Programmazione di Sistema

A.A. 2021-2022

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. Di Carlo)

Regole comuni ai progetti proposti dal Prof. Alessandro Savino e dal Prof. Stefano Di Carlo

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 fatto 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 successive a quello di "nuova frequenza", uno studente può essere equiparato a nuovo frequentante.
- 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 2021-2022 andranno completati e consegnati entro la sessione di esame di Febbraio 2023. L'inizio del nuovo corso, nell'a.a. 2022-2023 cancellerà automaticamente tutti i progetti ancora in sospeso.
- I progetti saranno valutati a seguito di una breve presentazione (un colloquio) 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.
- 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 3 Giugno** ogni gruppo che desidera fare un progetto invii 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>
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. OS161 Scheduler basato su AI

Definizione del problema.

La costruzione di uno scheduler coinvolge, potenzialmente, molti parametri e caratteristiche di ogni processo da eseguire. Essendo il numero di questi parametri troppo elevato per essere raccolti, memorizzati e gestiti in tempo reale, non vengono realmente utilizzati dalla maggior parte degli scheduler. Gli scheduler attualmente utilizzati di fatto improntano la scelta della sequenza di esecuzione dei processi basandosi su euristiche. L'esplosione degli algoritmi di Machine Learning offre, oggi, la possibilità di derivare modelli a partire da moli di dati prima non analizzabili tutte insieme. Questo apre la possibilità alla definizione di algoritmi di scheduling basati sul profiling delle applicazioni.

Obiettivi

Il progetto si divide in tre parti:

1. Introdurre all'interno dello scheduling di OS161 funzionalità di profiling delle applicazioni. Questi dati devono essere definiti dallo studente e devono essere raccolti in un formato utile alla seconda parte.
2. Sviluppare un modello di scheduler a partire da una rete neurale. I dati racconti nel punto precedente devono essere opportunamente suddivisi in training e test set, ed il modello finale deve essere esportabile per essere utilizzato nella terza ed ultima parte.
3. Modificare il processo di scheduling sostituendolo con la versione eseguibile del modello generato al punto precedente. Il modello deve risultare un input da fornire in fase di compilazione, così da permettere la ripetizione delle prime due parti nel caso fosse necessario aggiornare il modello di predizione.

La frase di training del modello non fa parte del sistema operativo e può pertanto essere eseguita con software e librerie di altro livello, con libertà totale nella scelta del linguaggio di programmazione. La fase di test richiede invece l'integrazione del modello addestrato all'interno di OS161, pertanto, limitando la scelta del linguaggio e della tipologia di modelli scelti sulla base delle restrizioni imposta dal kernel di OS161.

Contatti

Prof. Stefano Di Carlo

Referenze

- An Artificial Intelligence based scheduling algorithm for demand-side energy management in Smart Homes (https://www.sciencedirect.com/science/article/pii/S0306261920315555)
- Process Scheduling using Machine Learning (https://www.engr.scu.edu/~mwang2/projects/Schedule_usingMachineLearning_14s.pdf)
- Improvising process scheduling using machine learning (https://ieeexplore.ieee.org/document/9012330)

Progetto 1.2. Studio della mappatura di memoria di un Sistema Operativo Real-Time

Definizione del problema

Un sistema operativo Real-Time (RTOS) è un sistema operativo in cui la gestione delle operazioni è subordinata alla necessità di completare i diversi task in tempi predefiniti. Al fine di ottemperare a tale necessità, un RTOS è spesso fornito nella forma di librerie da compilare insieme alle applicazioni, per ottenere un unico eseguibile che include le parti necessarie alla gestione dell'HW e quelle per ottenere l'applicazione. In questo progetto, partendo da alcune documentazioni e repository di esempio, si richiede di sviluppare versione di FreeRTOS che modifichi la modalità di mappatura/allocazione della memoria del sistema.

Obiettivi

FreeRTOS è un RTOS aperto, sviluppato in C, che permette di essere compilato ed eseguito sia su HW sia come processo. Il primo obiettivo è quello di studiare il sistema operativo (organizzazione, configurazione e personalizzazione) evidenziando le strutture dati mantenute dal sistema operativo per gestire il suo ciclo di vita. In seguito, predisporre un ambiente che consenta (con CMake) di compilare il FreeRTOS su un sistema Host (potenzialmente supportando sia Linux/OSX sia Windows) in modalità Simulatore. Si richiede di prestare particolare attenzione alla possibilità di semplificare il più possibile la configurazione, ivi inclusa la definizione dei task utente da eseguire.
Quando il simulatore fosse disponibile, il secondo passo del progetto richiede di modificare il sistema di gestione della memoria. Infatti, il sistema di base consente di distinguere tra allocazione in stack ed allocazione in heap. Per l'allocazione nell'heap sono disponibili diverse strategie. In tutti i casi, le strutture per il mantenimento della memoria dei task così come quelle del sistema operativo non possono essere separate in modo netto, permettendo, per esempio, di allocare tutta la memoria del sistema operativo in un banco (ad esempio in un preciso range di indirizzi) e quella dei task in un altro.
L'obiettivo di questa fase è pertanto quello di modificare l'architettura di allocazione della memoria per permettere di aggiungere questo grado di configurabilità al sistema e di costruire un dimostratore utile allo scopo. Tale dimostratore deve essere fornito sotto forma di simulazione, progettandola in modo da visualizzare tramite log l'effettiva completa separazione della memoria dei vari task da quella dell'OS. In tal senso, si consideri di intervenire anche sul sistema di log implementato da FreeRTOS apportando modifiche ove necessario.
Opzionalmente, una volta separata la memoria, si proponga una strategia per il mantenimento della memoria del OS in modalità duplicata (cioè dove ogni byte di dato è duplicato in una differente zona di memoria e sia possibile verificare e segnalare eventuali discordanze di contenuto). Questa strada comporta modifiche sostanziali al codice del OS.

