RdS
RICERCA DI SISTEMA

ENEA
AGENZIA NAZIONALE PER LE NUOVE TECNOLOGIE, L'ENERGIA E LO SVILUPPO ECONOMICO SOSTENIBILE

MINISTERO DELL'AMBIENTE E DELLA SICUREZZA ENERGETICA

Ricerca di Sistema elettrico

Test dell'infrastruttura di calcolo HPC a basso consumo per il controllo informatico smart di reti cyber-resilienti

Michele Casà, Paolo Palazzari, Luigi Acampora

ENEA

RdS_PTR 22-24_PR 2.1_LA3.8_142

Test dell'infrastruttura di calcolo HPC a basso consumo per il controllo informatico smart di reti cyber-resilienti

M. Casà (ENEA), P. Palazzari (ENEA), L. Acampora (ENEA)

Dicembre 2024

Report Ricerca di Sistema Elettrico

Accordo di Programma Ministero dell'Ambiente e della Sicurezza Energetica -ENEA Piano Triennale di Realizzazione 2022-2024

Obiettivo 2: Digitalizzazione ed evoluzione delle reti

Progetto 2.1: Cybersecurity dei Sistemi Energetici

Linea di attività: 3.8

Responsabile del Progetto: Maria Valenti (ENEA)

Responsabile Linea di Attività: Luigi Acampora (ENEA)

Mese inizio previsto: luglio 2023

Mese inizio effettivo: luglio 2023

Mese fine previsto: dicembre 2024

Mese fine effettivo: dicembre 2024

Il presente documento descrive le attività di ricerca svolte all'interno dell'Accordo di collaborazione: Si ringrazia per la collaborazione alle attività svolte

Indice

1 Risultati attesi 4
2 Risultati ottenuti 4
2.1 Avanzamento della ricerca rispetto allo stato dell'arte internazionale con riferimento ai risultati ottenuti 5
3 Prodotti attesi 5
4 Prodotti ottenuti 5
5 Analisi degli scostamenti su attività e risultati 6
5.1 6
6 Sintesi delle attività svolte 6
7 Dettaglio delle attività svolte 6
8 Contributo delle eventuali consulenze alle attività sopra descritte 9
9 Pubblicazioni scientifiche 10
10 Eventi di disseminazione 10
11 Descrizione dei risultati ottenuti 10
11.1 Repository del traffico dati nei centri ENEA 11
11.2 Caratterizzazione del server di calcolo 12
11.2.1 SuperMicro server 12
11.2.2 Scheda FPGA U280 16
12 Appendice A 17
13 Appendice B 19
14 Appendice C 24

Indice delle figure

Figura 1: schema dell'infrastruttura di calcolo HPC a basso consumo per il controllo informatico smart di reti cyber-resilienti 7

Figura 2: Tempo di trasferimento a) e banda effettiva b) in funzione della quantità di dati spostata 12

Figura 3: Caratterizzazione L1 13

Figura 4: Caratterizzazione L2 14

Figura 5: Caratterizzazione L3 15

Figura 6: Caratterizzazione DDR 15

1 Risultati attesi

Il dominio della cybersecurity, ed in particolare della sicurezza della rete di gestione e controllo dell'infrastruttura elettrica, presenta un'ampia serie di casi d'uso che hanno spesso come fattore comune la rilevazione di anomalie nel flusso dei dati che vengono generati e successivamente utilizzati per analisi e conservazione delle informazioni. Per questo motivo, basandosi sulla infrastruttura realizzata nelle LA3.4, si propone di implementare e realizzare un test-case relativo alla rilevazione di intrusioni all'interno dei sistemi informatici tramite l'impiego della piattaforma di stream analytics della LA3.7. La piattaforma acquisirà i dati del sistema in esame, li depositerà nel sistema di storage (vedi LA3.9) e, sulla base di algoritmi ML opportunamente addestrati, consentirà di rilevare problemi in quasi real-time. Per garantire la sicurezza dei dati immagazzinati all'interno dell'infrastruttura, si integreranno, inoltre, le tecniche di criptazione sviluppate nella linea di attività LA1.11, implementate sulle risorse FPGA dell'architettura. Grazie alla implementazione che sfrutta il parallelismo (sia spaziale, sia temporale) e alle elevatissime bande di memoria che caratterizzano i dispositivi FPGA, le procedure di criptazione non ridurranno il throughput del processamento.

2 Risultati ottenuti

- Caratterizzazione delle bande di comunicazione e delle relative latenze per il server di calcolo, incluse:
- Accessi alla memoria DDR e alle cache (L1, L2, L3).
- Accesso ai banchi di memoria delle schede acceleratrici (FPGA) sia da parte dell'host che della FPGA stessa.
- Generazione di archivi di traffico statisticamente significativi, opportunamente pseudo-anonimizzati, per consentire la condivisione con collaboratori esterni, tra cui: Università di Roma Tre, Sapienza Università di Roma e Università di Bari.
- Test del cavo crittato realizzato nella LA1.11 dalla Sapienza Università di Roma, mediante comunicazioni tra il server di calcolo SuperMicro, situato presso il centro ENEA di Casaccia, e un nodo di calcolo nella rete di Portici. Le comunicazioni sono state cifrate utilizzando l'algoritmo QP-DYN, implementato su FPGA.
- Esecuzione di algoritmi di Machine Learning (in collaborazione con l'Università di Bari e l'Università di Roma Tre) sfruttando i dataset generati attraverso l'infrastruttura sviluppata nel progetto e le risorse di calcolo della stessa infrastruttura.
- Realizzazione dell'analisi in tempo reale del traffico dati, come descritto nel deliverable *"Realizzazione e implementazione su un caso d'uso di una piattaforma di stream analytics e implementazione delle logiche di training e inference dei modelli di ML e Artificial Intelligence su acceleratori di varia natura (FPGA/GPU)"* della LA3.7.

È importante sottolineare come un'infrastruttura di calcolo HPC a basso consumo, progettata per il controllo informatico avanzato di reti cyber-resilienti, rappresenti un asset strategico per il sistema elettrico nazionale. Grazie alle sue funzionalità di training di algoritmi di Machine Learning e al loro impiego nella fase di inferenza in tempo reale, unitamente alle capacità avanzate di crittografia, essa consente di implementare politiche di sicurezza indispensabili per la gestione delle smart grid energetiche.

2.1 Avanzamento della ricerca rispetto allo stato dell'arte internazionale con riferimento ai risultati ottenuti

È stata progettata e implementata un'architettura HW/SW che integra componenti allo stato dell'arte, tra cui server di calcolo e storage, firewall di ultima generazione, data logger e schede acceleratrici basate su FPGA per la cifratura. L'architettura include inoltre algoritmi di data analytics basati su tecniche di Machine Learning, generatori di traffico malevolo e procedure di pseudo-anonimizzazione che consentono l'utilizzo di dati di traffico reali, anziché sintetici. Questi elementi, combinati, rappresentano un significativo avanzamento rispetto allo stato dell'arte.

Un aspetto particolarmente rilevante è la creazione di una base di dati che raccoglie lo storico del traffico dati generato presso i centri ENEA. Questo archivio, opportunamente pseudo-anonimizzato attraverso una procedura documentata e conforme alla normativa sulla privacy, è condivisibile con la comunità scientifica interessata allo studio della sicurezza nelle reti dati, in particolare quelle elettriche.

Alla data di redazione del presente rapporto, tale archivio rappresenta il primo dataset di traffico dati, pseudo-anonimizzato e basato su dati reali, sviluppato per l'addestramento di algoritmi di data analytics, caratterizzato inoltre da contenuti statisticamente significativi.

3 Prodotti attesi

Prodotto: Rapporto tecnico RT_LA3.8_ENEA: "Test dell'infrastruttura di calcolo HPC a basso consumo per il controllo informatico smart di reti cyber-resilienti"

4 Prodotti ottenuti

Oltre ai risultati descritti nel presente rapporto tecnico, che documentano gli obiettivi raggiunti nell'ambito della linea di attività, è stato conseguito un ulteriore importante risultato, inizialmente non pianificato. Questo riguarda la creazione e la possibilità di mettere a disposizione della comunità scientifica un archivio contenente dati di traffico, sia benevolo che malevolo, rilevati presso i centri ENEA.

