RIASSUNTO dell'incontro del 06/11/15 con l'ing. Buonanno - Univeristà La Sapienza di Roma - Centro Nazionale delle Ricerche.

A cura di Federico Luzzi.

Elaborazione parallela e gestione di dati acquisiti da sensori satellitari SAR

In data 6 Novembre 2015 abbiamo accolto con piacere nel nostro istituto un dottorando al lavoro presto il prestigioso CNR di Napoli, e attualmente concentrato nell'ambito di rilevazioni elettromagnetiche ambientali presso l'istituto IREA. In due ore ci ha brevemente introdotti ai suoi compiti all'interno dell'organizzazione stessa e ai metodi di lavoro utilizzati per portarli a termine, focalizzando in particolare l'attenzione sul ruolo dei sistemi informatici ed elettronici nell'elaborazione di dati satellitari nella valutazione delle deformazioni della superficie terrestre. Questa valutazione è resa possibile grazie all'utilizzo intensivo di speciali sensori SAR (acronimo per Synthetic Aperture Radar, radar ad apertura sintetica) di qualità, che permettono di effettuare rilevamenti ad alto grado di precisione sul movimento della nostra litosfera. Il funzionamento dei SAR stessi risulta di per sè abbastanza semplice, se esposto genericamente e senza addentrarsi nei particolari. Sono infatti posizionati su svariati satelliti dedicati alla fruizione del servizio, ed una volta in orbita generano, all'altezza della zona interessata dalla misurazione, un fascio di onde elettromagnetiche, che rimbalzando sulla superficie terrestre viene nuovamente intercettato dal dispositivo, che provvede a registrare in una memoria interna le informazioni relative alle caratteristiche dell'impulso. Dopo un determinato intervallo di tempo (in parte dipendente anche dal grado di obsolescenza del satellite e/o del sensore stesso) viene poi effettuata una seconda rilevazione, con modalità totalmente identiche alla prima, e vengono confrontate le caratteristiche delle due, concentrandosi in particolare sulla variazione di fase tra i due rilevamenti presi in esame. E' infatti questa variazione e la sua intensità che permette di determinare con buona precisione lo spostamento di un determinato punto dalla seconda rilevazione rispetto alla sua posizione iniziale nella prima. Ogni singola rilevazione viene registrata sotto forma di immagine in scala di grigi, e può essere definita come un'istantanea dell'energia elettromagnetica riflessa da ciascun elemento investito dal fascio sparato dal satellite orbitante. L'incrocio di due o più di questi "ritratti" dà origine ad un interferogramma (un'immagine in cui le differenze tra due o più immagini vengono evidenziate secondo una precisa scala di colori), e a sua volta l'incrocio e l'opportuna elaborazione di più interferogrammi dà finalmente vita alla vera e propria mappa di deformazione del suolo, visionabile su intervalli di tempo selezionabili a piacere e interpolabile con immagini satellitari per avere un'idea più chiara sulla zona considerata. Come è facile immaginare, la memorizzazione delle rilevazioni sullo spostamento di un determinato punto e la loro elaborazione sono operazioni molto onerose dal punto di vista del calcolo computazionale, soprattutto tenendo a mente che i punti presi in esame sono centinaia di migliaia e che ciascuno di essi contiene centinaia di misure. Ecco, a titolo esemplicativo, alcuni dati che possono dare un'idea della mole di informazioni che deve essere maneggiata quotidianamente per la realizzazione di un progetto di telerilevamento SAR; ogni singola rilevazione di un "ritratto" in scala di grigi occupa da 1 a 1.6 Gigabyte di dati, e le rilevazioni vengono ripetute periodicamente per le zone interessate. Si passa da un intervallo rilevativo di circa un mese per i satelliti più obsoleti a circa 5/6gg per i dispositivi più attrezzati e

