Alma Mater Studiorum · Università di Bologna

Scuola di Scienze
Dipartimento di Fisica e Astronomia
Corso di Laurea in Fisica

Algoritmo di tagging di raggi cosmici in un sistema di acquisizione triggerless per un esperimento di fisica di particelle su fascio

Presentata da:
Antonio Ghinassi

Relatore:
Prof.
Francesco Giacomini

Corelatore:
Dott.
Carmelo Pellegrino

Sessione IV
Anno Accademico 2021-2022

Abstract

I raggi cosmici sono una fonte naturale di particelle ad alta energia ($1{-}10^{19}\,\mathrm{GeV}$) di origine galattica e extragalattica. I prodotti della loro interazione con l'atmosfera terrestre giungono fino alla superficie terrestre, dove vengono rilevati dagli esperimenti di fisica delle particelle come segnale di fondo. Si vuole quindi identificare e rimuovere questo segnale. Gli apparati sperimentali usati in fisica delle particelle prevedono dei sistemi di selezione dei segnali in ingresso (detti *trigger*) per rigettare segnali sotto una certa soglia di energia. Il progredire delle prestazioni dei calcolatori permette oggi di sostituire l'elettronica dei sistemi di *trigger* con implementazioni software (*triggerless*) in grado di selezionare i dati secondo criteri più complessi. TriDAS (Triggerless Data Acquisition System) è un sistema di acquisizione *triggerless* sviluppato per l'esperimento KM3NeT e utilizzato recentemente per gestire l'acquisizione di esperimenti di collisione di fascio ai Jefferson Lab (Newport News, VA). Il presente lavoro ha come scopo la definizione di un algoritmo di selezione di eventi generati da raggi cosmici e la sua implementazione come *trigger* software all'interno di TriDAS. Quindi si mostrano alcuni strumenti software sviluppati per costruire un ambiente di test del suddetto algoritmo e analizzare i dati prodotti da TriDAS.

Prefazione

L'ampliarsi dell'orizzonte della ricerca scientifica a cui assistiamo quotidianamente non sarebbe possibile senza un'altrettanto quotidiana evoluzione degli strumenti di ricerca. I computer diventano sempre più capaci di accorciare i tempi di calcolo e sono ormai l'interfaccia necessaria fra l'essere umano e lo strumento di misura, in particolare negli ambiti della scienza come la fisica in cui la ricerca raggiunge i confini estremi della realtà separandosi completamente dall'esperienza comune. Tuttavia, i computer restano strumenti, il cui valore risiede nella mente che li ha programmati. Essi sono materia morta fino a che le dita creatrici del programmatore non gli infondono ordine e vita. In questo, per me, risiede tutto il fascino e la bellezza della ricerca tecnica ed è questo che mi ha mosso in questi mesi di lavoro. È una sfida con se stesso che l'uomo ingaggia, è una libera creazione che l'uomo intraprende.

Indice

Introduzione

In Fisica gli esperimenti ad alte energie sono quelli che rendono possibile investigare i fenomeni che avvengono nelle parti remote dell'Universo, conoscere i comportamenti delle particelle e ricreare le condizioni in cui l'Universo si trovava nei primi istanti della sua vita. Per esempio, si realizzano esperimenti di collisione di fasci di particelle in cui le particelle accelerate vengono fatte scontrare su un bersaglio, per poi rilevare e analizzare i prodotti della collisione. Oppure, si tenta di intercettare i prodotti degli eventi ad alte energie che già avvengono nel nostro universo, come le esplosioni di stelle o le collisioni di buchi neri. In particolare i neutrini, particelle debolmente interagenti, attraversano l'universo conservando l'informazione dell'evento che li ha generati. Proprio per questo però la loro rilevazione è estremamente ardua. L'esperimento europeo KM3Net si propone di farlo sfruttando la radiazione Cherenkov emessa dalle particelle cariche relativistiche generate dalla fortuita interazione tra i neutrini cosmici e l'acqua o i fondali marini. L'apparato dell'esperimento consiste in centinaia di unità ottiche distribuite in un volume di un km^3 e ancorate al fondale del Mar Adriatico al largo delle coste di Sicilia, Francia e Grecia (Figura 1). Ciascuna unità ottica comprende 31 fotomoltiplicatori (PMT).

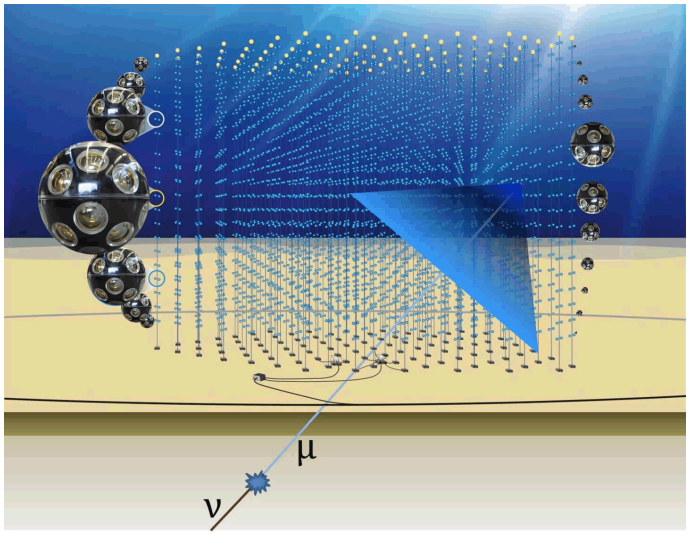

Figura 1 Centinaia di torri di 18 unità ottiche ciascuna ancorate al fondale e sostenute da un galleggiante vanno a formare l'installazione di KM3Net. Quando un neutrino interagisce con la materia terrestre in prossimità del volume del rilevatore, viene generato un muone che produce un cono di luce per effetto *Cherenkov*, il quale viene rilevato dalle unità ottiche. Quindi la traiettoria del neutrino e la posizione celeste dell'evento che lo ha generato vengono ricostruiti. [9]

L'intero apparato genera quindi un flusso di dati di circa 30 GBps, anche a causa del rumore di fondo come il decadimento di ^{40}K, il quale va necessariamente filtrato attraverso un *trigger*, ovvero impostando una soglia di energia o sfruttando le coincidenze temporali date dalla fisica del fenomeno, affinché sia possibile memorizzare e analizzare i dati. Se solitamente questa operazione avviene via hardware direttamente nel luogo di installazione dei sensori, in questo caso la locazione sottomarina impone un approccio *all-data-to-shore*, dove il flusso di dati è trasmesso a terra e quindi processato via software. A questo scopo è stato sviluppato TriDAS (Trigger and Data Acquisition System), un sistema *triggerless* (che non prevede, appunto, un *trigger* hardware) di acquisizione e filtraggio di dati.