L'archivio, aggiornato continuamente con nuovi dati, è stato realizzato utilizzando le procedure di pseudo-anonimizzazione sviluppate nel corso del progetto. Tali procedure risultano fondamentali per garantire la conformità alla normativa sulla privacy e per consentire la condivisione sicura dei dati con la comunità scientifica.

5 Analisi degli scostamenti su attività e risultati

5.1

Nei criteri per valutare il conseguimento degli obbiettivi, si è stabilito che il Rapporto Tecnico dovrà riportare i dettagli dei test condotti per verificare la funzionalità di tutti i componenti dell'infrastruttura digitale realizzata.

Esso dovrà, in particolare, fornire dettagli inerenti a:

- unit test per le procedure di comunicazione;
- test dei singoli moduli HW/SW;
- test di funzionalità delle varie interdipendenze e dell'infrastruttura nel suo complesso);
- risultati delle misure effettuate.

Durante la fase implementativa del progetto, tuttavia, si è scelto di privilegiare l'uso estensivo di test funzionali complessivi dell'architettura, adottando un modello di tipo black-box. In questo approccio, il corretto funzionamento dell'infrastruttura HW/SW è stato verificato attraverso la validità dei risultati prodotti, piuttosto che mediante l'analisi interna dei singoli componenti e delle loro interdipendenze.

Non sono stati implementati unit test, poiché molte delle funzionalità sono integrate nei software utilizzati e, di conseguenza, non si prestano all'implementazione di unit test.

6 Sintesi delle attività svolte

Una volta realizzata l'infrastruttura HW/SW di calcolo e comunicazione (LA3.4), sono state implementate procedure per misurare le bande di comunicazione e le latenze nel server di calcolo. Le funzionalità definite nella LA3.4 sono state utilizzate per intercettare e memorizzare, in forma pseudo-anonimizzata, i dati relativi al traffico nella rete dei centri ENEA, creando una base di dati in continua crescita. Grazie agli strumenti di generazione di traffico malevolo, sono stati simulati attacchi (es. Slowloris Denial of Service, attacchi forza bruta) per addestrare algoritmi di Machine Learning capaci di identificare anomalie operative. I dati pseudo-anonimizzati sono stati usati per addestrare modelli di ML successivamente implementati per analisi in tempo reale del traffico di rete. Infine, è stata verificata la funzionalità del cavo crittato per comunicazioni sicure punto-punto, cifrando il payload dei dati provenienti dai sensori.

7 Dettaglio delle attività svolte

- Una volta realizzata l'infrastruttura HW/SW di calcolo e comunicazione (LA 3.4), la cui struttura è richiamata nella seguente figura 1, sono state implementate procedure volte a caratterizzare le prestazioni ottenibili dal server di calcolo. A tal fine, ci si è concentrati sulla velocità di trasferimento dei dati all'interno del sistema, misurando la

banda di comunicazione ([B/s]) e le **latenze** ([s]) nei diversi percorsi di accesso ai dati. In particolare, sono state analizzate le seguenti configurazioni dell'infrastruttura di calcolo HPC a basso consumo per il controllo informatico smart di reti cyber-resilienti del processore ai dati:

- **Trasferimento tra processore e i vari livelli della gerarchia di memoria**: lettura/scrittura verso dati residenti in L1, L2, L3 o nella memoria DDR del nodo host. Per questa misura è stato sviluppato del codice ad hoc.
- **Trasferimento tra memoria host e scheda FPGA**: lettura/scrittura dalla memoria DDR del processore verso i dati situati nella memoria (HBM o DDR) delle schede acceleratrici FPGA. Per i valori di banda si è utilizzata la funzionalità "xbutil examine" fornita da Amd; per la misura delle latenze si è sviluppato un codice dedicato
- **Accesso dalla FPGA ai suoi banchi di memoria (DDR/HBM)**: lettura/scrittura verso i banchi di memoria HBM o DDR presenti sulla scheda FPGA. Per i valori di banda si è utilizzata la funzionalità "xbutil examine" fornita da Amd; per la misura delle latenze si è sviluppato un codice dedicato, implementando opportuni kernel su FPGA

L'attenzione è stata rivolta al movimento dei dati poiché, nei sistemi di calcolo, è spesso il **tempo di accesso ai dati** (il cosiddetto "memory wall") a rappresentare il principale limite per le prestazioni complessive.

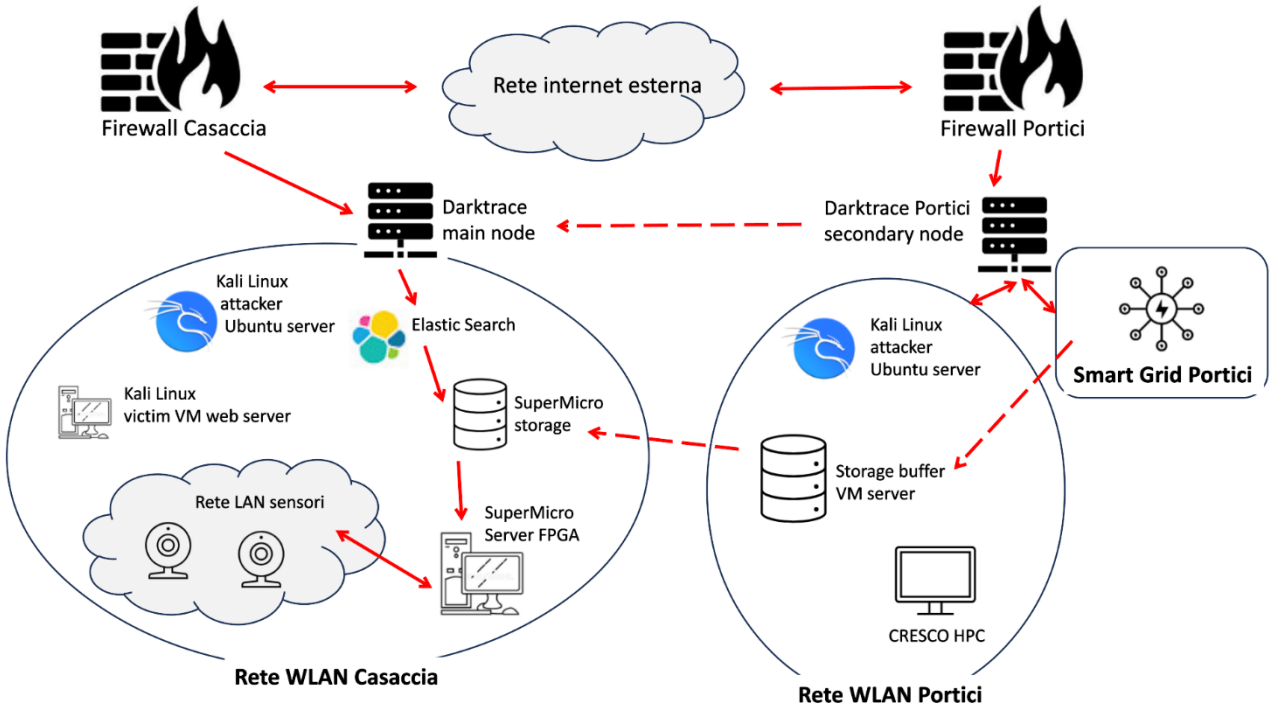

Figura 1: schema dell'infrastruttura di calcolo HPC a basso consumo per il controllo informatico smart di reti cyber-resilienti

La seconda attività svolta in questa LA è consistita nell'utilizzo delle tecniche e funzionalità sviluppate nella LA3.4 (pseudo-anonimizzazione, storage dei dati pseudo-anonimizzati, generazione di traffico malevolo) per creare un archivio statisticamente significativo contenente dati relativi al traffico di rete. La creazione di tale archivio pseudo-anonimizzato è di fondamentale importanza, poiché, grazie alla sua conformità ai criteri imposti dalla

normativa sulla privacy, esso può essere condiviso con altre istituzioni di ricerca impegnate nello sviluppo di metodi per la cyber-sicurezza.

