



IMPLEMENTAZIONE CUDA DI UN ALGORITMO PER LA RICERCA DI SOTTOSTRUTTURE COMUNI TRA MOLECOLE SU GPU

VIRGULTI FRANCESCO
DAVIDE VILLANI

ROAD-MAP

CORSI FORMAZIONE CUDA

STUDIO CODICE PYTHON

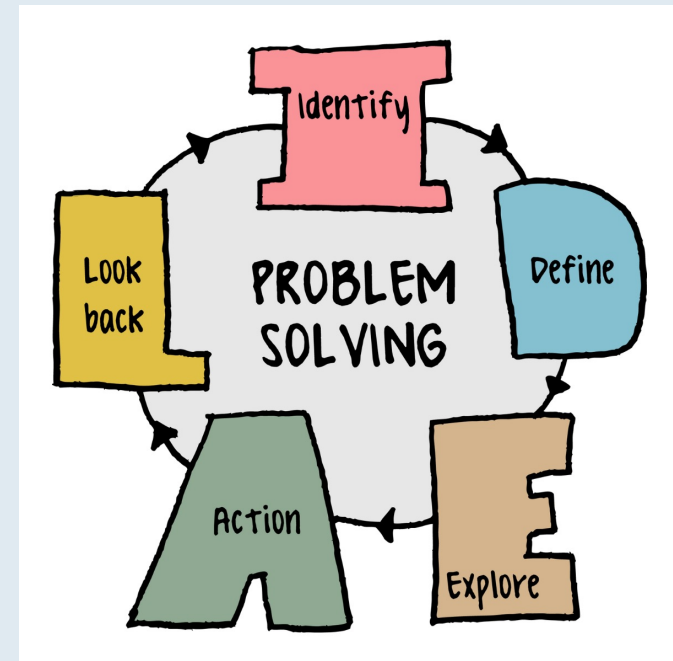
TRADUZIONE PYTHON – C++

SOLUZIONI PER LA PARALLELIZZAZIONE DEL CODICE

CREAZIONE ALGORITMO ITERATIVO

IMPLEMENTAZIONE IN CUDA

ANALISI DELLE PRESTAZIONI OTTENUTE



CORSI FORMAZIONE CUDA

Per comprendere l'architettura hardware per l'elaborazione parallela CUDA abbiamo seguito i seguenti corsi :

Getting Started with Accelerated Computing in CUDA C/C++

Fundamentals of Accelerated Computing with CUDA C/C++

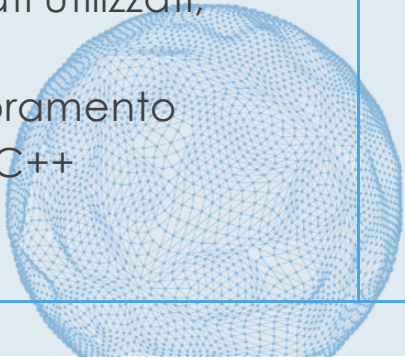
Abbiamo studiato gli strumenti e le tecniche fondamentali per accelerare le applicazioni C/C++ su GPU con CUDA. Abbiamo appreso come scrivere codice, configurare la parallelizzazione con CUDA, ottimizzare la migrazione della memoria tra CPU e GPU, e applicare queste conoscenze a un nuovo compito: accelerare un simulatore di particelle, originariamente solo per CPU, ottenendo significativi miglioramenti delle prestazioni. Alla fine del workshop, abbiamo avuto accesso a risorse aggiuntive per creare da soli nuove applicazioni accelerate con GPU.



STUDIO E TRADUZIONE CODICE PYTHON

Nel secondo step del nostro progetto, ci siamo concentrati sullo studio teorico del progetto "Ricerca della Massima Sottostruttura Comune nelle Molecole" di Pietro Beghetto, approfondendo in particolare il problema dell'MCSplit (Max Common SubGraph) tra due molecole. Dopo aver analizzato in dettaglio il codice Python fornito dall'autore e compreso il funzionamento di ogni singola funzione, ci siamo dedicati alla traduzione del codice in linguaggio C++.

Uno degli aspetti più complessi di questo processo è stato la scelta delle strutture dati appropriate, poiché il codice Python non specifica esplicitamente i tipi di dati utilizzati, rendendo necessario un lavoro di interpretazione e adattamento. Una volta completata la traduzione in C++, abbiamo constatato un significativo miglioramento nei tempi di esecuzione, attribuibile alla maggiore efficienza del linguaggio C++ rispetto a Python.



SOLUZIONI PER LA PARALLELIZZAZIONE DEL CODICE: CREAZIONE DI UN ALGORITMO ITERATIVO

L'MCSplit è un algoritmo ricorsivo, un approccio che non consente una corretta parallelizzazione del codice (i motivi sono dettagliatamente discussi nella Relazione). Per superare questo limite, abbiamo rielaborato l'algoritmo trasformandolo in una versione iterativa. Aniché effettuare chiamate ricorsive multiple, tutte le possibili chiamate vengono memorizzate in uno stack e vengono eseguite sequenzialmente, una dopo l'altra.

- ✓ L'MCSplit Iterativo ha mostrato notevoli miglioramenti nei tempi di esecuzione. Mentre nell'approccio ricorsivo ogni chiamata era vincolata alla precedente, l'inserimento di tutte le possibili chiamate all'interno di uno stack ha permesso di selezionare quelle con maggior probabilità di individuare la sottostruttura comune più ampia. Questo ha trasformato l'algoritmo da un processo casuale a uno euristico, migliorando ulteriormente l'efficienza complessiva.



IMPLEMENTAZIONE IN CUDA

Il codice che fino a questo punto abbiamo utilizzato in C++ faceva largo uso di librerie avanzate, come vector e list, che facilitavano la gestione delle strutture dati. Tuttavia, questi strumenti non sono disponibili nell'architettura hardware di CUDA. Inoltre, in C++, l'allocazione della memoria avviene in modo istantaneo e con costi limitati, mentre in CUDA è necessario allocare preventivamente tutte le risorse di memoria che si intende utilizzare.

Per adattarci a queste restrizioni, abbiamo dovuto sovrastimare le dimensioni delle molecole e riscrivere il codice per la terza volta, ottimizzandolo per l'ambiente CUDA.

