Time Series Pattern Recognition

Edoardo Sarri

Parallel Computing

Febbraio 2026

Introduction

Algoritmo per il riconoscimento di pattern in serie temporali:

- Input: Serie temporale T con N timestamps.
- Query: Pattern Q di lunghezza M ($M \ll N$).
- Output: Indice i che minimizza la distanza tra Q e T.

Versioni

Sono state realizzate due implementazioni:

- Sequenziale e CPU-based.
- Parallela e GPU-based (CUDA).

Introduction

Dataset

Human Activity Recognition (HAR) dataset: cattura il movimento di umani in gionate tipo con i sensori dello smartphone.

- Dimensioni: 6 (3 acc + 3 gyro).
- Samples: 2.56s a 50Hz; 10299 istanze da 128 timestamps.

Approccio

Sliding Window: per ogni istante temporale *i* si calcola la distanza tra la query e la porzione di input della stessa lunghezza.

Introduction

Input e Query

- Input:
  - Concatenazione di training e test set.
  - Data augmentation: Più istanze aggiungendo un rumore uniforme $\epsilon = \pm 0.01$.
- Query:
  - Sottosequenza estratta casualmente dall'input.
  - Aggiunto rumore Gaussiano con $\mu = 0, \sigma = 0.01$ per avere $SAD > 0$.

Introduction

Distanza

Sum of Absolute Differences (SAD). Date D dimensioni:

$$D(i) = \sum_{m=0}^{M-1} \sum_{d=0}^{D-1} |Q[m,d] - T[i+m,d]|$$

L'obiettivo è trovare $i^* = \arg\min_i D(i)$.

Sanitizer

Per trovare erorri a run-time è stato usato AUBSan

Sequential

L'esecuzione sequenziale segue quattro step:

1. **Data Loading:** In data_loader.cpp si carica il file di input.
2. **Query Loading:** In query_loader.cpp si carica il file di query.
3. **Pattern Matching:** In SAD_distance.cpp si calcola la distanza SAD trovando i^*.
4. **Reporting:** Si restituiscono le statiche e i risultati.

Sequential

Data Loading

Il file di input può raggiungere grandi dimensioni (≈ 3.5 GB).

- **Problema:** Overhead con gli steam per copie multiple e system calls.
- **Soluzione:** Memory Mapped I/O con `mmap`.
- **Ottimizzazione:** Si indica al kernel l'accesso sequenziale al file con `madvise` e il flag `MADV_SEQUENTIAL`.
- **Risultato:** Parsing ridotto da $\approx 47s$ a $\approx 13s$.

Sequential

Memory Layout

AoS memory pattern: $[x_{i,1}, \ldots, x_{i,D}, x_{i+1,1}, \ldots], i \in [0, N - D]$.

- **Vantaggio:** Località spaziale.
- **Accesso:** Con $Index(t, d) = buffer[t \cdot D + d]$.
- **Vettorializzazione:** Compiling con flaf `-march=native` e keyword `restrict` per permettere eliminare ambiguità al compilatore.
- **Padding:** Allineamento dei dati a multipli di 64 Byte:
  - 6 float per timestamp $\Rightarrow$ 24B.
  - Aggiunti 2 float fittizi per raggiungere 32B per timestamp.

Sequential

Early Abandoning

Interruzione del calcolo della distanza se la somma parziale supera il minimo globale corrente.

- **Speedup:** $\approx 2.87\times$ se attivato.
- **Baseline:** Disattivato per confronto equo con la versione parallela.

Parallel

L'esecuzione parallela segue i seguenti step:

1. **Data Loading:** Caricamento dati e query. Come nella versione sequenziale, ma in più si una pinned memory.
2. **Trasferimento H2D:** Si carica in GPU i dati grezzi.
3. **Kernel:** Ogni thread calcola la SAD per un timestamp.
4. **Trasferimento D2H:** Si porta in memoria host i risultati parziali.
5. **Final Reduction:** L'Host cerca il minimo globale tra i candidati ricevuti dalla GPU.

Parallel

Memory Layout

SoA memory pattern: $[x_{1,i}, \ldots, x_{N,i}, x_{1,i+1}, \ldots], i \in [0, D-1]$.