Durante il progetto è emerso quanto sia cruciale disporre di dati di traffico realistici per eseguire il training degli algoritmi di Machine Learning destinati a riconoscere potenziali minacce alla sicurezza della rete. Le istituzioni, in particolare quelle universitarie, che operano in questo settore non hanno generalmente accesso a quantità di dati statisticamente significative e spesso devono ricorrere a traffico sintetico per le procedure di Machine Learning.

D'altra parte, le istituzioni che possiedono dati di traffico reali tendono a non condividerli per vari motivi:

- La distribuzione di tali informazioni non rientra nella loro mission principale.
- Le procedure di pseudo-anonimizzazione richiedono una fase di preparazione formale complessa e laboriosa.
- Tali dati possono avere un valore economico o strategico che ne impedisce la libera condivisione.

Alla luce di quanto sopra, si ritiene di particolare importanza aver realizzato una procedura che:

- Preleva i dati di traffico forniti da DarkTrace tramite Elastic Search.
- Pseudo-anonimizza tali dati, in conformità con la normativa sulla privacy.
- Memorizza quotidianamente i dati relativi all'intero traffico di rete nei centri ENEA di Portici e Casaccia. Questi dati riguardano il traffico generato da ricercatori, borsisti, aziende ospitate nei centri, sensoristica, e riflettono i tipici andamenti temporali del traffico dati nei centri di calcolo (ad esempio, riduzione del volume dei dati scambiati durante i fine settimana, i mesi estivi e i periodi festivi come Natale e Pasqua).
- Offre la possibilità di generare traffico malevolo (ad esempio, attacchi DoS Slowloris o brute force) per consentire la calibrazione e il test degli algoritmi di Machine Learning.

La memorizzazione avviene in un buffer circolare che conserva i dati relativi a una finestra temporale mobile di 30 giorni. Quando i dati escono dalla finestra di osservazione, vengono trasferiti nell'archivio permanente mantenuto sullo storage server.

Seguendo la pipeline di processamento prevista dall'infrastruttura HW/SW realizzata, i dati pseudo-anonimizzati e storicizzati sono stati utilizzati per il training degli algoritmi di Machine Learning (ML) da parte delle università consorziate (Roma Tre, Bari). Queste hanno sfruttato sia le proprie risorse di calcolo sia il server di calcolo installato e configurato nell'ambito del progetto. Gli algoritmi utilizzati sono:

- Autoencoder
- Clustering
- k-means
- Gerarchico
- k-modes

Per maggiori dettagli sull'implementazione degli algoritmi di training e sui risultati ottenuti, si rimanda ai deliverable delle attività LA 3.9 per quanto riguarda gli algoritmi di autoencoding ("Progettazione e implementazione di un sistema di raccolta, conservazione e analisi di flussi di dati per la cybersicurezza delle reti di sensori" e "Sistema di raccolta, conservazione e analisi di flussi di dati per la cybersicurezza delle reti di sensori") e ai deliverable dell'attività LA3.10 per quanto riguarda gli algoritmi di clustering ("Sviluppo e implementazione di metodi di Machine Learning supervisionati e non supervisionati per lo studio delle anomalie" e "Schema degli algoritmi di clustering da utilizzare per la protezione da cyber minacce").

Dopo aver completato il training degli algoritmi di Machine Learning (ML), questi sono stati testati per analizzare in tempo reale il traffico di rete. L'infrastruttura per la cyber-sicurezza si è dimostrata capace di operare correttamente in questa modalità, come descritto nel report della LA 3.7 ("Realizzazione e implementazione su un caso d'uso di una piattaforma di stream analytics e implementazione delle logiche di training e inference dei modelli di ML e Artificial Intelligence su acceleratori di varia natura (FPGA/GPU)").

Infine, è stata verificata la funzionalità del cavo crittato per comunicazioni sicure punto-punto, con la seguente procedura:

- Il server di calcolo interroga i sensori tramite una richiesta http, ricevendo in risposta una stringa, in formato json, contenente le misure dei vari sensori e lo stato degli attuatori collegati all'indirizzo IP (appartenente alla VPN dei sensori) cui si sta accedendo.
- Il server di calcolo estrae l'informazione e la invia alla scheda FPGA che la restituisce cifrata con l'algoritmo QP-DYN sviluppato nella LA 1.11.
- Il server, tramite un socket TCP, invia il payload cifrato ad un secondo server situato all'interno della rete di Portici; tale server è un nodo appartenente alla infrastruttura di calcolo Cresco HPC ed è dotato di scheda FPGA U280.
- Il server di Portici riceve tramite socket il dato cifrato, lo invia alla sua FPGA che lo decifra usando l'IP sviluppato in LA 1.11 e lo restituisce in chiaro al server di calcolo.

Dettagli sul test di comunicazione tramite il cavo crittato sono riportati nel deliverable della LA1.11 "Implementazione efficiente su logiche programmabili di primitive crittografiche per la sintesi di funzioni di cifratura a flusso e autenticazione".

8 Contributo delle eventuali consulenze alle attività sopra descritte

Non è stata necessaria la partecipazione di alcuna società di consulenza per lo svolgimento delle attività e per il perseguimento delle finalità di progetto.

9 Pubblicazioni scientifiche

Le pubblicazioni scientifiche sono quelle congiunte con i partner di progetto, relative allo sviluppo e alla valutazione di algoritmi di machine learning e intelligenza artificiale eseguiti sul sistema IT integrato. Nello specifico sono in fase di sottomissione le seguenti pubblicazioni:

- Leveraging Machine Learning (Supervised and Unsupervised) to Enhance Cybersecurity: Mitigating DoS Attacks in Enea Network Infrastructures. Alfonso Monaco e gruppo di lavoro ENEA. In sottomissione a MAKE di MDPI.
- An unsupervised approach to intrusion detection in IoT networks. Riccardo Torlone, Stefano Iannucci e gruppo di lavoro ENEA. In sottomissione a Future Generation Computer Systems.
- Cybersicurezza dei sistemi energetici. Maria Valenti, Giovanna Adinolfi, Massimo Celino, Roberto Ciavarella. Rivista "Energia Ambiente Innovazione" 2025, Editor ENEA.

10 Eventi di disseminazione

La partecipazione all'evento conclusivo di presentazione dei risultati del Progetto Integrato Cybersecurity dei sistemi energetici (Workshop-Progetto-Cybersecurity.pdf), presso l'Auditorium GSE in viale Maresciallo Pilsudski 92, Roma, il 6 Dicembre ha dato l'opportunità di condividere con un'ampia platea di partecipanti interni ed esterni i risultati delle attività di progetto.

11 Descrizione dei risultati ottenuti

In questo paragrafo sono riportati i principali risultati ottenuti nella presente linea di attività, finalizzata al test dell'infrastruttura di calcolo HPC a basso consumo per il controllo informatico smart di reti cyber-resilienti. I risultati conseguiti sono collegati alle attività svolte nelle

- LA 1.11 (Implementazione efficiente su logiche programmabili di primitive crittografiche per la sintesi di funzioni di cifratura a flusso e autenticazione),
- LA 3.7 (Realizzazione e implementazione su un caso d'uso di una piattaforma di stream analytics e implementazione delle logiche di training e inference dei modelli di Machine Learning e Artificial Intelligence su acceleratori di varia natura (FPGA/GPU)),
- LA 3.9 (Progettazione e implementazione di un sistema di raccolta, conservazione e analisi di flussi di dati per la cybersicurezza delle reti di sensori) e
- LA 3.10 (Sviluppo ed implementazione di metodi di Machine Learning supervisionati e non supervisionati per lo studio delle anomalie).