Contatti

Prof. Alessandro Savino

Referenze

- https://www.freertos.org/FreeRTOS-simulator-for-Linux.html
- https://www.freertos.org/FreeRTOS-Windows-Simulator-Emulator-for-Visual-Studio-and-Eclipse-MingW.html
- https://github.com/megakilo/FreeRTOS-Sim
- https://github.com/alxhoff/FreeRTOS-Emulator

Sezione 2 - Sezione di Programmazione

Progetto 2.1. Ebook Reader

Definizione del problema

Un ebook reader è un programma basato su GUI che permette ad un utente di aprire file contenenti libri da visualizzare pagina per pagine, con la possibilità di tenere traccia del punto in cui si è arrivati, cambiare la grandezza del font, visualizzare a pagina singola o doppia, ecc.

Obiettivi

L'obiettivo primario è quello di sviluppare un programma per la gestione di libri in formato EPUB, lavorando con la libreria Rust DRUID (vedere link nelle referenze). Si cerchi di personalizzare al massimo l'interazione con l'utente, eventualmente sfruttando eventualmente altri programmi come riferimento (sia per funzionalità di base, sia per difetti da migliorare). La strutturazione del codice faccia ampio uso di test per sviluppare e verificare le singole funzionalità.
Inoltre, si permetta all'utente di attivare una modalità "correttore di bozza" che consenta di modificare il contenuto in presenza di errori (esempio errori di battitura), andando a generare un nuovo file ad ogni salvataggio.

Il secondo obiettivo è quello di integrare il sistema di fotocamera con OCR (vedere il link in reference) per mettere insieme il possesso del libro in formato cartaceo con quello in digitale in uno dei seguenti modi:

1. Dal libro che si stava leggendo su carta, saltare al punto a cui si è arrivati facendo la foto alla pagina
2. Operare l'inverso, cioè da due pagine riconosciute indicare a che pagina si trova il testo a cui si è arrivati sulla versione digitale.

Contatti

Prof. Alessandro Savino

Referenze:

- DRUID: https://github.com/linebender/druid
- Libri privi di copyright (legali): https://www.gutenberg.org
- Metodi di OCR con RUST: https://www.linkedin.com/pulse/ocr-rustleptess-tesseract-ha%C3%AFkel-ouaghrem/

Progetto 2.2. Bayesian Networks e parallelismo

Definizione del problema

Le reti bayesiane sono uno strumento di machine learning volto a sfruttare il Teorema di Bayes sulle probabilità condizionate per risolvere problemi di inferenza inesatti (es: la classificazione del contenuto di e-mail, la predizione di malattie, ecc.). Per poter risolvere il problema, è necessario innanzitutto modellare la dipendenza causale tra le variabili identificate nel problema stesso e costruire un grafo (orientato ed aciclico). Inoltre, per ogni variabile è necessario identificare un sottoinsieme di stati e di probabilità associate allo stato. Una volta costruito il grafo, degli algoritmi di inferenza sono in grado di elaborare tutte le probabilità condizionate seguendo la gerarchia di precedenze rappresentata dai collegamenti del grafo stesso. Tali algoritmi sono divisi in due gruppi: esatti ed approssimati. La differenza, legata alle prestazioni, si basa sul principio per cui i risultati possano essere forniti con un margine di errore se questo viene fatto corrispondere ad un risparmio in tempi di esecuzione.

Obiettivi

Il progetto mira a costruire una libreria per descrivere reti baysiane che sia provvista di un motore di inferenza per reti bayesiane basato su un algoritmo a scelta (preciso o approssimato) scelto dalla letteratura. Al fine di distinguere il lavoro svolto da quelli già presenti in letteratura (si usi l'articolo tra le referenze reperibile attraverso il servizio bibliografico del Politecnico), si richiedono una serie di vincoli di sviluppo:

- La struttura del grafo, ivi incluse le Conditional Probability Table (CPT) dei nodi, devono basarsi su strutture con semantica COW e per poter minimizzare l'occupazione di memoria in caso di strutture identiche.
- L'algoritmo scelto deve essere analizzato in termini di parallelizzazione.
- La rete sia modificabile attraverso opportuni metodi e/o tratti e sia caricabile o salvabile nel formato XDSL della libreria SMILE.

