Programmazione di Sistema

A.A. 2022-2023

Questo documento inizia con una breve descrizione di regole e tempi per assegnazione e realizzazione dei progetti. Seguono le descrizioni dei progetti stessi, con sezioni distinte per le parti:

- Application Programming (A. Savino)
- OS Internals (S. Azimi)

Regole comuni ai progetti proposti dal Prof. Alessandro Savino e dal Prof. Sarah Azimi

Si riassumono brevemente le regole operative, comuni alle due parti del corso. Successivamente si descrivono dettagli tecnici dei progetti proposti

- I progetti sono opzionali e la loro valutazione si somma a quella degli esami scritti.

- Ogni studente può ottenere l'assegnazione di un solo progetto nell'ambito del suo percorso di studi (non è quindi possibile cambiare progetto oppure rifarlo dopo averne già **richiesto** uno). In linea di massima il progetto dovrebbe essere assegnato durante l'anno accademico di "nuova frequenza". Tuttavia, al fine di tenere conto del fatto che più studenti frequentano un corso e/o partecipano alle attività formative in anni successivi a quello di "nuova frequenza", uno studente può essere equiparato a nuovo frequentante se non ha mai sostenuto alcun appello (ritirato = sostenuto).

- I progetti sono svolti in gruppi di 2 o 3 persone (non si esclude, in casi particolari, il lavoro fatto da una sola persona, ma non si può garantire, in tali casi, una riduzione del livello di difficoltà e/o del lavoro richiesto)

- I progetti 2022-2023 andranno completati e consegnati entro la sessione di esame di Febbraio 2024. L'inizio del nuovo corso, nell'a.a. 2023-2024 cancellerà automaticamente tutti i progetti ancora in sospeso. Per la consegna, in fase di assegnazione, vi verrà fornito l'accesso ad un repository Github privato. L'organizzazione di tale repository è a totale discrezione del gruppo.

- I progetti saranno valutati a seguito di una breve presentazione (un colloquio online che includa anche la dimostrazione del prodotto sviluppato) dei candidati, durante una sessione di esame. Affinché sia valutato un progetto in una data sessione, questo deve essere condiviso con il docente di rifermento per il proprio progetto prima dell'esame scritto (di uno dei due, nel caso della sessione estiva) di quella sessione. Per la sola sessione Estiva 2023, le due date sono fissate in alternativa al 30 Giugno 2023 ed al 14 Luglio 2023.

- Il voto può essere diverso tra studenti dello stesso gruppo, in quanto verranno valutate, nel colloquio orale, le singole persone e i rispettivi contributi al lavoro. La valutazione potrà variare tra -2.0 e +6.0. Un progetto assegnato ma non completato sarà valutato con punteggio -2.0 (per tutti gli studenti del gruppo di lavoro). Una volta valutato, il voto del progetto non ha scadenza.

Per l'assegnazione, entro **Venerdì 16 Giugno** ogni gruppo che desidera fare un progetto è invitato ad inviare all'indirizzo alessandro.savino@polito.it un'e-mail con la seguente riga (il numero di matricole dipende da quanti studenti compongono il gruppo):
<matricola1> <matricola2> <matricola3> <titolo progetto scelto>
<id GITHUB matricola 1>
<id GITHUB matricola 2>
<id GITHUB matricola 3>
Prima della scadenza ci sarà la possibilità di avere chiarimenti e/o risposte a eventuali dubbi/domande.

Sezione 1 - Sezione di Sistemi Operativi

Progetto 1.1. Analisi Comparativa tra OS161 e altri Sistemi Operativi Open-Source All'avanguardia per Sistemi Embedded e Computer general purpose

Sezione 1 (fino a 2 punti):

L'obiettivo di questo progetto è condurre un'analisi comparativa approfondita tra OS161 e altri sistemi operativi open-source all'avanguardia progettati per sistemi embedded e computer ad uso generale. L'analisi mira a valutare la presenza di varie funzioni e caratteristiche essenziali all'interno dei sistemi operativi selezionati. Inoltre, il progetto valuterà la fattibilità e l'usabilità dell'implementazione/modifica/interazione di queste caratteristiche rispetto a OS161. Le funzioni da valutare includono, ma non si limitano a:

- System Calls
- Meccanismi di sincronizzazione
- Memoria virtuale e Memory Management Unit (MMU)
- Implementazione di diversi algoritmi di scheduling
- Altre caratteristiche rilevanti

Sezione 2 (fino a 4 punti):

Nel caso in cui una particolare funzione o caratteristica risulti assente nel sistema operativo all'avanguardia, lo studente che svolge il progetto sarà responsabile dell'implementazione della funzionalità mancante e dell'integrazione nella versione studiata del sistema operativo. Per dimostrare l'integrazione riuscita, verrà preparata una macchina virtuale con un'immagine del sistema operativo selezionato, mostrando la nuova funzionalità integrata.