Risultati ottenuti:

- Repository, continuamente aggiornato sino alla giornata precedente, con lo storico del traffico nella rete dati dei centri ENEA di Casaccia, Portici e Brindisi. I dati di traffico sono privati del payload e sono pseudo-anonimizzati per essere conformi alle linee guida dettate dalla normativa sulla privacy;
- Caratterizzazione della velocità di trasferimento interna alle varie sezioni del server di calcolo. Sono misurate banda e latenza nei vari percorsi dati interni al server di calcolo;
- Test, con esito positivo, delle procedure di Machine Learning. L'infrastruttura è stata impiegata con successo per fare il training, su dati prelevati dal repository del traffico, di algoritmi di ML;
- Test, con esito positivo, delle procedure di inference in tempo reale. L'infrastruttura è stata impiegata con successo per analizzare in tempo reale il traffico prelevato ds ElasticSearch e processato tramite gli algoritmi di ML addestrati nella fase di training;
- Test, con esito positivo, dell'impiego del "cavo crittato", inviando ad un server nella rete di Portici dei dati cifrati prelevati dai sensori presenti in Casaccia; i dati ricevuti dal server di Portici sono stati correttamente decifrati dall'IP di cifratura QP-DYN.

Di seguito si riportano i dettagli relativi alla creazione del repository relativo al traffico dati nei centri di Portici, Brindisi e Casaccia e alle misure finalizzate a caratterizzare le prestazioni del server di calcolo. Per il dettaglio dei test:

- degli algoritmi di training (autoencoding) si rimanda al deliverable della LA3.9
- degli algoritmi di training (clustering) si rimanda al deliverable della LA 3.10
- dell'inference real-time si rimanda al deliverable della LA 3.7
- del cavo crittato si rimanda al deliverable della LA 3.11

11.1 Repository del traffico dati nei centri ENEA

La possibilità di poter memorizzare, in forma pseudo-anonimizzata, i dati relativi al traffico presente nella rete dei centri ENEA, è uno dei risultati fondamentali del progetto in quanto costituisce una tecnologia abilitante per le seguenti attività di Machine Learning e di inferenza per la protezione delle smart grid energetiche.

I dati di traffico raccolti da Darktrace vengono trasferiti al server storage Supermicro (descritto nel deliverable della Linea di Attività 3.4) su cui è installato l'ambiente ElasticSearch, per l'archiviazione e le successive analisi. Per garantire la tutela della privacy, i dati vengono pseudo-anonimizzati tramite una ingest pipeline configurata in ElasticSearch. Questa pipeline utilizza uno script che applica un algoritmo di hash (SHA-256) agli indirizzi IP sorgente e destinazione, trasformandoli in valori crittografati non riconoscibili. Questo processo consente di mantenere l'integrità e l'utilità dei dati per analisi e monitoraggio, rispettando al contempo i requisiti di riservatezza. A valle della procedura di pseudo-anonimizzazione, i dati vengono indicizzati quotidianamente, permettendo un accesso rapido e organizzato alle informazioni.

Il sistema è configurato per conservare i dati per un periodo massimo di due anni, in conformità con le disposizioni del Data Protection Impact Assessment (DPIA). Questa soluzione garantisce il rispetto delle normative sulla protezione dei dati, mantenendo elevate prestazioni e sicurezza nell'elaborazione delle informazioni.

Attualmente in una giornata lavorativa tipica viene prodotto un file di circa 32 GB contenente i dati di traffico pseudo-anonimizzati (e privi del payload); nel fine settimana le dimensioni di

tale file si riducono di circa il 50%. Ne consegue che in un mese lavorativo il repository cresce di 32*22+16*8 = 832 GB e, in un anno, abbattendo del 50% il volume di traffico nel mese di agosto, il traffico prodotto richiede 832*11,5 = 9.568 GB ~ 10 TB.

11.2 Caratterizzazione del server di calcolo

11.2.1 SuperMicro server

Il server di calcolo che ospita due schede FPGA Alveo U280 è un sistema biprocessore AMD EPYC 9124, con 256 GB di memoria DDR, 2 TB di spazio disco (SSD) e scheda Ethernet da 10 GB/s. Ogni processore è dotato di 16 core fisici (32 core logici), ha 32 istanze di memoria cache privata di primo livello (L1d: 32KB dedicata ai dati e L1i:32KB dedicata alle istruzioni), 32 istanze di memoria cache di secondo livello (L2, ogni istanza privata misura 1 MB) e 8 istanze di memoria cache di terzo livello condivisa (L3: 16 MB è la dimensione di ogni istanza).

Per caratterizzare le prestazioni effettivamente ottenibili dalla gerarchia di memoria del sistema da noi installato, è stato sviluppato il codice, riportato in appendice A, per la determinazione di banda e latenza quando si accede ai vari livelli della memoria.

Nella seguente Figura 2: Tempo di trasferimento a) e banda effettiva b) in funzione della quantità di dati spostata viene riportato, in funzione della dimensione del dato, il tempo necessario a trasferire il dato in memoria (Figura 2.a) e la banda effettiva sostenuta per il trasferimento (Figura 2.b).

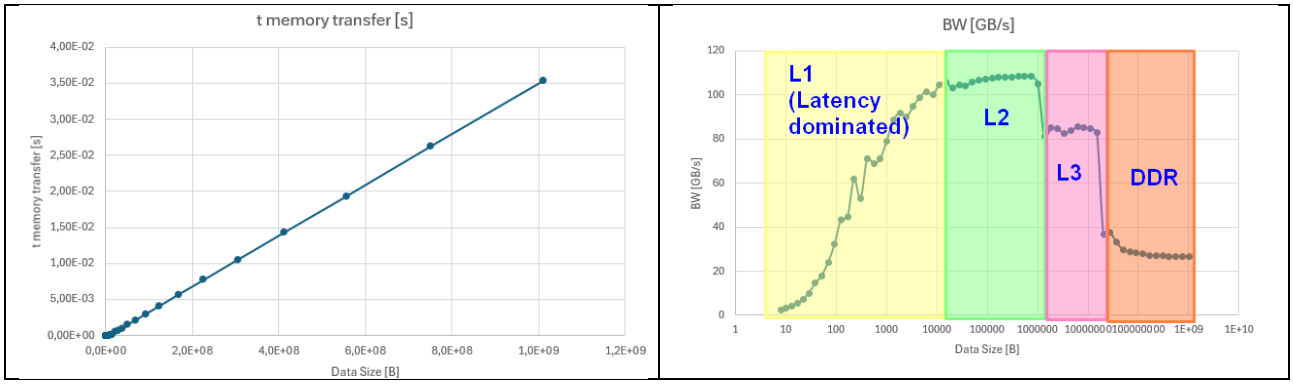

Figura 2: Tempo di trasferimento a) e banda effettiva b) in funzione della quantità di dati spostata

Come si vede, nella zona L1 il tempo di latenza è predominate, mentre per trasferimenti di dimensioni maggiori il ruolo della latenza diviene trascurabile e le prestazioni sono determinate dalla banda di memoria disponibile. Una stima della latenza per l'accesso a L1 può essere derivata dai dati relativi ai tempi di comunicazione per messaggi brevi, ossia la cui lunghezza sia tale da poter essere contenuti in L1. In questo caso possiamo considerare il tempo di trasferimento verso la memoria di una sequenza di x byte come espresso dalla relazione

$$t_{Mem}(x) = t_{Latenza} + x/BW$$

Facendo riferimento alla seguente figura 3, che è l'esplosione del tratto iniziale della figura 2.a, l'equazione della linea di tendenza fornisce la latenza e la banda di accesso alla memoria. Il tempo di latenza per L1 è dato da

