Vettorizzazione di operazioni aritmetiche a 4 bit utilizzando registri a 64 bit sull'architettura x86-64

Tesi di Laurea in

Ingegneria Informatica

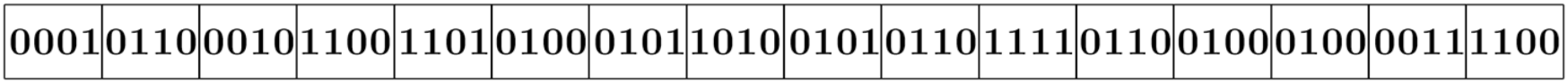
Candidato
Lorenzo Grassi

Relatore
Prof. Marco Cococcioni

UNIVERSITÀ DI PISA

Introduzione e Problema

- I *Large Language Model* utilizzano miliardi di parametri. Per risparmiare spazio, questi vengono a volte memorizzati su pochissimi bit.
- I processori non hanno istruzioni che lavorano direttamente su tipi più piccoli di 1 byte.
- Utilizzando tipi dato a 4 bit, possiamo inserire 16 valori in ogni registro *general-purpose*. In particolare usiamo interi *unsigned*.
- Il nostro obbiettivo è eseguire operazioni in parallelo tra i *nibble* corrispondenti di due registri, in pratica facendo in software quello che fanno le istruzioni SIMD.



Registro a 64 bit interpretato come 16 valori a 4 bit.

Soluzioni

- Utilizzando operazioni aritmetiche e bit a bit, abbiamo implementato varie procedure che calcolano somma, differenza, e prodotto dei *nibble* corrispondenti in modo vettorizzato. I 16 risultati sono anch'essi su *nibble* di un registro a 64 bit.
- Abbiamo scritto anche le varianti con saturazione, che in caso di overflow limitano il risultato al massimo (o minimo) possibile.

Esempio di esecuzione dell'algoritmo a doppio XOR per la somma. Questa esecuzione ha richiesto due passaggi.

Soluzioni

- Oltre alle semplici operazioni aritmetiche, abbiamo anche implementato operazioni più complesse dell'algebra lineare, in particolare:
  - Interpretando i due registri di input come vettori a 16 componenti, abbiamo scritto un algoritmo per calcolare il prodotto scalare.
  - Abbiamo implementato la moltiplicazione e accumulo su corsia (*multiply-accumulate lane*), che prende tre registri (`a`, `b`, `c`) e uno scalare (`lane`) come input (ciascuno come vettore di 16 valori a 4 bit) e, per ogni componente `i`, calcola `a[i] + b[i] * c[lane]`.
  - Infine, sfruttando quest'ultima, abbiamo scritto una procedura che calcola il prodotto tra matrici con elementi a 4 bit in modo vettorizzato.
  - Per queste ultime due operazioni abbiamo progettato anche una versione con saturazione.

Risultati

- Abbiamo confrontato queste operazioni vettorizzate con alternative non vettorizzate, e con funzioni basate su *lookup table*.
- La *lookup table* è risultata quasi sempre più lenta, tranne che per la moltiplicazione con saturazione.
- Rispetto all'approccio non vettorizzato, l'approccio vettorizzato è risultato tra le 3 e le 5 volte più veloce per quasi tutte le operazioni, e quasi 7 volte più veloce per la moltiplicazione tra matrici.
- Compilando con `-O3`, GCC ha vettorizzato il codice inizialmente scalare di alcune operazioni, utilizzando veri e propri registri SIMD. Queste ottimizzazioni competono e in molti casi superano le nostre operazioni vettorizzate su registri *general-purpose*.