Nota 1: Per ogni studente interessato a questo progetto, è disponibile un sistema embedded Zynq 7020 SoC integrato in una scheda di sviluppo PYNQ-Z2PYNQ-Z2, fornita dall'azienda Xilinx. La scheda Pynq incorpora un microprocessore ARM che può essere utilizzato per far eseguire il sistema operativo in studio.

Nota 2: Se uno studente esprime interesse per questo progetto, può essere organizzato un incontro individuale per discutere il progetto in maggior dettaglio.

Contatti

Prof.sa Sarah Azimi

Sezione 2 - Sezione di Programmazione

Progetto 2.1. Configurazione ed esecuzione ottimizzata di una rete neurale generica descritta con standard ONNX

Definizione del problema

Open Neural Network eXchange (ONNX) [1] è uno standard open source. Viene usato per definire le operazioni che un modello di Machine Learning, in particolare una rete neurale, necessita per eseguire un'inferenza su dei dati di ingresso.

L'utilizzo di un formato standard per definire tali operazioni consente di passare con semplicità un modello tra diversi framework (e.g. pytorch, tensorflow, MATLAB), e di semplificare la sua applicazione in fase di produzione. In questo caso sarà sufficiente realizzare un *interprete* che traduca la descrizione ONNX del modello in una sua implementazione, ad esempio usando un qualunque linguaggio di programmazione.

Error! Reference source not found. riporta un esempio di un file ONNX contenente la descrizione di una semplice rete, mentre **Error! Reference source not found.** mostra il grafo ottenuto da tale descrizione

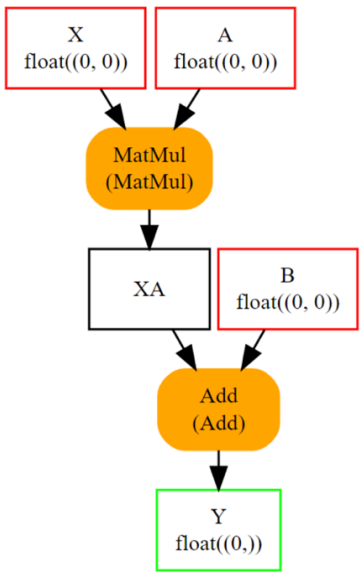

Figure 1-1: rappresentazione grafica dello snippet 1

Error! Reference source not found.

```
ir_version: 9
graph {
  node {
    input: "X"
    input: "A"
    output: "XA"
    op_type: "MatMul"
  }
  node {
    input: "XA"
    input: "B"
    output: "Y"
    op_type: "Add"
  }
  name: "lr"
  input {
    name: "X"
    type {
      tensor_type {
        elem_type: 1
        shape {
          dim {
          }
          dim {
          }
        }
      }
    }
  }
  input {
    name: "A"
    type {
      tensor_type {
        elem_type: 1
        shape {
          dim {
          }
          dim {
          }
        }
      }
    }
  }
  input {
    name: "B"
    type {
      tensor_type {
        elem_type: 1
        shape {
          dim {
          }
          dim {
          }
        }
      }
    }
  }
  output {
    name: "Y"
    type {
      tensor_type {
        elem_type: 1
        shape {
          dim {
          }
        }
      }
    }
  }
}
opset_import {
  version: 20
}
```

Obiettivi

Si chiede di creare un interprete ONNX [2] [3] utilizzando il linguaggio Rust. Le richieste sono in particolare le seguenti:

1. Creazione di un *parser* per estrarre dal file ONNX l'informazione necessaria per creare la rete. La fase di salvataggio andrebbe immaginata come una procedura di serializzazione su file (risultato simile a **Error! Reference source not found.**). Sarà quindi necessaria una operazione di de-serializzazione per poter accedere ricostruire il modello.