$$t_{Latenza}(L1) = 2.62 \text{ ns}$$

e la banda di accesso alla memoria è

$$BW(L1) = 1/(8.72x10^{-12}x1024^3) = 107 \text{ GB/s}$$

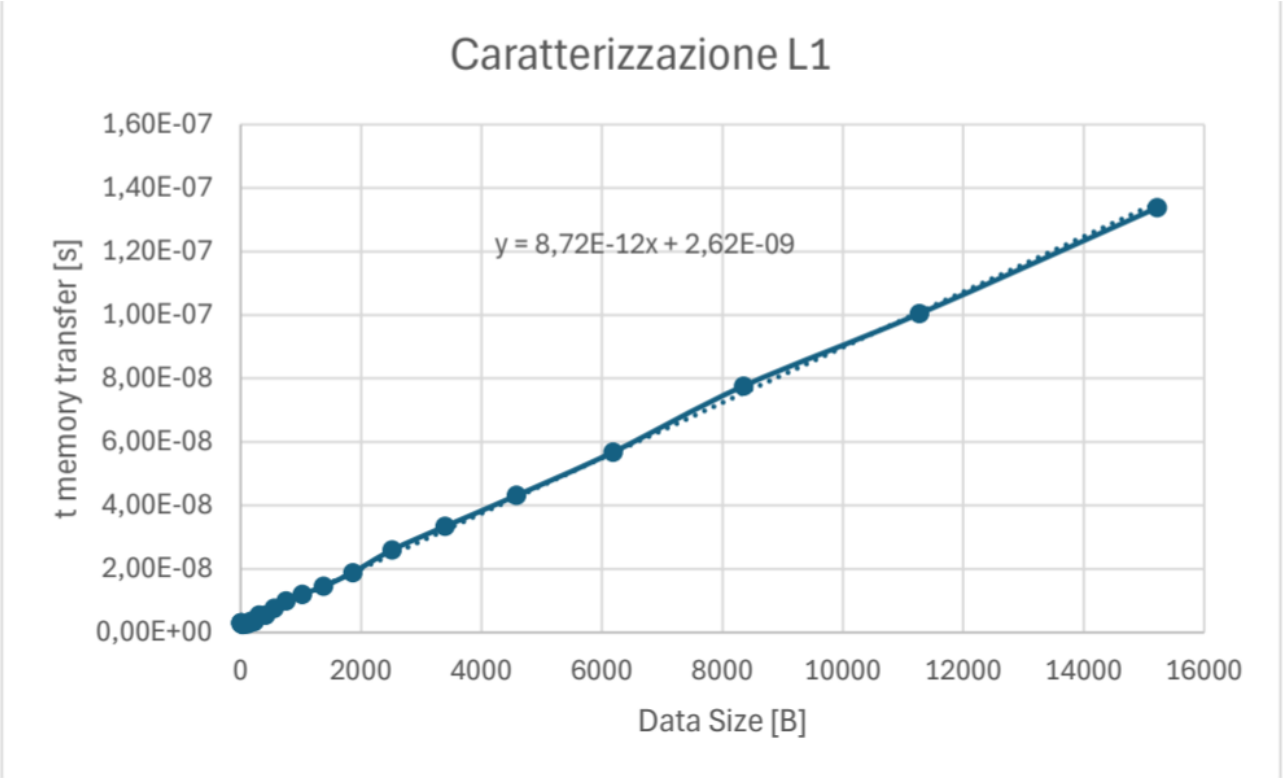

Figura 3: Caratterizzazione L1

In maniera analoga, possiamo estrarre i dati relativi alla dimensione di L2 e ricavare i valori di latenza e di banda.

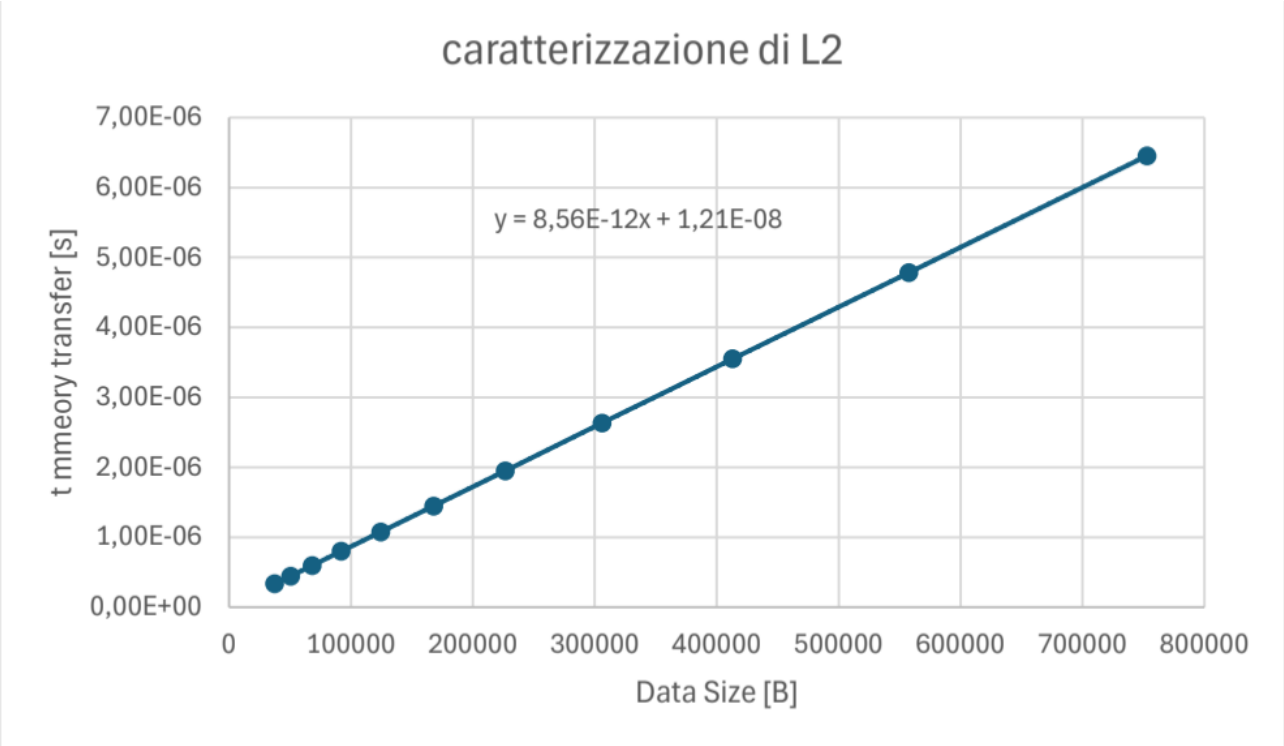

Figura 4: Caratterizzazione L2

l'equazione della linea di tendenza fornisce il tempo di latenza per L2

$$t_{Latenza}(L2) = 12.1 \text{ ns}$$

e la banda di accesso alla memoria è la stessa di L1.
Ripetendo la stessa operazione per L3 e per l'accesso alla DDR, tralasciando i valori di latenza che sono eccessivamente sensibili alla pendenza della retta di regressione, si ottengono i seguenti valori per la banda di accesso, rispettivamente, a L3 e ai banchi di memoria DDR:

BW(L3) = 85,4 GB/s
BW(DDR) = 26,5 GB/s

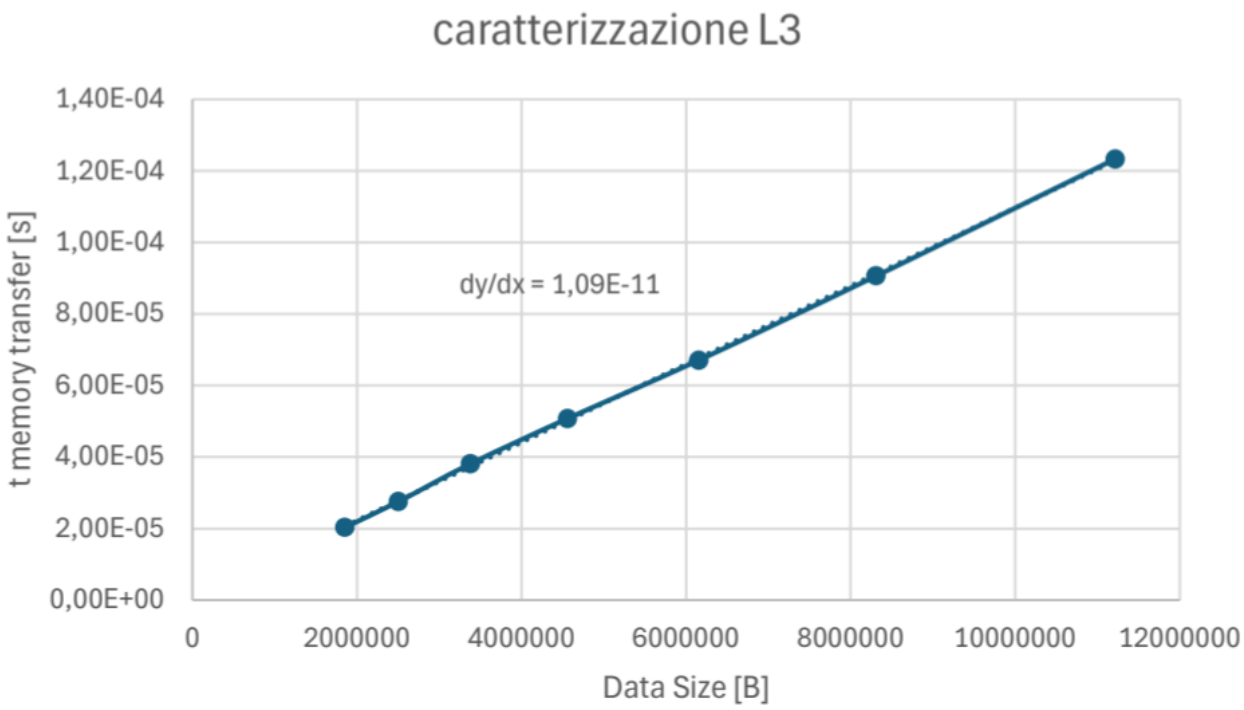

Figura 5: Caratterizzazione L3

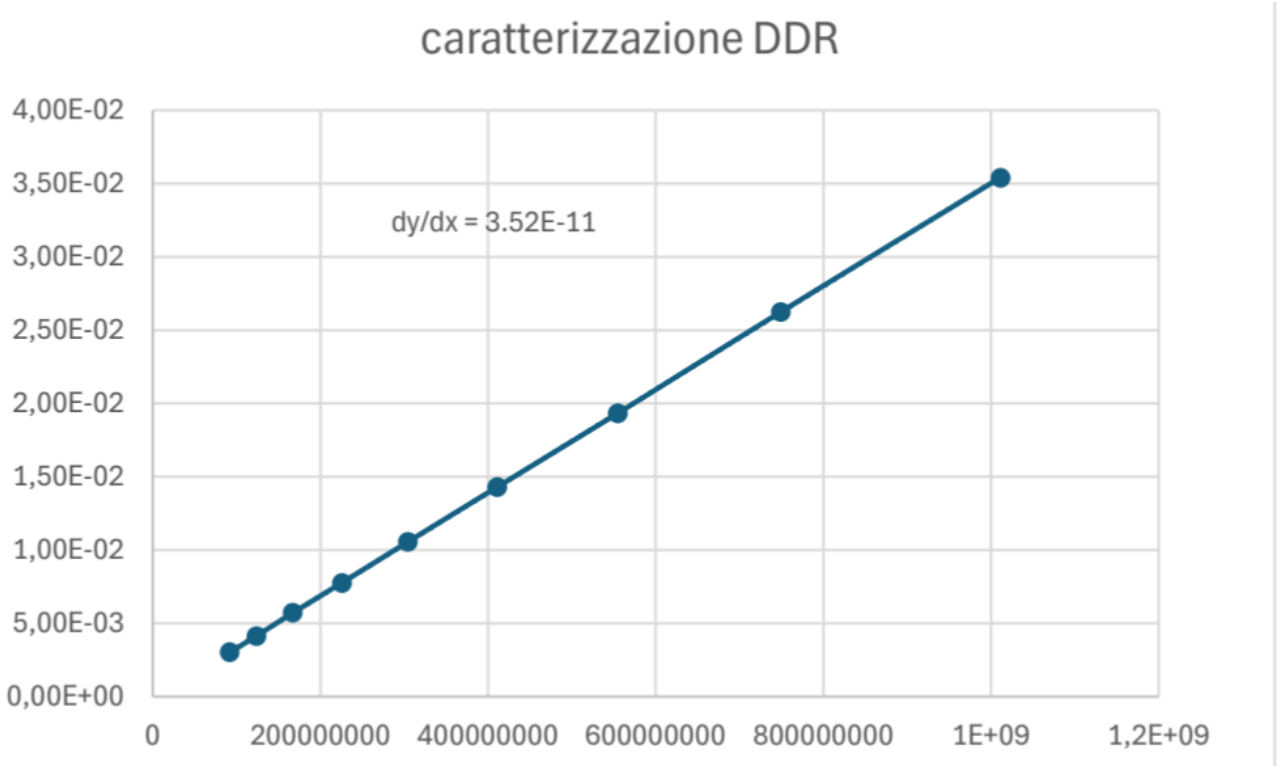

Figura 6: Caratterizzazione DDR

Per determinare le bande di comunicazione, attraverso il PCIe, ottenibili comunicando con la scheda Alveo U280, abbiamo usato il programma di analisi xbutil fornito da AMD.

Eseguendo il comando xbutile examine –device 0000:01:00.1, vengono eseguiti vari test di funzionalità sulla scheda Alveo installata. In particolare, verificando i dati relativi alla comunicazione tramite PCIe tra il nodo host e la prima scheda Alveo, abbiamo rilevato i seguenti valori per la banda di comunicazione:

Buffer size - '16 MB'
Host -> PCIe -> FPGA write bandwidth = 11101.5 MB/s
Host <- PCIe <- FPGA read bandwidth = 11732.4 MB/s

Eseguendo il comando xbutile examine –device 0000:a1:00.1, che afferisce alla seconda scheda Alveo installata nel server, abbiamo ottenuto i seguenti valori

Buffer size - '16 MB'
Host -> PCIe -> FPGA write bandwidth = 11103.9 MB/s
Host <- PCIe <- FPGA read bandwidth = 11747.4 MB/s

Le latenze di lettura e scrittura sono state ricavate con il programma riportato in appendice B

Host -> PCIe -> FPGA write latency= 26.9 us
Host <- PCIe <- FPGA read latency = 25.0 us

11.2.2 Scheda FPGA U280

La caratterizzazione della banda disponibile tra la FPGA ed i banchi di memoria (DDR e HBM) è stata eseguita utilizzando il comando xbutile examine –device 0000:01:00.1, (e xbutile examine –device 0000:a1:00.1) che hanno fornito come misura della banda disponibile verso la DDR una banda complessiva di ~34 GB/s (quindi 17 GB/s per ognuno dei due banchi DDR) e verso un banco di memoria HBM la banda misurata è stata di ~12 GB/s.

mem-bw : Throughput (Type: DDR) (Bank count: 2) : 33931.2MB/s
Throughput (Type: HBM) (Bank count: 1) : 12383.4MB/s

Mem-Latency: Read Latency (Type: DDR) : 289 ns
Write Latency (Type DDR) : 229 ns

Mem-Latency: Read Latency (Type: HBM) : 260 ns
Write Latency (Type HBM) : 110 ns

Per la misura delle latenze dei trasferimenti che coinvolgono il PCIe abbiamo usato il codice riportato in appendice C.