Con l'aumento delle capacità dei sistemi di calcolo diventa sempre più comune implementare sistemi *triggerless* via software. Per esempio, presso i *Jefferson Lab*, centro di ricerca statunitense di fisica nucleare (Newport News, Virginia) TriDAS è stato utilizzato nel 2020 per processare i dati di un esperimento di collisione di fascio di elettroni. TriDAS prevede anche la possibilità di inserire altri livelli di *trigger* definiti dall'utente, caratteristica che lo rende molto versatile. Questo lavoro ha come scopo l'implementazione di un livello di *trigger* che, nell'ambito di un esperimento di collisione di fascio, escluda dall'acquisizione quei segnali provenienti da fonti diverse dal fascio, in particolare da raggi cosmici.

Nel primo capitolo si mostra una breve descrizione degli esperimenti di collisione di fascio di particelle a target fisso e del funzionamento dei rilevatori comunemente utilizzati. Il secondo capitolo mostra la struttura e il funzionamento del software TriDAS e la descrizione di uno degli esperimenti di cui TriDAS ha gestito l'acquisizione, realizzato presso i Jefferson Lab. Quindi nel terzo capitolo si descrive come l'algoritmo è stato definito e implementato all'interno del sistema di TriDAS. Si mostrano inoltre strumenti *software* utili a realizzare test e analisi dell'algoritmo.

Capitolo 1

Esperimenti di fascio di particelle a *target* fisso

Da sempre l'uomo ha intuito che la materia, in tutta la sua varietà di forme, colori e consistenze, è il risultato della combinazione di pochi elementi fondamentali. Nel IV secolo a.C., Platone affermava che tutti i corpi sono solidi geometrici, definiti da superfici che è possibile ricoprire utilizzando soltanto triangoli isosceli e scaleni. Quindi questi triangoli sarebbero i costituenti fondamentali dei corpi, come le lettere dell'alfabeto sono i costituenti fondamentali delle sillabe e delle parole. Altri due filosofi dell'antica Grecia, Democrito e Leucippo, hanno introdotto il concetto di atomo come costituente fondamentale e indivisibile della materia. Specie diverse di atomi per forma geometrica e grandezza, conferiscono diverse proprietà alla materia.

Solo all'inizio del secolo scorso, grazie agli esperimenti di Dalton riguardo alle reazioni chimiche e alle teorie di Einstein sul moto Browniano, sono emerse le prime evidenze sull'esistenza degli atomi. La scoperta dell'elettrone da parte di Thomson nel 1897, dimostra l'esistenza di particelle più piccole dell'atomo, inaugurando la fisica nucleare.

1.1 Esperimenti di diffusione

I primi esperimenti che hanno indagato la struttura interna dell'atomo sono quelli svolti da Rutherford, Geiger e Marsden tra il 1908 e il 1913, spinti nella loro ricerca dalle criticità del modello atomico sviluppato qualche anno prima da Thomson. Essi hanno utilizzato un flusso di particelle α, i.e. nuclei di elio, di energia $5.6\,\mathrm{MeV}$ generate dal decadimento del radon, per bersagliare una foglia d'oro di spessore di appena $8.6 \times 10^{-6}\,\mathrm{cm}$ (Figura 1.1). Intorno al bersaglio uno scintillatore composto da vetro dipinto con solfuro di zinco (ZnS) rendeva evidenti le particelle cariche prodotte dalla collisione. Come risultato, una certa percentuale di particelle α veniva deviata ad angoli molto maggiori di $90\,^\circ$ o addirittura respinta, negando definitivamente il modello atomico di Thomson, che prevede un atomo mediamente neutro anche al proprio interno. Infatti, solo una grande concentrazione di carica poteva generare un campo elettrico capace di respingere le pesanti particelle α. Si ipotizza quindi la presenza di un volume all'interno dell'atomo di dimensione molto minore dell'atomo stesso, in cui è concentrata tutta la carica di quest'ultimo: il nucleo.