- **Problema:** Con AoS gli accessi non sono coalesced all'interno del warp.
- **Soluzione:** Con SoA il primo thread che accede alla sua dimensione di cui ha bisogno, porta in memoria anche la dimensione necessaria agli altri.
- **Accesso:** Con $Index(t, d) = d \cdot N + t$.

Parallel

Memory Hierarchy

Strategia per minimizzare la latenza della Global Memory:

- **Global Memory:** Dataset di input.
- **Constant Memory:** Query (Q).
  - Accesso rapido si una cache.
  - I dati non sono trasportati dalla global memory per ogni thread.
  - Query di 64 time stamps occupa $64 \cdot 6 \cdot 4 = 1536$ Byte.
- **Shared Memory:** Tiling dell'input e riduzione parziale del blocco.
- **Registers:** Memoria locale al thread per accumolare SAD.

Parallel

Shared Memory

- **Tiling:** Accessi alla global memory ridotti di un fattore $\approx M$.
  - I thread del blocco collaborano per caricare la parte di input che serve a tutti dalla global memory.
  - Ogni thread calcola la SAD usando i dati in Shared Memory, locale rispetto allo SM e molto più veloce.
- **Riduzione:** Minimo locale all'interno del blocco con il pattern Reduction.
  - Riduce le scritture in global memory da N a $\lceil N/M \rceil$.
  - Riduce il trasporto dalla global memory alla memoria host.

Parallel

Grid

Grid 1D: il thread i-esimo calcola la distanza tra query e il timestamp i-esimo.

- **Block Size:** 256. Le motivazioni sono:
  - Multiplo di 32 (warp size).
  - Usando la riduzione locale al blocco, le scritture in global memory sono ridotte del $\frac{1}{1/M} \approx 99.6\%$.
  - Blocchi più grandi portano a un overhead sulla barriera prima della riduzione locale al blocco.
- **Grid Size:** $\lceil N/256 \rceil$.
- Index: $tid = blockIdx.x * blockDim.x + threadIdx.x$.

Analysis

Il dataset è stato aumentato con un `multiplier=50`. La dimensione finale su cui sono fatte le analisi è 3.5 GB.

Il server usato (papavero):

- Architettura: x86-64
- OS: Ubuntu 22.04.4 LTS
- Kernel: Linux 6.8.0-52-generic

Analysis - Sequential

Tempo totale

- $Q = 64$ (default): 38.8 secondi.
- $Q = 128$: 69.4 secondi.

Ealing Abandoning

Il tempo del matching (senza I/O) è:

- **Senza** (default): 25.35 secondi.
- **Con**: 8.84 secondi.
- **Speed-up**: 2.87.

Analysis - Sequential

Profiling

- **Data Loading**: 65% del tempo totale.
- **Matching**: 30% del tempo totale.

Padding

- **Senza** (default): 26.4 secondi.
- **Con**: 35.8 secondi.
- **Causa**: 33% di dati di padding processati.

Analysis - Sequential

Lunghezza Query

Tempo di matching:

- $Q = 32$: 10.4 secondi.
- $Q = 64$ (default): 26.2 secondi.
- $Q = 128$: 56.3 secondi.

Analysis - Parallel

Tempo totale

I vantaggi si ottengono aumentando la dimensione della query e dell'input:

- $Q = 64$ (default): 14.114 secondi. Speed-up di 2.75x rispetto al sequenziale.
- $Q = 128$: 14.931 secondi. Speed-up di 4,65x rispetto al sequenziale.

Analysis - Parallel

Lunghezza query

Tempo di matching:

- $Q = 64$ (default): 0.053448 secondi.
- $Q = 512$: 0.328047 secondi.
- $Q = 2048$: 1.215921 secondi.
- $Q = 4096$: Troppo grande per la constant memory.

Analysis - Parallel

Block size

Tempo di matching:

- $Q = 8$: 0.190880 secondi. Abbiamo una perdita di occupazione hw: blocchi da 8 threads ma warp da 32 threads.
- $Q = 32$: 0.061503 secondi.
- $Q = 64$: 0.056112 secondi.
- $Q = 256$ (default): 0.055100 secondi.
- $Q = 1024$ (max): 0.067674 secondi. Costo della sincronizzazione per il calcolo di i^* del blocco.

Time Series Pattern Recognition

Edoardo Sarri

Grazie