2. Implementazione di un sotto-insieme delle funzionalità ONNX [4]. Per testare il codice sviluppato si dovrà dimostrare che questo, partendo da un file ONNX contenete una rete già allenata, sia in grado di fare inferenza. Un gran numero di modelli allenati si può trovare online [5]. Scegliere un sotto-insieme di operazioni che permetta di fare inferenza con almeno due di essi (vedi tutti i campi op_type nei file onnx relativi ai modelli selezionati per sapere quali operazioni sono richieste).

3. Fare utilizzo estensivo di parallelizzazione. La rete creata partendo dalla descrizione ONNX dev'essere eseguita in modo efficiente, sfruttando le funzioni di parallelizzazione messe a disposizione dal Rust.

4. Opzionale. Rust consente di fornire binding verso altri linguaggi. A scelta, prevedere che le funzionalità siano esportate verso uno o più linguaggi a scelta (es.: C++, Python, ecc.).

Contatti

Prof. Alessandro Savino

Referenze

[1] «ONNX,» [Online]. Available: https://onnx.ai/onnx/index.html.
[2] «Interpreti,» [Online]. Available: https://github.com/microsoft/onnxruntime.
[3] «ONNXRUNTIME,» [Online]. Available: https://onnxruntime.ai.
[4] «Operatori,» [Online]. Available: https://onnx.ai/onnx/operators/index.html.
[5] «Modelli,» [Online]. Available: https://github.com/onnx/models/tree/main.

Progetto 2.2. Spiking Neural Networks e Resilienza

Introduzione

Le reti neurali spiking sono un tipo emergente di rete neurale il cui scopo principale è un'emulazione più fedele del funzionamento di un cervello biologico (Carpegna, 2021). I modelli classici di rete neurale, infatti, sono basati su funzioni di attivazione non lineari (sigmoide, ReLu, leaky ReLu), capaci sì di raggiungere risultati eccellenti sui problemi di classificazione più disparati, ma totalmente distanti dall'effettivo comportamento di un neurone biologico.

La prima caratteristica fondamentale delle reti spiking è il modo in cui l'informazione viene trasmessa da un neurone all'altro. Ciò avviene per mezzo di impulsi binari, in modo simile a quanto osservato all'interno di un cervello biologico, in cui i neuroni comunicano attraverso brevi impulsi di corrente. In questo caso il tempo diventa una parte fondamentale del modello: un singolo impulso è infatti in grado di trasportare una semplice informazione binaria (1 = impulso ricevuto, 0 = nessun impulso). Ciò che permette di codificare informazioni più complesse è la sequenza temporale degli impulsi, come mostrato in figura 1.

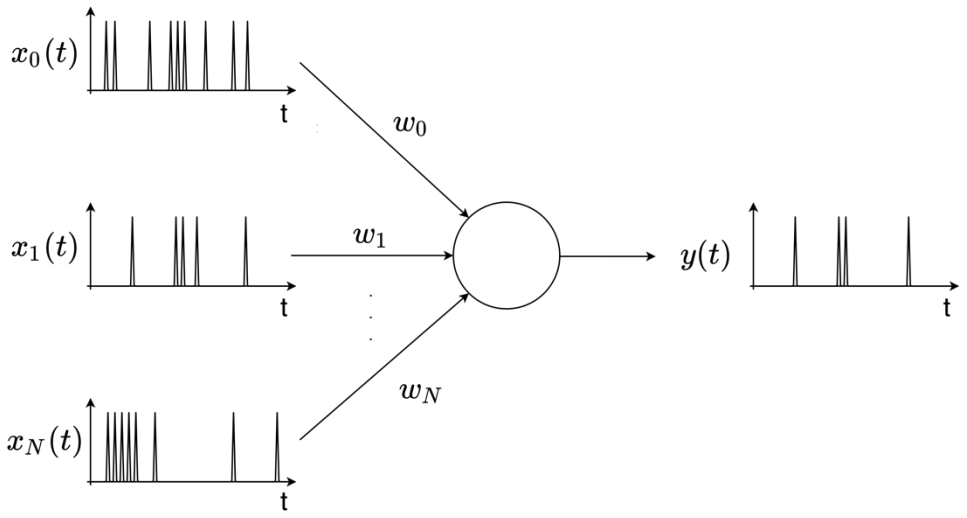

Figura 1: interfaccia di un neurone spiking

Il secondo punto da considerare è come gli impulsi vengano elaborati all'interno del neurone. Anche in questo caso il modello considerato prende ispirazione da quanto osservato in biologia. Il comportamento può essere sinteticamente descritto come segue:

