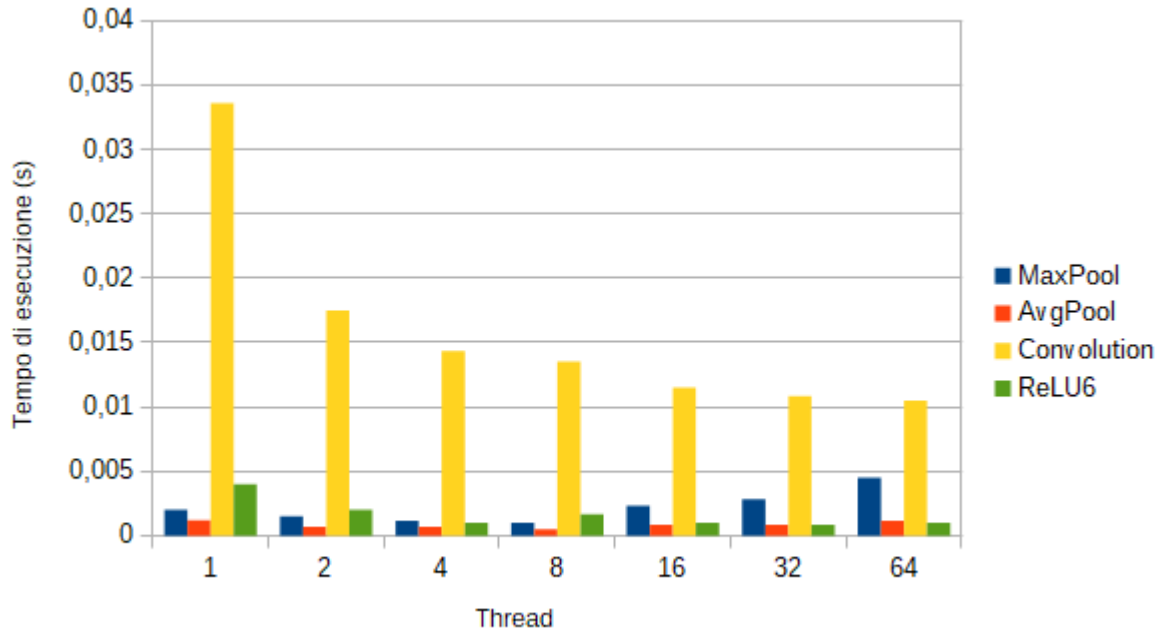
DANDREA FRANCESCO 218836

TPA II Esercitazione

Dopo aver scritto il codice sequenziale, questo è stato parallelizzato. Per assicurarsi che il codice parallelizzato fosse coerente con quello sequenziale nel file CNN.h e CNN.cpp sono state mantenute le funzioni anche in sequenziale.

Il codice è stato trasformato usando la direttiva "#pragma omp for" utilizzando le clausule di data sharing shared, private e default per assicurare che tutte le variabile abbiano la corretta proprietà.
Nel caso della convoluzione e nella operazione AvgPool dove è presente un'aggiornamento di variabile all'interno del for è probabilmente più sensato usare una clausula reduction, ma facendo diversi tentativi l'output senza questa calsusula risulta in ogni caso corretto. Reduction inoltre causa un tangibile aumento nei tempi di esecuzione. Il codice (presente comunque all'interno di AvgPool e Convolution parallelizzati) è stato quindi commentato. Un aumento si è misurato anche utilizzando la clausula schedule, questa così come reduction va a imporre condizioni che rallentano il programma. Per questi motivi il codice omp non prevede schedule. Nei for annidati invece la clausula collapse non porta nessun cambiamento nei tempi.

Per la scelta e lo studio del numero di thread più efficiente è stato scritto del codice atto proprio al raccoglimento dei tempi di esecuzione per le 4 operazioni a seconda del numero di thread. (il codice si trova commentato in fondo al main.cpp). Per ogni thread l'esecuzione è stata ripetuta almeno 5 volte. Da tenere presente che il codice prevedeva che le operazioni venissero eseguite ogni volta su nuove matrici e nuovi kernel generati casualmente tramite la funzione generate().
Il codice per testare le operazioni e stampare su file le matrici risultanti è impostato per una dimensione della matrice di input di 128x128 (presa da matrice.txt e kernel.txt). Per studiare i tempi di esecuzione, invece, le 4 operazioni sono state fatte su matrici di 512x512: in questo modo il tempo risulta sempre di grandezza misurabile (con matrici più piccole spesso il tempo restituito era pari a 0 per diversi thread e fare un confronto risultava difficile).
Di seguito il grafico che rappresenta a seconda del numero di thread i tempi per le 4 operazioni.



Come prevedibile la convoluzione richiede il maggior tempo. Aumentando il numero di thread l'efficienza aumenta chiaramente (nel caso non proposto di matrice 128x128 questo miglioramento non era così netto e già a 32 thread si notava invece un aumento nel tempo di esecuzione). Lo stesso non si può dire per le altre: MaxPool e AvgPool sembrano avere un minimo per un numero di thread pari a 8, l'operazione di ReLU6 superati i 2 thread sembra di efficienza abbastanza stabile. Un compromesso accettabile risulta essere quello per 8 thread o al massimo 16. Da ricordare che ciò vale per una matrice 512x512, variando la dimensione, come già accennato, 16 thread potrebbero già essere troppi e si potrebbe perdere efficienza.