vera e propria. I tre tentativi di crack sono stati eseguiti successivamente e non simultaneamente, in modo da analizzare come incida sul tempo la posizione della password all'interno del dizionario. Inoltre, ciascun tentativo di crack viene ripetuto per due volte cosicché l'influenza di eventuali interferenze di altri processi sia minimizzata.

Come già accennato, il crack è diviso in due parti: implementazione sequenziale e parallela. Entrambe le due implementazioni ad ogni esecuzione arriveranno allo stesso risultato ma con tempi diversi.

2.1. Implementazione sequenziale

L'implementazione sequenziale sviluppata è molto semplice e basilare. Un ciclo for scorre tutte le password del dizionario e via via le codifica con l'algoritmo DES. Se ad un'iterazione la codifica ottenuta corrisponde alla password codificata da crackare, il ciclo si interrompe e viene stampata a video la password in chiaro, altrimenti il ciclo for continua fino alla fine del dizionario.

2.2. Implementazione parallela

L'implementazione parallela è più articolata rispetto a quella sequenziale dal momento che è necessario gestire la memoria e i thread della GPU.

Inizialmente si copia nella memoria della GPU il dizionario e la password codificata da crackare. Quest'ultima è stata allocata nella memoria costante anziché nella memoria globale della GPU dal momento che non varia per tutta l'esecuzione, richiede solo 8 byte di memoria ed è letta più e più volte da tutti i thread.

In questa fase è inoltre necessario allocare due spazi di memoria che verranno utilizzati dal thread che riuscirà a crackare la password: nel primo il thread scriverà la password in chiaro trovata, il secondo spazio verrà usato dal thread come flag per avvertire gli altri che la password è stata crackata e che pertanto possono interrompersi.

Successivamente viene lanciato il kernel CUDA. Il kernel è strutturato in modo che ciascun thread sia responsabile di un certo numero di password del dizionario. Su un totale di T thread, le password del dizionario che il thread t verifica sono: t-esima, t-esima + T, t-esima + $2T$… fino ad arrivare ad un indice che va oltre la dimensione del dizionario e che fa quindi interrompere il thread stesso.

Il kernel è composto principalmente da un ciclo while dove la guardia verifica di volta in volta che il thread non abbia raggiunto la fine del dizionario e che un altro thread non abbia già crackato la password. All'interno del ciclo si ricalca sostanzialmente l'implementazione sequenziale: si codifica la password corrente del dizionario e si confronta il risultato con la codifica della password da crackare. Se non corrisponde, il ciclo continua con la prossima password. Altrimenti si avvertono gli altri thread impostando ad 1 il flag allocato precedentemente, attraverso un'operazione atomica CAS, e si scrive la password in chiaro trovata nella memoria globale.

Al termine del kernel, l'host acquisisce la password in chiaro dalla memoria della GPU e la stampa a video.

3. Risultati

Nei prossimi paragrafi saranno presentati i risultati sperimentali del progetto che dimostreranno come l'implementazione parallela sia di gran lunga migliore rispetto a quella sequenziale. Sarà anche calcolato lo speed-up che concretizzerà numericamente il guadagno ottenuto.

Tutti i risultati sono stati ottenuti su un hardware basato su CPU Intel Core i5-9400 e su GPU Nvidia GTX 1660 Ti.

3.1. Organizzazione dei thread

In questo progetto non ci sono vincoli specifici che limitino la scelta di quanti thread e blocchi avere. Solitamente si preferisce scegliere potenze del due in modo da utilizzare l'intera potenza della GPU ed è questa la linea che è stata seguita nel progetto.

Sono stati eseguiti diversi test per misurare le performance al variare delle dimensioni dei blocchi e del grid mantenendo costanti il numero di thread totali. Dai test effettuati non si sono registrate apprezzabili differenze di performance tra dimensioni di blocchi differenti, eccezion fatta per blocchi molto piccoli con meno di 64 thread.

Detto ciò, il numero di thread per blocco è stato fissato a 512, giusto compromesso tra il limite massimo di 1024 e valori più piccoli. Inoltre, scegliendo 1024 si potrebbe non sfruttare appieno la potenza dei CUDA cores su GPU diverse da quella usata per i test.

3.2. Risultati temporali

Il tempo di esecuzione di un tentativo di crack è stato misurato in secondi usando funzioni standard della libreria del C++11.

Nella versione sequenziale si è misurata la sola computazione del ciclo for: non è stata considerata la stampa a video della password in chiaro.

Nella versione parallela si è misurato il tempo di esecuzione dell'intero kernel, dal suo lancio fino alla restituzione del controllo all'host. Anche in questo caso non si è considerata la stampa a video.

Di seguito verranno presentati i risultati principali al variare delle uniche due variabili dell'intero progetto: n numero di password del dizionario e T numero di thread totali della versione parallela. Per evitare problemi come la divergenza o il non utilizzo del 100% della GPU, che possono falsare la comparazione delle performance con la CPU, alle due variabili sono stati assegnati solo valori potenze del due. Ai soli fini del crack questo vincolo non è assolutamente necessario e può essere rimosso.