1. Gli impulsi di ingresso vengono pesati dalle corrispondenti sinapsi, così come mostrato in figura 1.

2. Gli ingressi così pesati vengono poi sommati al potenziale di membrana (integrazione pesata degli impulsi di ingresso).
3. Se il potenziale di membrana supera un valore di soglia il neurone emette a sua volta un impulso e il potenziale viene resettato.
4. In assenza di stimoli esterni, ovvero di impulsi di ingresso, il potenziale decade verso il suo valore di riposo.

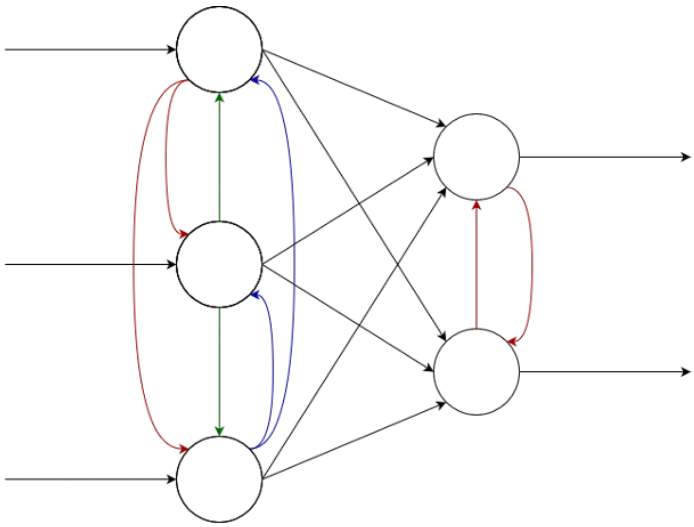

Figura 2: esempio di struttura della rete neurale. In nero le connessioni tra strati. Si noti che ogni neurone è collegato a tutti quelli dello strato successivo, ottenendo una struttura fully-connected. Le connessioni tra neuroni appartenenti allo stesso strato sono riportati invece con colori differenti.

Definizione del problema

Si chiede di creare uno strumento per lo studio della resilienza di una rete neurale spiking utilizzando il linguaggio Rust. Le richieste sono in particolare le seguenti:

1. L'interfaccia del neurone dev'essere generica e indipendente dal modello interno scelto, così come riportata in figura 1: il neurone riceve degli impulsi binari in ingresso e restituisce degli impulsi binari in uscita.
2. I parametri interni del neurone, come potenziale di reset, potenziale di riposo e soglia, devono essere configurabili.
3. Utilizzare una struttura fully connected. Il numero di strati e di neuroni in ognuno di essi dev'essere completamente configurabile. In aggiunta ogni neurone deve poter essere collegato a tutti i neuroni dello stesso strato. Una connessione di questo tipo, associata ad un peso negativo, è utilizzata spesso nelle reti spiking per limitare l'attività complessiva

dello strato: quando un neurone genera un impulso questo va a decrementare il potenziale di membrana di tutti i neuroni nello stesso strato, riducendo la probabilità che questi generino a loro volta un impulso.
4. Dev'essere fatto uso estensivo di tecniche di parallelizzazione per il funzionamento della rete.

Lo studio della resilienza deve essere supportato dalla possibilità, una volta costruita la rete, di studiare potenziali guasti in diverse componenti della rete: connessioni tra neuroni, zone di memoria che mantengono informazioni numeriche (come pesi, soglie, potenziali, ecc.) o blocchi elaborativi interni al neurone (come sommatore, moltiplicatore o comparatore di soglia). I guasti possibili, limitatamente a questo progetto sono a singolo bit, ovvero prevedono che solo un bit tra tutti quelli passibili di guasto si "rompa". La rottura corrisponde ad un comportamento funzionale ben preciso e catalogabile come segue: stuck-at-0 (il bit rimane fisso a 0, anche se richiesto il contrario) o stuck-at-1 (il bit rimane fisso a 0, anche se richiesto il contrario), transient bit-flip (il valore del bit viene invertito). La natura temporale dei guasti identifica come deve essere modellato: nei primi due casi, ogni volta che il bit viene usato, lo stuck-at-X corrispondente viene applicato forzando il bit al valore X per tutta la durata dell'inferenza (ovvero garantendone il valore a X ad ogni aggiornamento/scrittura), mentre nel tezo caso (transient bit-flip), tale forzatura avviene solo in un istante di tempo specifico ed eventuali nuove scritture successive non subiscono variazioni.