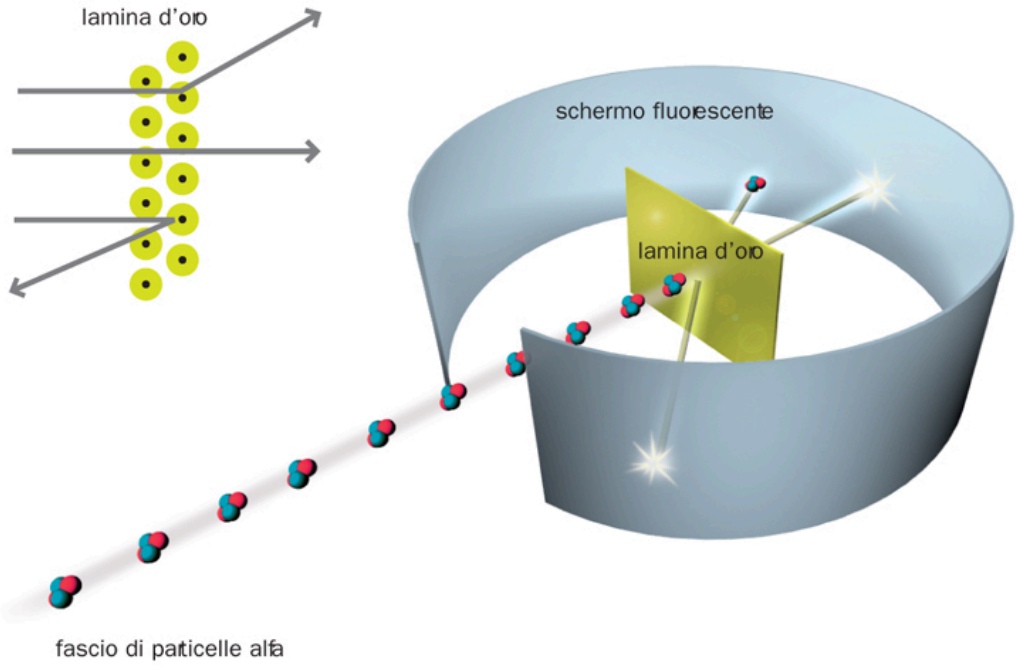

Figura 1.1 L'esperimento ideato da Rutherford prevede un bersaglio d'oro di appena 8.6×10^{-6} cm di spessore, irradiato da un flusso di particelle α generate dal decadimento di una lastra di radon. Il bersaglio è circondato da vetro dipinto con solfuro di zinco (ZnS), che scintilla al contatto con particelle cariche. Le particelle α, cariche positivamente, in alcuni casi proseguono indisturbate e in altri vengono deviate a grandi angoli, fino anche ad essere respinte. Ciò dimostra l'inesattezza del modello atomico di Thomson, in cui la carica positiva dell'atomo ne riempie tutto il volume. Invece deve esistere all'interno dell'atomo una concentrazione di carica positiva abbastanza alta da poter deviare le pesanti particelle α. [5]

Nasce così il modello atomico planetario di Rutherford e le seguenti implicazioni che contribuirono all'origine della fisica quantistica.

Questo tipo di esperimento viene detto oggi "a target fisso", per distinguerlo dai *collisori* in cui due fasci di particelle diretti in direzioni opposte vengono fatti scontrare uno contro l'altro, e viene realizzato ancora nello stesso modo ideato da Rutherford: un fascio di particelle di opportuna energia viene scagliato contro un bersaglio a riposo nel sistema di riferimento dell'osservatore, e i prodotti della collisione sono rivelati con opportuna strumentazione.

L'invenzione e lo sviluppo degli acceleratori ha reso possibile selezionare e aumentare l'energia dei fasci e ridurne la dispersione energetica, parametri fondamentali per la buona riuscita degli esperimenti. Infatti, da essa dipendono il potere risolutivo del fascio e la massa disponibile per la creazione di particelle. A ciascuna particella di quantità di moto p e energia E è associata una lunghezza d'onda $\lambda = h/p$ dove h è la costante di Planck, per il dualismo onda-particella, e una massa relativistica $m = E/c^2$ dove c è la velocità della luce nel vuoto, per la relatività ristretta. Pensando l'onda-particella come una "nube" di probabilità, la lunghezza d'onda ne rappresenta le dimensioni. Quindi, il volume di questa "nube" può essere ristretto aumentando l'energia della particella. Maggiore l'energia, minore la lunghezza

d'onda associata e minori le dimensioni degli oggetti con cui la particella può interagire. Inoltre, durante la collisione, l'energia delle particelle proiettile si deposita sul bersaglio e può eccitarne i nuclei. Questi, legati internamente dalla forza forte, dissipano l'energia producendo nuovi adroni. Maggiore l'energia, maggiore la massa disponibile. Queste nuove particelle possono differire da quelle iniziali sia per natura, sia proprietà cinematiche. Inoltre, studiando le distribuzioni di questi stati finali, si possono ottenere informazioni sulla natura delle interazioni che sono intervenute nella collisione.

1.2 Sezione d'urto

Una delle grandezze fisiche che è possibile misurare in un esperimento di fascio di particelle a target fisso è il numero di particelle prodotte dall'interazione fascio-bersaglio nell'unità di tempo. Per esempio, nell'esperimento ideato da Rutherford, si tratta di contare le particelle α deflesse nell'unità di tempo, $\frac{\mathrm{d}N_{def}}{\mathrm{d}t}$. Questa frequenza è proporzionale al numero di particelle del fascio che arrivano sul bersaglio nell'unità di tempo, $\frac{\mathrm{d}N_f}{\mathrm{d}t}$, allo spessore Δx del bersaglio e alla densità di volume del bersaglio n_b

$$\frac{\mathrm{d}N_{def}}{\mathrm{d}t} = \sigma_r \cdot \frac{\mathrm{d}N_f}{\mathrm{d}t} \cdot n_b \cdot \Delta\mathrm{x} \tag{1.1}$$

Il fattore di proporzionalità σ_r viene chiamato *sezione d'urto* della reazione e ha le dimensioni di una superficie. Essa dipende dalla natura e dalle proprietà delle particelle che costituiscono fascio e bersaglio, oltre che dalla intensità della loro interazione. Di fatto, la sezione d'urto misura la probabilità che una certa interazione abbia luogo. Infatti, integrando la precedente equazione nel tempo, si ottiene

$$\sigma_r = \frac{N_{def}}{N_f}\frac{1}{n_b \cdot \Delta x} \tag{1.2}$$

da cui è possibile determinare il valore della sezione d'urto della reazione. Si vede come la sezione d'urto sia proporzionale alla probabilità $\frac{N_{def}}{N_f}$.

La sezione d'urto è comunemente espressa in *barn*, definito come

$$1 \text{ barn} = 10^{-28}\,\mathrm{m}^2 \tag{1.3}$$

Assumiamo ora un fascio con particelle distribuite in modo uniforme su una superficie Σ. L'equazione 1.1 può essere riscritta come

$$\frac{\mathrm{d}N_{def}}{\mathrm{d}t} = \sigma_r \cdot \phi \cdot N_b \tag{1.4}$$

dove $\phi = \frac{\mathrm{d}N_f}{\mathrm{d}t \cdot \Sigma}$ è il flusso di particelle del fascio per unità di tempo e unità di superficie, e $N_b = n_b \cdot \Sigma \cdot \Delta\mathrm{x}$ è il numero di particelle del bersaglio illuminate dal fascio.