Dalla Tabella 1 si percepisce subito come la posizione della password da crackare all'interno del dizionario influenzi notevolmente la performance. Se la password è la prima del set, il primo thread la individuerà subito e interromperà gli altri.

T	Implementazione parallela		
	Iniziale	Centrale	Finale
2^{10}	0,010	30,11	60,61
2^{13}	0,010	3,910	7,780
2^{16}	0,011	0,578	3,305
2^{19}	0,011	0,895	3,063
2^{22}	0,011	0,022	3,035

Tabella 1. Tempi di esecuzione in secondi dei tre tentativi di crack (password da crackare iniziale, centrale e finale nel dizionario) al variare del numero *T* dei thread. Si è considerato $n = 2^{24}$.

n	Implementazione sequenziale			Implementazione parallela		
	Iniziale	Centrale	Finale	Iniziale	Centrale	Finale
2^{20}	0,000	10,30	20,43	0,008	0,088	0,197
2^{22}	0,000	41,69	83,05	0,012	0,022	0,772
2^{24}	0,000	165,4	330,3	0,011	0,053	3,019
2^{26}	0,000	660	1320	0,012	0,147	12,10
2^{28}	0,000	2640	5280	0,010	0,560	48,70

Tabella 2. Tempi di esecuzione in secondi dei tre tentativi di crack (password da crackare iniziale, centrale e finale nel dizionario) al variare di *n* numero di password del dizionario. Si è considerato $T = 2^{20}$.

Viceversa, se la password è l'ultima, sarà necessario attendere che tutti i thread abbiano completato e solo nell'ultima iterazione dell'ultimo thread la password verrà crackata.

La Tabella 1 ci mostra anche come l'aumentare del numero dei thread totali diminuisca i tempi di esecuzione. Ovviamente non ha senso che il numero dei thread superi il numero di password del dizionario, ma si intuisce che la situazione ottimale sia $n = T$. Questa situazione ottimale si può ottenere fin tanto che *n* è abbastanza piccolo: se *n* assumesse valori nell'ordine del miliardo o superiori sarebbe impossibile per CUDA gestire un numero di thread così elevato.

Per questi motivi può esser ragionevole scegliere $T = n$ quando $n \leq 2^{20}$ e mantenere $T = 2^{20}$ quando $n > 2^{20}$.

La Tabella 2 mostra palesemente come la soluzione parallela sia più efficiente rispetto a quella sequenziale. Sebbene ci sia un leggero svantaggio per la GPU (dovuto all'overhead) nel caso la password sia la prima del set, in tutti gli altri casi non c'è paragone.

In Tabella 2 non sono stati calcolati i tempi di esecuzione della versione sequenziale per $n = 2^{26}$ e $n = 2^{28}$ dato l'elevato tempo richiesto. I valori segnati in rosso sono stati inseriti a mano dopo aver osservato come il tempo di esecuzione raddoppi al raddoppiare dell'indice nel dizionario.

Nei test non sono stati sperimentati valori di *n* maggiori di 2^{28} a causa di limiti tecnici del linguaggio C.

n	Iniziale	Centrale	Finale
2^{20}	0	117,0	103,7
2^{22}	0	1895	107,5
2^{24}	0	3120	109,4
2^{26}	0	4489	109,0
2^{28}	0	4714	108,4

Tabella 3. Speed-up del programma relativo alla Tabella 2

3.3. Speed-up

In Tabella 3 sono riportati i valori dello speed-up relativi ai risultati ottenuti nella Tabella 2. La versione sequenziale è stata ovviamente eseguita su un solo core, mentre l'implementazione parallela poteva beneficiare di una GPU con 1536 CUDA cores.

Il caso in cui la password da crackare sia centrale è un caso limite per lo speed-up. Questo perché se la password è esattamente quella centrale, nella versione parallela sarà proprio il primo thread che nel suo ciclo la individuerà, mentre la CPU deve arrivare fino a metà lista prima di crackarla. Ciò spiega perché talvolta si presenti speed-up superlineare.

Sono stati svolti alcuni test aggiuntivi, non riportati nelle tabelle, per individuare lo speed-up minimo. Il caso peggiore ipotizzato si verifica quando la password da crackare si trova in posizione *T-1* con $n \gg T$. La CPU impiegherà sempre lo stesso tempo per arrivare a *T-1* ma la GPU dovrà aspettare che venga messo in esecuzione l'ultimo thread per crackarla: più *n* è grande rispetto a *T* e più tardi partirà il thread *T-1*.

In certi casi sfavorevoli può accadere che l'algoritmo sequenziale sia più veloce di quello parallelo. Ad esempio, se in un dizionario di 2^{28} password quella da crackare avesse indice $2^{20} - 1$ la CPU impiegherebbe $\approx$ 20 s, la GPU $\approx$ 48 s!

In conclusione, si può determinare che in questo elaborato lo speed-up sia molto variabile (in base alla posizione della password) ed in generale sia mediamente pari a 100, valore ricavato dal caso peggiore sia per la CPU che per la GPU.