La simulazione di tali malfunzionamenti deve prevedere che l'utente possa fornire:
1. Una lista di componenti da verificare
2. Il modello di guasto (stuck-at-0, stuck-at-1, transient bit-flip)
3. Il numero di occorrenze dei guasti da verificare
4. La normale sequenza di input della rete stessa.

A fronte dei dati in questa lista, la simulazione deve ripetere l'inferenza, per tutti gli input, tante volte quante è il numero di occorrenze dei guasti indicato. Per ogni input deve selezionare in modo casuale un bit appartenente a uno dei componenti da verificare e inserire il guasto richiesto.

Esempio: una configurazione che indichi:
1. Soglie, pesi e potenziali di membrana
2. Stuck-at-0
3. 100 guasti

Richiederebbe 100 ripetizioni di tutte le inferenze, selezionando a caso un bit su cui applicare lo stuck-at-0 tra tutti i bit di soglie, pesi e potenziali.

Utilizzare il già citato LIF come modello interno al neurone per lo sviluppo ma nella scelta della tecnica implementativa, sfruttare i paradigmi messi a disposizione dal linguaggio per generalizzare il modello e rendere possibili definizioni basate su funzioni diverse. La sezione successiva ne riporta i dettagli matematici.

Non è richiesta la parte di allenamento della rete, si può supporre di avere a disposizione gli iper-parametri (pesi delle sinapsi e potenziali di soglia) già allenati. Si supponga inoltre di avere gli ingressi già codificati in forma di impulsi. Si forniscano in uscita gli impulsi generati dalla rete.

Leaky Integrate and Fire (LIF)

Nel modello LIF la membrana viene descritta come il parallelo di una capacità e una resistenza. L'evoluzione temporale del potenziale di membrana può essere descritta come segue:

$$V_{mem}(t_s) = V_{rest} + [V_{mem}(t_{s-1}) - V_{rest}] \cdot e^{-\frac{t_s - t_{s-1}}{\tau}} + \sum_{i=0}^{N} s_i \cdot w_i$$

$$Se\ V_{mem} > V_{th} \ \Rightarrow \ V_{mem} = V_{reset}$$

Dove $V_{mem}(t)$ è il potenziale di membrana, t_s l'istante di tempo in cui vengono ricevuti uno o più impulsi, $V_{mem}(t_{s-1})$ il valore del potenziale di membrana nell'istante t_{s-1} , ovvero nell'ultimo istante in cui è stato ricevuto un impulso, V_{rest} il potenziale di riposo, $t_s - t_{s-1}$ la distanza di tempo tra due impulsi di ingresso, τ la costante di tempo della membrana, data dal prodotto tra la sua capacità e la sua resistenza, N il numero di ingressi, s_i l'impulso ricevuto dall'ingresso i -esimo, quindi 1 per gli ingressi attivi nell'istante e 0 per tutti gli altri, il peso relativo all'ingresso i -esimo, V_{th}il potenziale di soglia e V_{reset} quello di reset.

In questo modo l'elaborazione può essere di tipo event-based: il neurone svolge solo i calcoli strettamente necessari nel momento in cui uno spike di ingresso ne forza l'aggiornamento (in t_s). Gli impulsi propagati all'interno di una rete spiking sono generalmente sparsi e un approccio di questo tipo può ridurre considerevolmente il numero di calcoli richiesti.

Contatti

Alessio Carpegna (alessio.carpegna@polito.it)
Prof. Alessandro Savino

Bibliografia

Carpegna, A. (2021, Ottobre). Design of an hardware accelerator for a Spiking Neural Network. (accessibile tramite il sistema bibliotecario di ateneo)