Il flusso ϕ può essere espresso attraverso la densità e la velocità dei proiettili. Infatti, il numero di proiettili può essere scritto come $\mathrm{d}N_f = n_f \Sigma \cdot v\mathrm{d}t$ dove v è la velocità del fascio. Quindi

$$\phi = \frac{n_f \Sigma \cdot v\mathrm{d}t}{\mathrm{d}t \cdot \Sigma} = n_f \cdot v \tag{1.5}$$

1.3 Raggi cosmici

Prima dell'invenzione degli acceleratori, avvenuta nel secondo Novecento, la fisica delle particelle aveva come unico ambito lo studio delle radiazioni ionizzanti prodotte da fenomeni naturali. Nel 1912, Hess mostra la presenza di una radiazione di origine extraterrestre, i raggi cosmici, rilevando come l'intensità di radiazione ionizzante cresce all'aumentare dell'altitudine.

Di origine galattica e extragalattica (*Active Galactic Nuclei*, *Gamma Ray Bursts*), i raggi cosmici (Figura 1.2) sono composti principalmente da protoni (85%), nuclei di elio (12%), nuclei pesanti (1%) e elettroni (2%). Interagendo coi nuclei dell'atmosfera terrestre in collisioni ad alta energia ($1{-}10^{10}$ GeV), essi generano una radiazione secondaria costituita da pioni, mesoni-K e altri adroni più pesanti. Questi a loro volta, se hanno abbastanza energia, possono produrre altri adroni nell'interazione coi nuclei atomici nell'atmosfera terreste, realizzando quella che viene

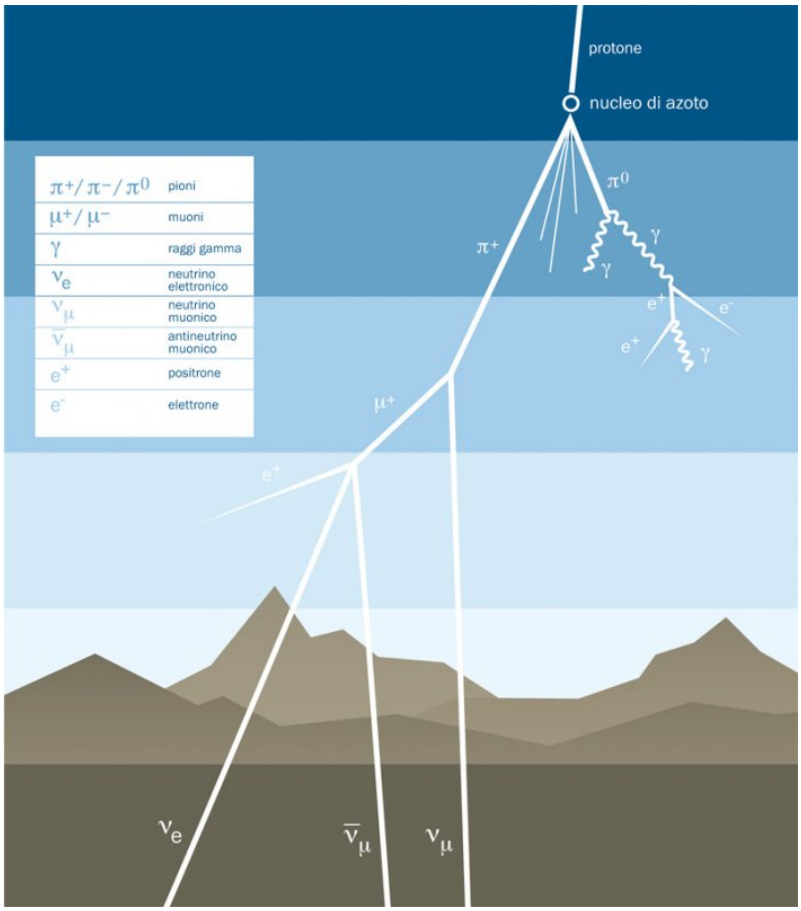

Figura 1.2 Rappresentazione pittorica delle reazioni generate dall'ingresso di un protone cosmico nell'atmosfera. I pioni carichi, prodotti dalla collisione tra protoni e nuclei atomici nell'atmosfera terrestre, decadono in neutrini e muoni e quest'ultimi in ulteriori neutrini e elettroni, realizzando così una *cascata adronica*. I pioni neutri, invece, decadono in una coppia di fotoni, ciascuno dei quali può decadere in una coppia elettrone-positrone. A loro volta elettroni e positroni possono produrre fotoni come radiazione di frenamento, innescando così reazioni a catena dette *cascata elettromagnetica*. [13]

definita una *cascata adronica*. Inizialmente il numero di particelle prodotte nella cascata aumenta fino ad arrestarsi quando l'energia delle particelle si riduce al di sotto della soglia energetica per la formazione di nuove particelle ($E_{th} = m_0c^2$). Infatti gli adroni prodotti sono instabili. I pioni carichi decadono con vita media di appena 26 ns (a riposo) in muoni, a loro volta instabili con vita media (a riposo) di 2 µs. I pioni neutri invece, decadono tramite interazione elettromagnetica in una coppia di fotoni, i quali, interagendo coi nuclei dell'atmosfera, generano elettroni e positroni. Questi emettono a loro volta fotoni come radiazione di frenamento, realizzando così una *cascata elettromagnetica*.

In conclusione, a Terra riceviamo principalmente due componenti della radiazione cosmica: una *soft*, composta dalla cascata elettromagnetica, la quale viene assorbita da materiali come il piombo, e una *hard*, composta principalmente da muoni, la quale è in grado di penetrare anche strati spessi di materia. Una terza componente, estremamente difficile da rivelare, è costituita dai neutrini prodotti dal decadimento degli adroni e dei muoni cosmici in elettroni.