12 Appendice A

Codice per la caratterizzazione dei canali di comunicazione tra il processore e L1, L2, L3, DDR

```
#include <iostream>
#include <sys/time.h>
#include <cstring>
#include <cstdio>
using namespace std;

#define MEM_SIZE (unsigned int)(2U*1024U*1024U*1024U)
#define NB_REPETITIONS (1024*1024*1024)
#define NB_STEPS 65
#define INITIAL_SIZE 4

double computeElapsed(timeval ts, timeval te)
{
     double retVal = 0.0;
     retVal = (te.tv_sec-ts.tv_sec)*1000000.0+(te.tv_usec-ts.tv_usec);
     return retVal;
}

void useDataForBW(unsigned int *data, int nbRepetitions, int size, int
value)
{
unsigned int i, j, index;
    #pragma GCC unroll 32
for (i=0; i<nbRepetitions; i++)
memset((void *)data,value,size);
}

void useDataForLatency(unsigned int *data, int nbRepetitions, int value)
{
int i, index = 1;;
    #pragma GCC unroll 32
for (i=0; i<nbRepetitions; i++)
memset((void *)data,value,1);
}
int main() {
FILE *f;
unsigned int *data = new unsigned int[MEM_SIZE];
int k,l;
int n, vsize;
double elapsed;

    timeval ts,te;

    n = NB_REPETITIONS;
    vsize = INITIAL_SIZE;

    f = fopen("results.txt","wt");

    if (f==NULL)
```

```
    {
     cout << "error opening result.txt file"<<std::endl;
     return -1;
    }

    fprintf(f,"Nb Bytes [B]\tBW [GB/s]\tt_comm\n");
    for (k=0; k<NB_STEPS; k++)
    {
     for (l=0; l<vsize; l++)
          data[l] = l+1;
     gettimeofday(&ts,NULL);
     useDataForBW(data, n, vsize, k);
     gettimeofday(&te,NULL);

     elapsed = computeElapsed(ts, te);
     double dataSize = (double)vsize*(double)n;
     cout << "Sent "<< n <<" times " << vsize << " bytes. Globally, sent
       " << dataSize/(1024.0*1024.0*1024.0) << " GB in " << elapsed/1.e6
       << " sec" << std::endl;
     cout    <<    "BW    L"    <<    k+1    <<    "    =    "    <<
      (dataSize*1.0e6/((1024.0*1024.0*1024.0)*elapsed)) << " [GB/s]" <<
      std::endl;
    fprintf(f,"%lf\t%lf\t%e\n",(double)vsize*(double)sizeof(int),(dataSi
      ze*1.0e6/((1024.0*1024.0*1024.0)*elapsed)),(double)(elapsed/(doubl
      e)n*(double)1.0e-6));

     vsize = (double)vsize*1.35;
     n = ((double)n)/1.35;
    }

     n = 1024*1024*1024;
gettimeofday(&ts,NULL);
useDataForLatency(data, n,1);
gettimeofday(&te,NULL);
elapsed = computeElapsed(ts, te);
double latency = elapsed/(1024.0*1024.0*1024.0);

cout<<"Latency = "<<latency<<" us"<<std::endl;
fprintf(f,"\n\nLatency = %lf us\n",latency);

    if (fclose(f)!=0)
     cout << "error closing file" << std::endl;;
return 0;
}
```

13 Appendice B

Codice dei kernel e dell'host per la misura delle latenze di comunicazione in lettura e scrittura tra la FPGA ed i banchi di memoria DDR/HBM

```
////////////////////////////////////////////////////////////////////////////////////////////////
//                                        CODICE KERNELS                                      //
////////////////////////////////////////////////////////////////////////////////////////////////
#include <stdint.h>
#include <hls_stream.h>


void measureReadLatency(long long int* Mem,
                        long long int *outMem,
                        int N1, int N2)
{
long long int a = 0;
for (int i=0; i<N1; i++)
for (int j=0; j<N2; j++)
a = Mem[a&0x00FFFFFF];
outMem[0] = a;
}


void measureWriteLatency(long long int* Mem,
                         long long int *inMem,
                         int IndexMap[255],
                         int N1,
                         int N2)
{
long long int a;
a = inMem[0];
for (int i=0; i<N1; i++)
for (int j=0; j<N2; j++)
Mem[IndexMap[(i+j)&0xFF]]=a;
}


extern "C" {


void krnl_measureLatency(long long int* readDDR,
                         long long int* writeDDR,
                         int N1,
                         int N2)
{
#pragma HLS INTERFACE m_axi port = readDDR
#pragma HLS INTERFACE m_axi port = writeDDR num_write_outstanding=1
```

```
int IndexMap[255];
for (int i =0; i<255; i++)
IndexMap[i] = i*256;
// Uncomment to measure Read Latency
measureReadLatency(readDDR, writeDDR, N1, N2);
//  Uncomment to measure Write Latency
//    measureWriteLatency(writeDDR, readDDR, IndexMap, N1, N2);


}
}
```

```
////////////////////////////////////////////////////////////////////////////////////////////
//                                    CODICE HOST                                         //
////////////////////////////////////////////////////////////////////////////////////////////
#define OCL_CHECK(error, call)
\
    call;
\
    if (error != CL_SUCCESS) {
\
        printf("%s:%d Error calling " #call ", error code is: %d\n",
__FILE__, __LINE__, error); \
        exit(EXIT_FAILURE);
\
    }


#include "measureLatency_host.h"
#include <fstream>
#include <iostream>
#include <stdlib.h>
#include <sys/time.h>
#include <unistd.h>


static const int DATA_SIZE = (1);


static const std::string error_message =
    "Error: Result mismatch:\n"
    "i = %d CPU result = %d Device result = %d\n";


double getElapsed(timeval ts, timeval te)
{
double elapsed = (te.tv_sec-ts.tv_sec)*1.0e6+(te.tv_usec-ts.tv_usec);
return elapsed;
}
```

```
int main(int argc, char* argv[]) {
    if (argc != 4) {
        std::cout << "Usage: " << argv[0] << " <xclbin> <N1> <N2>" <<
std::endl;
        return EXIT_FAILURE;    }
    struct timeval ts, te;
    int N1 = atoi(argv[2]);
    int N2 = atoi(argv[3]);


    std::string xclbinFilename = argv[1];


    // Compute the size of array in bytes
    size_t size_in_bytes = DATA_SIZE * sizeof(long long int);


    std::vector<cl::Device> devices;
    cl_int err;
    cl::Context context;
    cl::CommandQueue q;
    cl::Kernel krnl_measureLatency;
    cl::Program program;
    std::vector<cl::Platform> platforms;
    bool found_device = false;


    // traversing all Platforms To find Xilinx Platform and targeted
    // Device in Xilinx Platform
    cl::Platform::get(&platforms);
    for (size_t i = 0; (i < platforms.size()) & (found_device == false);
i++) {
        cl::Platform platform = platforms[i];
        std::string platformName = platform.getInfo<CL_PLATFORM_NAME>();
        if (platformName == "Xilinx") {
            devices.clear();
            platform.getDevices(CL_DEVICE_TYPE_ACCELERATOR, &devices);
            if (devices.size()) {
                found_device = true;
                break;
            }
        }
    }
    if (found_device == false) {
        std::cout << "Error: Unable to find Target Device " << std::endl;
        return EXIT_FAILURE;
    }


    std::cout << "INFO: Reading " << xclbinFilename << std::endl;
    FILE* fp;
    if ((fp = fopen(xclbinFilename.c_str(), "r")) == nullptr) {
```

```
        printf("ERROR: %s xclbin not available please build\n",
xclbinFilename.c_str());
        exit(EXIT_FAILURE);
    }
    // Load xclbin
    std::cout << "Loading: '" << xclbinFilename << "'\n";
    std::ifstream bin_file(xclbinFilename, std::ifstream::binary);
    bin_file.seekg(0, bin_file.end);
    unsigned nb = bin_file.tellg();
    bin_file.seekg(0, bin_file.beg);
    char* buf = new char[nb];
    bin_file.read(buf, nb);


    // Creating Program from Binary File
    cl::Program::Binaries bins;
    bins.push_back({buf, nb});
    bool valid_device = false;
    printf(" Nb FPGA devices : %d\n", devices.size());
    for (unsigned int i = 0; i < devices.size(); i++) {
        auto device = devices[i];
        // Creating Context and Command Queue for selected Device
        OCL_CHECK(err, context = cl::Context(device, nullptr, nullptr,
nullptr, &err));
        OCL_CHECK(err, q = cl::CommandQueue(context, device,
CL_QUEUE_PROFILING_ENABLE, &err));
        std::cout << "Trying to program device[" << i << "]: " <<
device.getInfo<CL_DEVICE_NAME>() << std::endl;
        cl::Program program(context, {device}, bins, nullptr, &err);
        if (err != CL_SUCCESS) {
            std::cout << "Failed to program device[" << i << "] with
xclbin file!\n";
        } else {
            std::cout << "Device[" << i << "]: program successful!\n";
            OCL_CHECK(err, krnl_measureLatency = cl::Kernel(program,
"krnl_measureLatency", &err));
            valid_device = true;
            break; // we break because we found a valid device
        }
    }
    if (!valid_device) {
        std::cout << "Failed to program any device found, exit!\n";
        exit(EXIT_FAILURE);
    }


    std::vector<long long int, aligned_allocator<long long int> >
ptr_DDR(1);
    std::vector<long long int, aligned_allocator<long long int> >
ptr_result_DDR(1);


    // These commands will allocate memory on the Device. The cl::Buffer
objects can
```

```
    // be used to reference the memory locations on the device.
    OCL_CHECK(err, cl::Buffer buffer_DDR(context, CL_MEM_USE_HOST_PTR |
CL_MEM_READ_ONLY, size_in_bytes, ptr_DDR.data(), &err));
    OCL_CHECK(err, cl::Buffer buffer_result_DDR(context,
CL_MEM_USE_HOST_PTR | CL_MEM_WRITE_ONLY, size_in_bytes,
ptr_result_DDR.data(), &err));


    std::cout << "Allocated read and write buffers on the board. Press
any key to continue ...";
    getchar();


    OCL_CHECK(err, err = krnl_measureLatency.setArg(0, buffer_DDR));
    OCL_CHECK(err, err = krnl_measureLatency.setArg(1,
buffer_result_DDR));
    OCL_CHECK(err, err = krnl_measureLatency.setArg(2, N1));
    OCL_CHECK(err, err = krnl_measureLatency.setArg(3, N2));


    std::cout << "done. Press any key to continue and start the kernel"
<< std::endl;
    getchar();


    gettimeofday(&ts, NULL);
    OCL_CHECK(err, err = q.enqueueTask(krnl_measureLatency));
    q.finish();
    gettimeofday(&te,NULL);
    double DDRReadLatency = getElapsed(ts, te);
    printf("Overall measure time : %lf us\n",DDRReadLatency);
    printf("DDR read latency = %lf
us\n",DDRReadLatency/((double)N1*(double)N2));

    std::cout << "TEST FINISHED" << std::endl;
    return (EXIT_SUCCESS);
}
```