tecnologici. Abbiamo quindi, nel peggiore dei casi, un'elaborazione esclusivamente in quanto a "ritratti" che si aggira sugli 8 Gigabyte mese su ogni singola area. Ovviamente tutte queste informazioni vanno memorizzate in uno o più database (un software Perl parallelizzato carica le informazioni su un DB Postgres) e non scordiamoci la parte computazionalmente (ma non solo) più onerosa, e cioè quella di elaborazione punto punto dei dati per la generazione dei risultati finali. Proprio in relazione a queste sorprendenti curiosità siamo stati introdotti alla parte dell'incontro che più verteva sulla parte elettro/informatica dell'incontro, e cioè quella che ha risposto alla domanda: come è possibile effettuare l'elaborazione di una mole così gigantesca di dati in tempo reale e senza ritardi? La risposta è stata presto svelata: si utilizza un cluster. Infatti, è stata effettuata una scelta architetturale che ha portato alla creazione di un cluster formato da 23 PC, e nel quale la vera forza computazionale è rappresentata dalle GPU (Graphic Processing Unit) NVidia Tesla K20, particolarmente adatte allo svolgimento di calcoli aritmetici ad altissima velocità. La necessità richiesta dal progetto era quella di poter eseguire il maggior numero di istruzioni identiche tra loro in modo parallelo, dando però in pasto a ciascuna di esse parametri differenti (quelli estrapolati dai singoli punti di ogni singolo interferogramma), abbattendo i tempi di esecuzione. E' proprio questo che fa una scheda video dotata di architettura SIMD (Single Instruction Multiple Data, secondo la classificazione di Flynn) quale la K20, dotata inoltre di ben 5 Gigabyte di memoria RAM, 2048 core totali e 13 stream multiprocessor (circa 160 core a processore). Il funzionamento della stessa è basato su architettura CUDA, sistema proprietario di NVidia e realizzato sostanzialmente tramite la presenza a cascata dei sopracitati stream multiprocessor (che altro non sono che multiprocessori dall'elevato numero di core), ciascuno dei quali può gestire un predeterminato numero di thread (in questo caso 1024) e dare un altrettanto predeterminato numero di risultati contemporaneamente (per la K20 sono multi per multiprocessore). La particolarità dal punto di vista software che incontriamo lavorando con le GPU in architettura CUDA è quella rappresentata dal fatto che non abbiamo vincoli di sincronizzazione gestibili a piacimento quali quelli offerti ad esempio nella programmazione C multithreading in ambiente Linux, e cioè semafori e mutex, ma solamente un meccanismo di basso livello chiamato barriera che ci dà un minimo di controllo sull'evoluzione parallela dei thread nel tempo. Come suggerisce il nome stesso, possiamo intendere la barriera come una specie di muro di contenimento: quando un determinato thread la raggiunge passa in uno stato di attesa, e continua nella sua elaborazione solamente quando tutti gli altri thread in esecuzione contemporaneamente nell'hardware avranno anche essi raggiunto la barriera stessa. La conoscenza dell'architettura CUDA e del metodo di lavoro dell'hardware è prerogativa assoluta per il programmatore che vuole approcciarsi a scrivere del software per uno qualsiasi dei dispositivi NVIDIA della serie K e non solo, ma bisogna anche essere consapevoli dello studio delle librerie messe a disposizione dall'azienda americana per programmare liberamente i propri prodotti che sicuramente attende chiunque voglia avventurarsi nella programmazione multithreading per GPU. Due dei principali svantaggi di questo tipo di programmazione sono la sua difficoltà di debugging e l'ostico linguaggio utilizzato (un C con registro e connotati che ricordano molto da vicino alcuni linguaggi di basso livello). La prima difficoltà è dovuta al fatto che l'operazione di debugging del codice può essere fatta solamente tramite monitoraggio diretto (quindi tramite appositi tool) e in tempo reale dei parametri fisici del dispositivo, in quanto siamo impossibilitati dall'ambiente di esecuzione hardware del codice ad un debugging prettamente visuale tramite

l'utilizzo di tradizionali breakpoint e printf, mentre la seconda è dovuta invece al fatto che al programmatore è completamente delegata tutta la parte di allocazione della memoria, prima lato host e poi lato GPU, con relative problematiche. Il metodo di programmazione delle GPU in ambiente CUDA si basa più in generale sulla definizione di funzioni di base dette kernel, e cioè un insieme di istruzioni che andranno eseguite, con dati diversi per ogni istanza, su ciascun core di ciascun stream multiprocessor, ottenendo risultati in maniera molto più velocizzata rispetto alla normale esecuzione sequenziale, che viene pur sempre inserita in alcune zone di codice. La definizione di un kernel (o di una qualsiasi funzione che desideriamo essere eseguita dalla GPU) avviene tramite la parola chiave __global__. Come l'ingegner Buonanno ci ha mostrato tutti i calcoli, sulla mole impressionante di dati prima accennata, vengono svolti da diversi algoritmi e in particolare come quelli già ottimizzati per GPU hanno evidenziato un ottimo incremento delle performance. Questo dimostra la potenza di uno strumento come il cluster, che se ampliato e sfruttato al massimo permettere di spingere la potenza di calcolo di un sistema fino al limite. Questo il riassunto di quello che il ricercatore del CNR ci ha insegnato nel corso di questo intervento, molto interessante e dai risvolti inaspettatamente incentrati sul nostro programma e in particolare su quello che proprio ora stiamo affrontando. E' stato inoltre interessante vedere un approccio pratico ad una programmazione di più basso livello rispetto a quella da noi affrontata, per arrivare quasi a toccare con mano la sua complessità logica dovuta all'avvicinamento del creativo pensiero umano alla fredda sequenzialità della macchina.