1.4 Scintillatori

Una delle tecniche più usate per rivelare il passaggio di particelle cariche è sfruttare la ionizzazione del mezzo che attraversano. In particolare, esistono materiali chiamati "scintillatori" che presentano caratteristiche ideali. Si possono distinguere due categorie di materiali scintillatori. *Organici*, ovvero composti aromatici come il benzene, caratterizzati da un tempo di risposta di pochi nanosecondi. Possono essere in forma di cristallo (come l'*anthracene*), liquida o plastica. *Inorganici*, ovvero cristalli alcalini come lo ioduro di sodio. Questo tipo di scintillatore ha grande efficienza luminosa, i.e. produce molti fotoni per quantità di radiazione ionizzante, ma ha tempi di risposta più lenti degli scintillatori di tipo organico.

I rivelatori a scintillazione (Figura 1.3) raccolgono i fotoni emessi dallo scintillatore e li incanalano in un fotomoltiplicatore, il quale produce elettroni per effetto fotoelettrico e li converte in segnale elettrico. Il tutto è protetto da un materiale che scherma da campi magnetici esterni, come quello terrestre.

Tipicamente si usano tubi fotomoltiplicatori costituiti da una serie di dinodi, ciascuno con potenziale maggiore del precedente. All'ingresso del fotomoltiplicatore, un catodo viene colpito dai fotoni prodotti dallo scintillatore ed emette elettroni per effetto fotoelettrico, i quali vengono attratti dal primo dinodo della serie a causa della differenza di potenziale. Da qui, un maggior numero di elettroni vengono attirati verso il dinodo successivo per *emissione secondaria*. Quindi, un anodo raccoglie gli elettroni al termine della serie di dinodi, per un fattore totale di moltiplicazione di $10^5 - 10^7$.

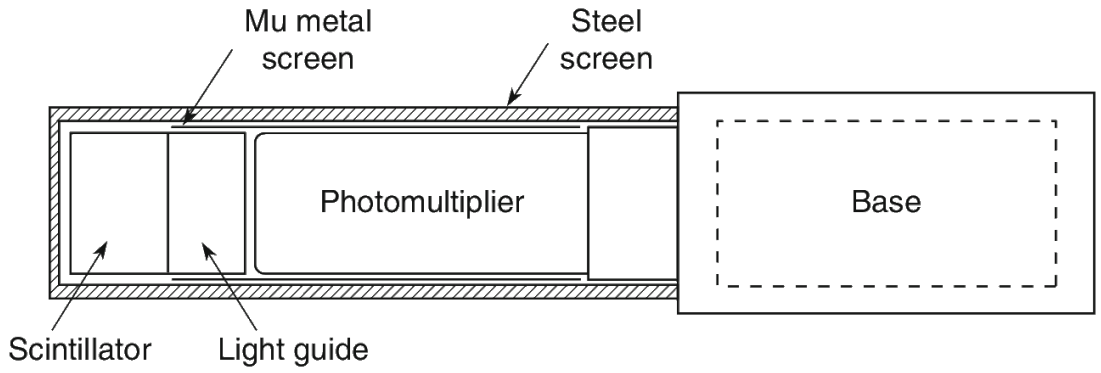

Figura 1.3 Lo schema di un rivelatore a scintillazione. Quando una radiazione ionizzante entra nello scintillatore, ne eccita gli atomi i quali, diseccitandosi, emettono fotoni. Questi entrano nel fotomoltiplicatore dove vengono convertiti in segnale elettrico tramite effetto fotoelettrico. Uno strato di materiale ferromagnetico (*mu-metal*: 80 % nickel, 20 % ferro) scherma da campi magnetici come quello terrestre. [6]

Capitolo 2

TriDAS

I moderni esperimenti nell'ambito della fisica ad alte energie producono sempre più vaste quantità di dati le quali devono essere selezionate, aggregate e salvate in memoria per poter essere analizzate. Con l'intento di ridurre questa mole di dati di diversi ordini di grandezza, si realizzano dei meccanismi di selezione dei dati (*trigger*), le cui caratteristiche variano in base alle necessità dell'esperimento e al fenomeno fisico di interesse. L'evoluzione delle capacità di calcolo dei computer permette oggi di sostituire i classici *trigger* basati su circuiti elettronici (approccio *triggered*) con implementazioni software in grado di disporre di informazioni riguardo l'intero apparato e di elaborarle in modo più complesso (approccio *triggerless*).

Un sistema di questo tipo è stato sviluppato nell'ambito del progetto NEMO (NEutrino Mediterranean Observatory) [8]. L'esperimento prevede rivelatori a scintillazione ancorati al fondale marino, tramite i quali rivelare la radiazione cherenkov emessa dalle particelle cariche generate dalla collisione tra neutrini cosmici e il fondale marino o le molecole d'acqua. La locazione sottomarina dell'esperimento ha richiesto la semplificazione dell'hardware dei rilevatori. Quindi tutti i dati sono inviati a terra dove TriDAS (Triggerless Data Acquisition System) realizza la selezione e l'archiviazione dei dati. Lo scopo del progetto NEMO è stato anche verificare e realizzare le condizioni per la realizzazione del progetto europeo KM3NeT [9], i cui rilevatori ricoprono il volume di un kilometro cubo al largo delle coste di Francia, Italia e Grecia. Quindi, TriDAS è stato scritto in modo da essere scalabile e adattabile a diverse configurazioni di rilevatori, caratteristica che lo ha reso capace di gestire esperimenti in ambiti diversi da quello delle astroparticelle, come gli esperimenti di collisione di fascio ai Jefferson Lab (Newport News, Virginia, US) [3].