Contatti

Prof. Alessandro Savino

Referenze:

- XDSL file format: https://support.bayesfusion.com/docs/
- Haipeng Guo, William Hsu, Survey of Algorithms for Real-Time Bayesian Network Inference

Progetto 2.3. Spiking Neural Networks e parallelismo

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.

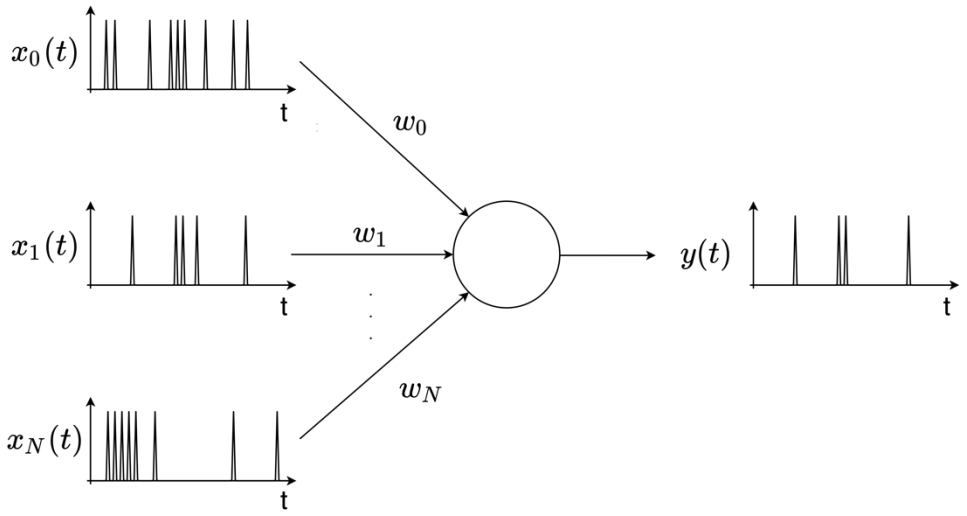

Figura 1: interfaccia di un neurone spiking

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.

Definizione del problema

Si chiede di creare una libreria per la definizione 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 il già citato LIF come modello interno al neurone. La sezione successiva ne riporta i dettagli matematici. Nella scelta della tecnica implementativa, sfruttare i paradigmi messi a disposizione dal linguaggio per generalizzare il modello.
4. 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.
5. Dev'essere fatto uso estensivo di tecniche di parallelizzazione per il funzionamento della rete.
6. 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.
7. Si supponga di avere gli ingressi già codificati in forma di impulsi. Si forniscano in uscita gli impulsi generati dalla rete.

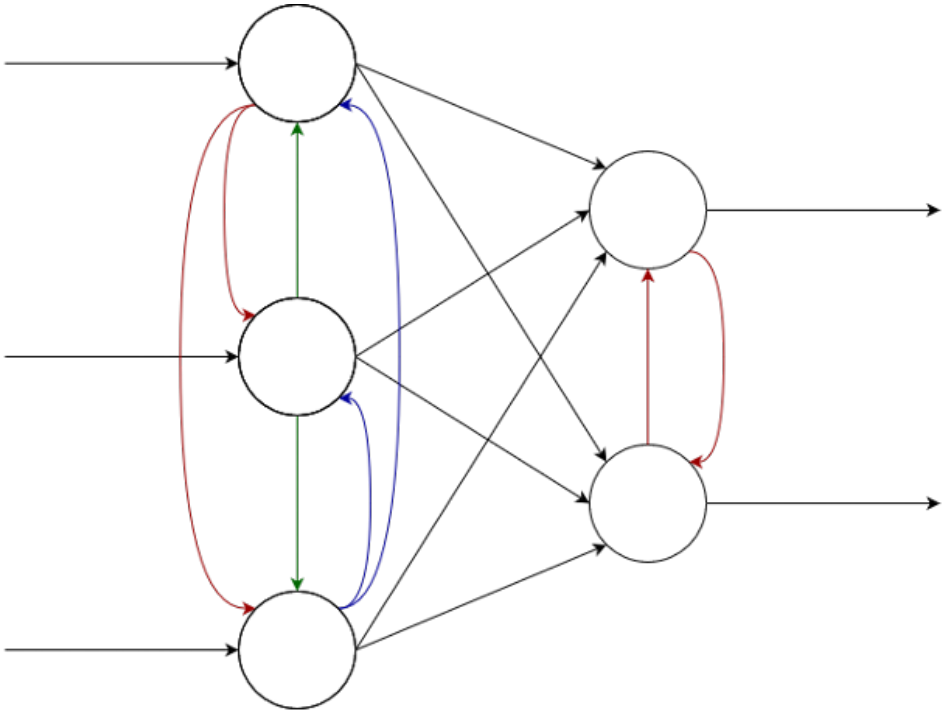

Figura 2: esempio di struttura della rete neurale. In nero le connessioni tra strati. Si noti che ogni neurone è collegato a tutti quelli dello strato successivo, ottenendo una struttura fully-connected. Le connessioni tra neuroni appartenenti allo stesso strato sono riportati invece con colori differenti.

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)