#### Claudio Musmeci 1000056290

# Progetto Iot: Applicazione di una tecnica di Approximate Computing ad un programma IoT related

L’approximate computing si basa sull’abilità delle applicazioni di tollerare della perdita di precisione nell’output per scambiarlo con migliori performance ed efficienza energetica.

Tra le varie tecniche di approximate computing vi è quella definita come “Loop Perforation”.

“Loop Perforation” è una tecnica di approssimazione che permette di saltare alcune iterazioni all’interno di un loop. Vi sono diversi tipi di perforazione:

* small: consiste nello skippare una iterazione ogni 𝑛 iterazioni,
* large: consiste nell’eseguire una iterazione ogni 𝑛 iterazioni,
* random: consiste nell’eseguire un 𝑛% casuale del numero totale di iterazioni,
* ini: consiste nello skippare le prime 𝑛% del numero totale di iterazioni,
* fini: consiste nello skippare le ultime 𝑛% del numero totale di iterazioni.

I tipi di perforazione sono quindi configurabili sulla base del parametro “skip factor” n, dove n è un intero per i primi due tipi di perforazione, ed un numero reale compreso tra 0.0 e 100.0 per i rimanenti.

La tecnica di loop perforation scelta è quella di tipo large ed è stata applicata su un programma di image processing che implementa il Sobel operator, che è utilizzato nel campo dell’ image processing e computer vision, come algoritmo di edge detection, dove data una immagine di input viene creato un output in cui vengono enfatizzati i contorni dei soggetti.

Il codice sorgente del programma scelto è reperibile nella seguente repository github:

https://github.com/avanhatt/llvm-loop-perforation/tree/master/benchmarks/sobel

Il programma si occupa di applicare l’operatore di Sobel su una immagine di input in formato pgm (caratterizzata dall’essere realizzata in una scala di grigi) attraverso l’utilizzo della convoluzione del kernel Sobel per rilevare i cambiamenti orizzontali nell’immagine.

Le modifiche apportate al programma riguardano l’aggiunta del parametro skip\_factor all’interno dei loop su cui si vuole applicare la tecnica e l’inserimento dell’immagine in input che non verrà più fornita tramite riga di comando, ma è definita come un parametro.

Il core del programma è rappresentato dalla funzione sobel\_filtering(). Essa si basa su due blocchi di iterazione:

Blocco relativo all’immagine di input

Immagine che contiene testo, schermata, schermo

Descrizione generata automaticamente

I due cicli for più esterni permettono di scorrere i vari pixel dell’immagine, mentre i due cicli più interni si occupano di sommare i prodotti tra i valori del kernel weight e i valori dei pixel nell'intorno del pixel corrente, contribuendo così al calcolo del nuovo valore del pixel dopo l'applicazione del filtro Sobel.

Blocco relativo all’immagine di output

Immagine che contiene testo, schermata, Carattere

Descrizione generata automaticamente

I due cicli for più esterni si occupano di scorrere i vari pixel dell’immagine, mentre i due cicli for più interni calcolano nuovamente la somma pesata dei pixel nell'intorno del pixel corrente, ma questa volta contribuisce al calcolo dei nuovi valori dei pixel nell'immagine di output image2. I valori calcolati nel primo blocco consentono di applicare una trasformazione lineare a pixel\_value\_app.

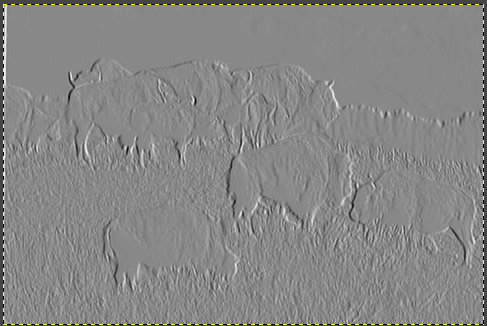
Si è scelto di utilizzare la seguente immagine di input:

Immagine che contiene mammifero, aria aperta, erba, bisonte

Descrizione generata automaticamente

## Caso 1: Nessuna approssimazione

Senza applicare tecniche di approssimazione, otterremo il seguente output.