2.1 TriDAS-core

TriDAS (Triggerless Data Acquisition System) [10] è un sistema di acquisizione dati *triggerless* sviluppato originariamente per il prototipo di rilevatore per neutrini NEMO. Il progetto NEMO ha avuto un'evoluzione graduale secondo diverse fasi di test, richiedendo al sistema di acquisizione di essere scalabile con i cambiamenti della struttura del rilevatore, del flusso di dati, etc. Quindi, TriDAS implementa un design modulare, dove ciascun componente ricopre un ruolo specifico nella catena di elaborazione e acquisizione dati (Figura 2.1). I componenti sono scritti in `C++11`

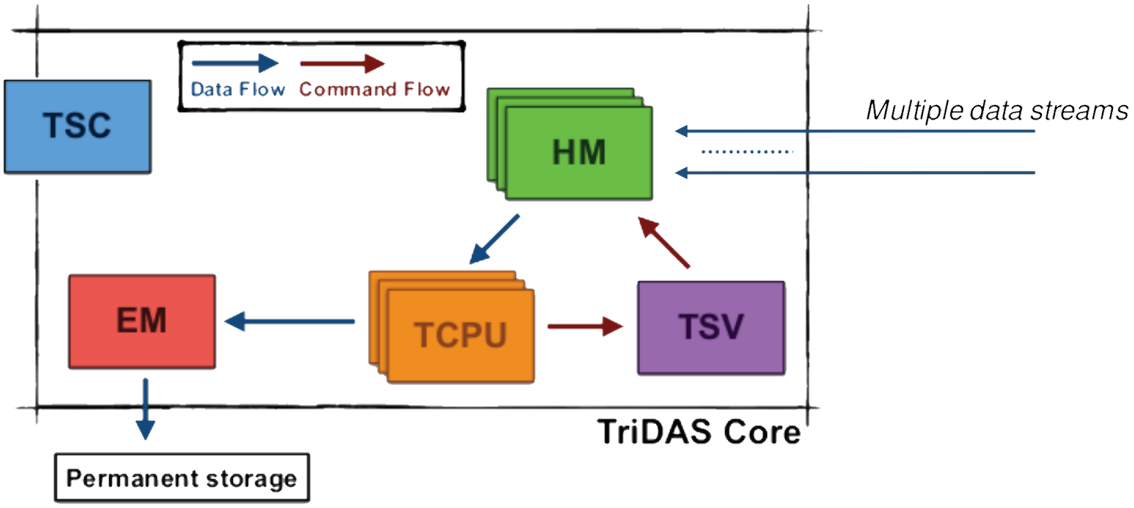

Figura 2.1 Rappresentazione dei componenti di TriDAS e delle loro interazioni. Ciascun *Hit Manager* (HM) riceve un flusso continuo di dati dall'elettronica del rilevatore e lo divide in intervalli temporali di lunghezza prefissata. Condividendo lo stesso orologio, gli HM inviano a una *Trigger CPU* (TCPU) i dati dei diversi flussi che si riferiscono allo stesso intervallo di tempo. Il *TriDAS Supervisor* (TSV) regola questo scambio indicando agli HM quale TCPU sia disponibile a processare i dati. Quindi, la TCPU sottopone i dati agli algoritmi di *trigger* e invia all'*Event Manager* (EM) le porzioni di segnale che li hanno soddisfatti, per essere salvati in memoria. L'utente interagisce e configura il sistema tramite il *TriDAS System Controller* (TSC). [7]

e comunicano tramite connessioni TCP e il sistema di messaggistica fornito dalla libreria *ZeroMQ* e *Boost Asio*.

2.1.1 *Hit Manager* (HM)

Gli *HitManager* (HM) sono il primo punto di aggregazione dei dati. Ciascun settore del rilevatore (tipicamente un gruppo di sensori) invia un flusso continuo di dati a un HM. L'HM quindi divide il flusso di dati in buffer ordinati temporalmente di durata definita dall'utente detti *Sector Time Slice* (STS). Essendo sincronizzati temporalmente, gli HM inviano a una TCPU tutte le STS che si riferiscono allo stesso intervallo di tempo, detto *Time Slice* (TS) (Figura 2.2).

2.1.2 *Trigger CPU* (TCPU)

La *Trigger CPU* (TCPU) esegue l'analisi *online* dei dati. Ciascuna TCPU riceve dagli HM tutte le STS che si riferiscono a una TS, e le aggrega in un oggetto chiamato *Telescope Time Slice* (TTS). In questo modo, ciascuna TCPU dispone del segnale trasmesso da tutto il rilevatore nella finestra di tempo di una certa TS. Quindi, la TCPU sottopone la TTS a due livelli di *trigger*. Il primo (L1) scorre la TTS in cerca di eccessi di carica o segnali correlati spazio-temporalmente, detti *seed* (Figura 2.3). Gli eccessi di carica sono definiti da una certa soglia regolabile dall'utente nel file di configurazione. Quindi, per ognuno, la TCPU costruisce un evento detto *Triggered Event* (TE) con l'intervallo di segnale centrato nel *seed* di ampiezza