14 Appendice C

Codice per la misura della latenza delle comunicazioni attraverso il PCIe

```
#define OCL_CHECK(error, call)
\
    call;
\
    if (error != CL_SUCCESS) {
\
        printf("%s:%d Error calling " #call ", error code is: %d\n",
__FILE__, __LINE__, error); \
        exit(EXIT_FAILURE);
\
    }

#include "measureLatency_host.h"
#include <fstream>
#include <iostream>
#include <stdlib.h>
#include <sys/time.h>
#include <unistd.h>

static const int DATA_SIZE = (1);

static const std::string error_message =
    "Error: Result mismatch:\n"
    "i = %d CPU result = %d Device result = %d\n";

double getElapsed(timeval ts, timeval te)
{
double elapsed = (te.tv_sec-ts.tv_sec)*1.0e6+(te.tv_usec-ts.tv_usec);
return elapsed;
}

int main(int argc, char* argv[]) {
    if (argc != 4) {
        std::cout << "Usage: " << argv[0] << " <xclbin> <N1> <N2>" <<
std::endl;
        return EXIT_FAILURE;    }
    struct timeval ts, te;
    int N1 = atoi(argv[2]);
    int N2 = atoi(argv[3]);

    std::string xclbinFilename = argv[1];

    // Compute the size of array in bytes
    size_t size_in_bytes = DATA_SIZE * sizeof(long long int);

    std::vector<cl::Device> devices;
    cl_int err;
```

```
    cl::Context context;
    cl::CommandQueue q;
    cl::Kernel krnl_measureLatency;
    cl::Program program;
    std::vector<cl::Platform> platforms;
    bool found_device = false;

    // traversing all Platforms To find Xilinx Platform and targeted
    // Device in Xilinx Platform
    cl::Platform::get(&platforms);
    for (size_t i = 0; (i < platforms.size()) & (found_device == false);
i++) {
        cl::Platform platform = platforms[i];
        std::string platformName = platform.getInfo<CL_PLATFORM_NAME>();
        if (platformName == "Xilinx") {
            devices.clear();
            platform.getDevices(CL_DEVICE_TYPE_ACCELERATOR, &devices);
            if (devices.size()) {
                found_device = true;
                break;
            }
        }
    }
    if (found_device == false) {
        std::cout << "Error: Unable to find Target Device " << std::endl;
        return EXIT_FAILURE;
    }

    std::cout << "INFO: Reading " << xclbinFilename << std::endl;
    FILE* fp;
    if ((fp = fopen(xclbinFilename.c_str(), "r")) == nullptr) {
        printf("ERROR: %s xclbin not available please build\n",
xclbinFilename.c_str());
        exit(EXIT_FAILURE);
    }
    // Load xclbin
    std::cout << "Loading: '" << xclbinFilename << "'\n";
    std::ifstream bin_file(xclbinFilename, std::ifstream::binary);
    bin_file.seekg(0, bin_file.end);
    unsigned nb = bin_file.tellg();
    bin_file.seekg(0, bin_file.beg);
    char* buf = new char[nb];
    bin_file.read(buf, nb);

    // Creating Program from Binary File
    cl::Program::Binaries bins;
    bins.push_back({buf, nb});
    bool valid_device = false;
    printf(" Nb FPGA devices : %d\n", devices.size());
    for (unsigned int i = 0; i < devices.size(); i++) {
        auto device = devices[i];
        // Creating Context and Command Queue for selected Device
        OCL_CHECK(err, context = cl::Context(device, nullptr, nullptr,
nullptr, &err));
```

```
        OCL_CHECK(err, q = cl::CommandQueue(context, device,
CL_QUEUE_PROFILING_ENABLE, &err));
        std::cout << "Trying to program device[" << i << "]: " <<
device.getInfo<CL_DEVICE_NAME>() << std::endl;
        cl::Program program(context, {device}, bins, nullptr, &err);
        if (err != CL_SUCCESS) {
            std::cout << "Failed to program device[" << i << "] with
xclbin file!\n";
        } else {
            std::cout << "Device[" << i << "]: program successful!\n";
            OCL_CHECK(err, krnl_measureLatency = cl::Kernel(program,
"krnl_measureLatency", &err));
            valid_device = true;
            break; // we break because we found a valid device
        }
    }
    if (!valid_device) {
        std::cout << "Failed to program any device found, exit!\n";
        exit(EXIT_FAILURE);
    }

    std::vector<long long int, aligned_allocator<long long int> >
ptr_DDR(1);
    std::vector<long long int, aligned_allocator<long long int> >
ptr_result_DDR(1);

    // These commands will allocate memory on the Device. The cl::Buffer
objects can
    // be used to reference the memory locations on the device.
    OCL_CHECK(err, cl::Buffer buffer_DDR(context, CL_MEM_USE_HOST_PTR |
CL_MEM_READ_ONLY, size_in_bytes, ptr_DDR.data(), &err));
    OCL_CHECK(err, cl::Buffer buffer_result_DDR(context,
CL_MEM_USE_HOST_PTR | CL_MEM_WRITE_ONLY, size_in_bytes,
ptr_result_DDR.data(), &err));

    std::cout << "Allocated read and write buffers on the board. Press
any key to continue ...";
    getchar();

    OCL_CHECK(err, err = krnl_measureLatency.setArg(0, buffer_DDR));
    OCL_CHECK(err, err = krnl_measureLatency.setArg(1,
buffer_result_DDR));
//   //Measure PCIe Write Latency
    cl::Event event;
    gettimeofday(&ts, NULL);
    for (int i = 0; i<N2; i++)
    {
     q.enqueueMigrateMemObjects({buffer_DDR}, 0, NULL, &event);
     event.wait();
    }
    gettimeofday(&te,NULL);
    double PCIeWriteLatency = getElapsed(ts, te);
    printf("PCIe write latency = %lf us\n",PCIeWriteLatency/(double)N2);
```

```
    std::cout << "Press any key to continue to start measuring PCI read
BW...";
    getchar();

//    //Measure PCIe Read Latency
     gettimeofday(&ts, NULL);
     for (int i = 0; i<N2; i++)
     {
      q.enqueueMigrateMemObjects({buffer_result_DDR},
CL_MIGRATE_MEM_OBJECT_HOST, NULL, &event);
      event.wait();
     }
     q.finish();
     gettimeofday(&te,NULL);
     double PCIeReadLatency = getElapsed(ts, te);
     printf("PCIe read latency = %lf us\n",PCIeReadLatency/(double)N2);
    std::cout << "TEST FINISHED" << std::endl;
    return (EXIT_SUCCESS);
}
```