Le prestazioni che si ottengono sono:

Immagine che contiene schermata, testo, Software multimediale, software

Descrizione generata automaticamente

Dove la maggior parte del tempo di calcolo è speso per il calcolo di pixel\_value (3 unità CPU, 5,56%) e pixel\_value\_app(5 unità CPU, 9.26%)

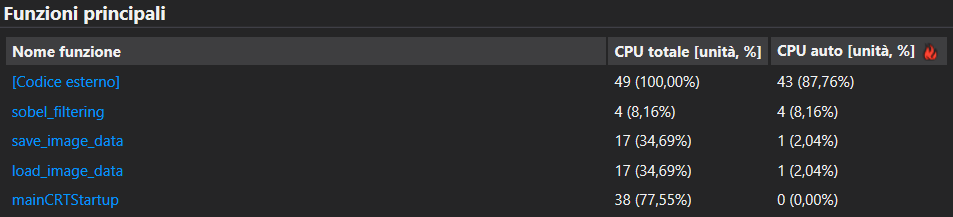
## Caso 2: skip\_factor = 2

Considerando una buona approssimazione con skip\_factor pari a 2 l’output è:

Immagine che contiene modello, tessuto, arte

Descrizione generata automaticamente

Le prestazioni che si ottengono sono:



Dove la maggior parte del tempo di calcolo è speso per il calcolo di pixel\_value (1 unità CPU, 2,04%) e pixel\_value\_app(1 unità CPU, 2.04%)

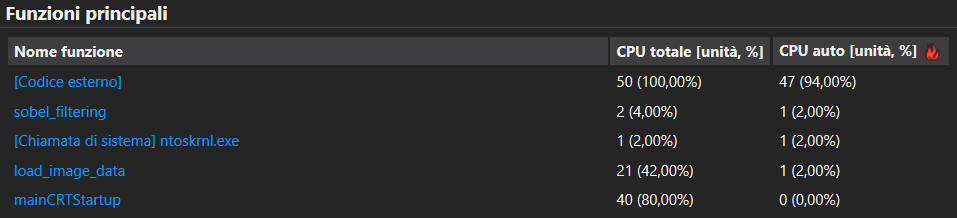
## Caso 3: skip\_factor = 3

Considerando una forte approssimazione con skip\_factor = 3 l’output è:

Immagine che contiene modello, Rettangolo, schermata, punto

Descrizione generata automaticamente

Le prestazioni che si ottengono sono:



Dove la maggior parte del tempo di calcolo è speso per il calcolo di pixel\_value (1 unità CPU, 2,00%) e pixel\_value\_app(1 unità CPU, 2.00%)

## Conclusioni

Il programma che implementa la funzione sobel\_filtering() presenta una buona resistenza all’approssimazione, in quanto nonostante lo skip\_factor venga raddoppiato o triplicato (e di conseguenza vengano utilizzati sempre meno pixel per il calcolo di image2) è comunque possibile ottenere un “perfored output” fedele a quello non approssimato.   
Per quanto riguarda le performance, limitatamente alla funzione sobel\_filtering() a cui è stata applicata la loop perforation:

|  |  |  |  |
| --- | --- | --- | --- |
| skip\_factor | CPU totale [unità, %] | pixel\_value | pixel\_value\_att |
| 1 | 10 | 3 | 5 |
| 2 | 4 | 1 | 1 |
| 3 | 2 | 1 | 1 |

Applicando uno skip\_factor pari a 2 abbiamo una riduzione del tempo di CPU pari al 60%.

Applicando uno skip\_factor pari a 3 abbiamo una riduzione del tempo di CPU pari al 80%.

Alle riduzioni del tempo di CPU corrisponde anche un minore consumo energetico da parte del processore stesso.