Algoritmi per il Calcolo Parallelo

studente: Pajola Luca
matricola: 116383

1.INTRODUZIONE

L'esercitazione consiste nello sviluppare un algoritmo di interpolazione utilizzando la tecnica di Neville. Lo sviluppo consiste nel generare un algoritmo risolutorio prima di tipo seriale e poi in parallelo, con diversi livelli di ottimizzazione. L'interpolazione verra' fatta per 512 e 200000 punti.

2.SVILUPPO

Per lo sviluppo troviamo un file,il quale contiene il codice seriale e parallelo. Per la realizzazione del codice e' stato utilizzato un codice di base di partenza fornito il quale genera 32 punti di base, che ci servono per interpolare. Come funzione di base si e' scelta il *seno* e dunque in base a questo abbiamo generato la serie di punti necessari. Il codice fornisce anche un vettore random di punti da interpolare. Degli algoritmi si osserva il tempo di esecuzione e l'errore medio, calcolato tramite i punti osservati (frutto dell'interpolazione) e i valori effettivi (applicando il seno sulle rispettive x).

2.1 Seriale

La prima fase di sviluppo consiste nel passare in un vettore temporaneo l'ascissa dei vari punti (i 32) e poi applicarne l'algoritmo di Neville. La difficolta' si trova nel traformare la formula in algoritmo.

2.2 Parallelo

Per la realizzazione del codice in parallelo utilizziamo il codice sviluppato nel seriale. I punti parallelizzabili risultano essere dunque 2:

1. copia in s dei valori py;
2. calcolo dei valori di s;

Nel primo kernel si e' tentato di parallelizzare il punto uno e lasciare il calcolo dei valori ad un unico thread.Per far cio' non si e' fatto altro che utilizzare un vettore condiviso. Il secondo kenrel modifica il precedente e punta a parallelizzare il secondo punto. Dato che, in questo caso, i thread devono modificare anche il vettore condiviso si e' utilizzato un *__synthreads()* per copiare i valori necessari al thread e calcolare il nuovo valore.

3.RISULTATI

tempi

	Seriale [ms]	kernel0 [ms]	kernel1 [ms]
512	10	2.04	0.236
200000	1470	596.925	55.291

speedup

	Seriale [ms]	kernel0 [ms]	kernel1 [ms]
512	1	5	42
200000	1	2	27

Analizzando i risultati vediamo come con la parallelizzazione del solo punto 1 la velocita' cambia. Nel complesso, la parallelizzazione risulta essere efficace; si nota, pero', che lo speed-up risulti diminuire all'aumentare dei punti. Questo perche', comunque, la generazione di thread non e' gratuita. Per quanto riguarda la qualita' dei valori ottenuti si nota, eseguendo il codice, che in tutti e 3 i casi, l'errore medio risulta essere evidente dopo la decima cifra decimale.