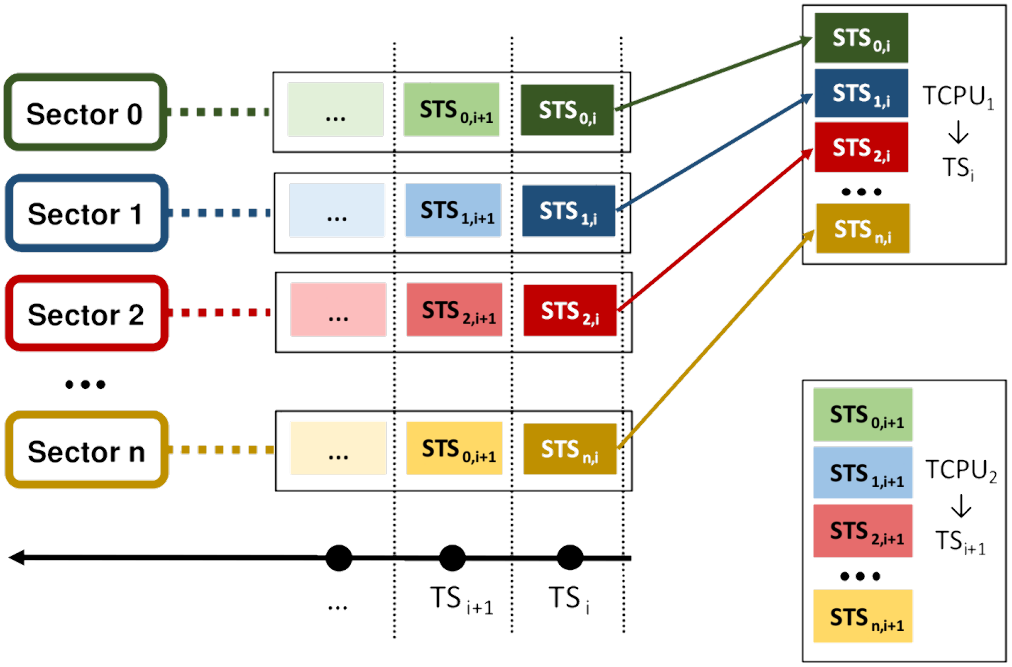

Figura 2.2 Rappresentazione del meccanismo di aggregazione dati degli *Hit Managers* (HM). L'apparato sperimentale è suddiviso in un certo numero di settori, ciascuno dei quali invia un flusso continuo di dati a un HM. Questo divide il flusso in intervalli temporali detti *Sector Time Slice* (STS). Condividendo lo stesso orologio, gli HM inviano a una TCPU tutte le STS che si riferiscono a uno stesso intervallo di tempo, detto *Time Slice* (TS). [7]

definibile dall'utente. Se due eventi si intersecano, viene creato un unico evento il cui intervallo è l'unione dei due. Considerando l'ampiezza tipica di un evento (~ns) e quella di una TS (~µs), la probabilità che il segnale di un evento venga spezzato tra due STS è estremamente ridotta. Il secondo (L2) implementa algoritmi più complessi e li esegue sugli eventi costruiti da L1. Se un evento soddisfa anche questo secondo livello, viene segnato con un tag, per essere poi salvato permanentemente in memoria.

Ogni TCPU procede su un proprio thread in modo da elaborare più TTS contemporaneamente. Anche gli algoritmi L2 procedono ognuno su un proprio thread in modo da analizzare contemporaneamente una stessa TTS. Inoltre, ciascun algoritmo viene caricato dinamicamente con un eseguibile esterno all'avvio della TCPU. In questo modo i *trigger* L2 possono essere cambiati tra diverse prese dati dello stesso esperimento, senza dover ricompilare il codice.

È importante scalare il numero di TCPU e di HM con il flusso di dati in ingresso. Infatti, se non sono disponibili TCPU a ricevere le STS processate dagli HM, questi continuano a ricevere dati, sovrascrivendo i propri buffer interni, risultando in una perdita di dati. (**da verificare**)

2.1.3 *Event Manager* (EM)

L'*Event Manager* (EM) si occupa di salvare in memoria permanente i dati selezionati dai *trigger*. Esso riceve gli eventi creati da tutte le TCPU e li scrive in memoria

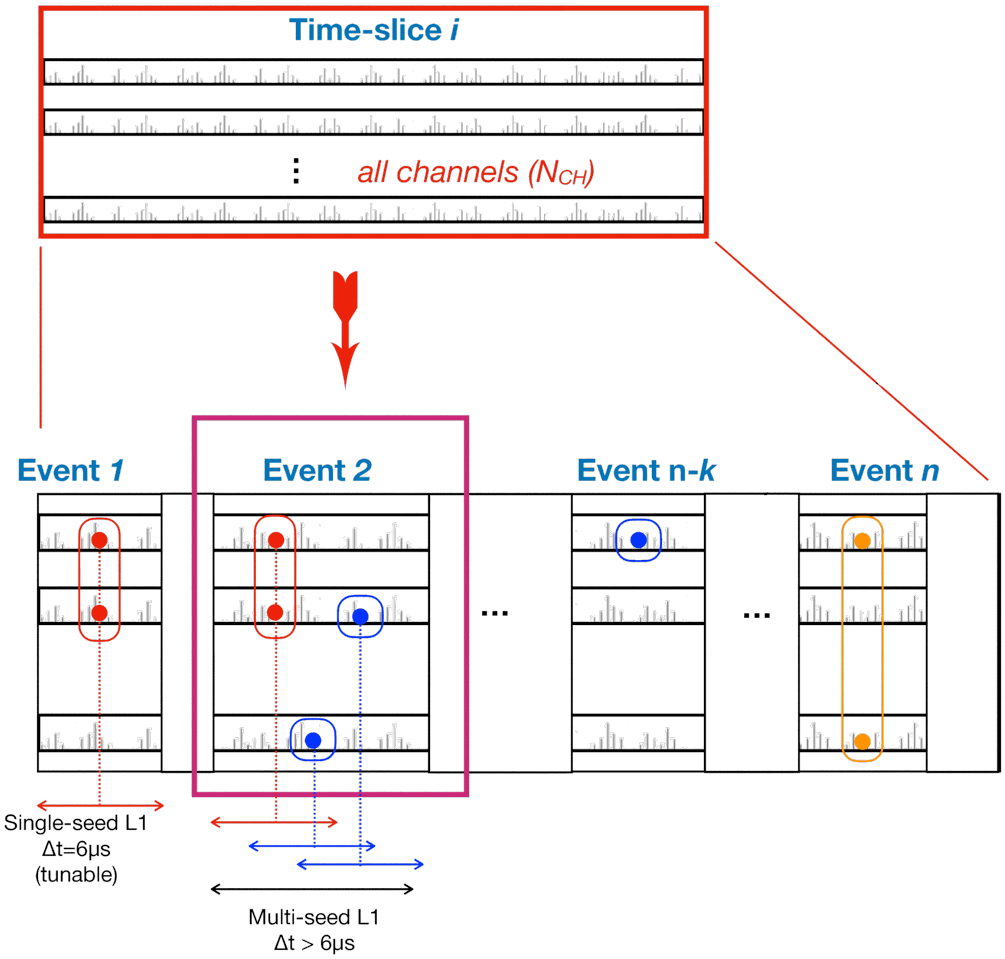

Figura 2.3 Rappresentazione del processo di costruzione di eventi dell'algoritmo L1. L'algoritmo cerca segnali che eccedono una certa soglia definibile dall'utente (indicati in blu) e segnali che siano correlati spazio-temporalmente tra i diversi sensori (indicati in rosso e giallo), detti *seed*. Per ciascuno di questi, l'algoritmo costruisce un evento con l'intervallo di segnale centrato nel *seed* di ampiezza definibile dall'utente. Se due eventi si intersecano, vengono estesi e uniti in uno unico. [7]

sotto forma di *Post Trigger* (PT) file. Sia $E_{j,k}$ l'evento j-esimo appartenente alla TS k-esima. Il PT file è una sequenza di dati binari che rappresentano nell'ordine: informazioni sulla presa dati (tempo di inizio della presa dati, numero di eventi totali, numero di TS totali, dimensione del file, etc.), file di configurazione (geometria del rilevatore, parametri dei *trigger*, posizione degli eseguibili dei *trigger* L2, etc.), informazioni sulla TS_1 (numero identificativo dell TS, numero di eventi contenuti nella TS, dimensione della TS), informazioni sull'evento $E_{1,1}$ (numero di hit contenute nell'evento, numero di *seed* del *trigger* L1, numero di *seed* di ciascun *trigger* L2, etc.), informazioni sulle hit contenute nell'evento $E_{1,1}$ (carica, tempo di occorrenza, identificativo del sensore che l'ha ricevuta, etc.), informazioni sull'evento $E_{2,1}$, etc.

Riportando anche le informazioni sull'esperimento, la presa dati e la geometria dei rilevatori, ciascun PT file permette di eseguire un'analisi offline dei dati. In più, ciò ha un impatto trascurabile sulla dimensione del PT file, in quanto si tratta di pochi KiloBytes, a fronte di una dimensione totale del file dell'ordine dei GigaBytes.

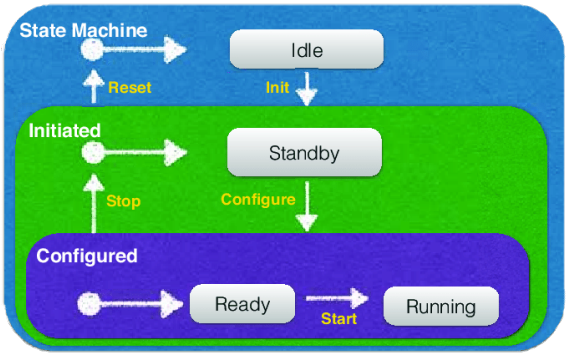

Figura 2.4 Rappresentazione del funzionamento del *TriDAS System Controller*. Una volta avviato, TriDAS viene inizializzato con il comando *init*, passando come argomento il file di configurazione. Quindi i vari moduli vengono configurati con il comando *configure*, avviati con il comando *start* e interrotti con il comando *stop*. Il sistema può essere inizializzato di nuovo, eseguendo prima il comando *reset*. [7]

Una libreria è fornita per la lettura dei PT file.

2.1.4 *TriDAS SuperVisor* (TSV)

Il *TriDAS SuperVisor* (TSV) regola lo scambio di dati tra gli HM e le TCPU, tenendo un elenco delle TS che man mano vengono processate. Quando una TCPU è disponibile a processare dati, invia un *token* al TSV, il quale le assegna una TTS tra quelle ancora non processate. Quindi, il TSV invia un messaggio all'HM che ha costruito la TTS assegnata indicandogli il numero della TCPU a cui inviarla. Diverse TS possono avere tempi di processamento diversi in base al contenuto di *seed*, perciò si implementa un approccio *first-come first-served*, ovvero appena una TCPU è disponibile, le viene assegnato del lavoro, in modo da ridurre i tempi di attesa.

2.1.5 *TriDAS System Controller*(TSC)

Il *TriDAS System Controller* (TSC) è l'interfaccia tramite la quale controllare e configurare TriDAS. Il TSC quindi gestisce i processi dei vari moduli di TriDAS secondo una macchina a stati (Figura 2.4). Inizialmente nello stato *idle*, TriDAS viene inizializzato tramite il comando *init*, che riceve come argomento il file di configurazione (formato *json*), giungendo allo stato *standby*. Quindi, il sistema può essere configurato con il comando *configure* e avviato col comando *start*. Fermato col comando *stop*, il sistema mantiene il file di configurazione in memoria, a meno che non si chiami anche il comando *reset*.

2.2 Test presso i Jefferson Lab

L'esperimento CLAS12 (CEBAF Large Acceptance Spectrometer for operations at 12 GeV beam energy) presso i Jefferson Lab (Newport News, Virginia) è usato per

studiare reazioni adroniche e nucleari indotte dalla collisione di elettroni [3]. Contatori Cherenkov, scintillatori e calorimetri elettromagnetici permettono di identificare gli elettroni diffusi e gli adroni prodotti nelle collisioni. Quindi, un veloce meccanismo di *triggering* e un'alta frequenza di acquisizione dati (frequenza di eventi prodotti pari a $30\,\mathrm{kHz}$) permettono di operare a una luminosità di $10^{35}\,\mathrm{cm^{-2}s^{-1}}$. L'apparato dell'esperimento sta subendo un aggiornamento che porterà la luminosità ad aumentare di un fattore due. Si prevede di poter aumentare l'efficienza dell'attuale sistema di trigger basato su elettronica fino a raggiungere una frequenza di eventi prodotti di $100\,\mathrm{kHz}$. Per raggiungere prestazioni più elevate, è in considerazione l'utilizzo di un sistema *triggerless*. Verrebbero così mitigate alcune limitazioni, come l'identificazione di particelle neutre (fotoni prodotti dal decadimento di pioni neutri, controllando il valore della massa invariante) e si avrebbe un miglioramento della precisione della selezione, inficiata dal rumore prodotto dalle cascate adroniche. Si avrebbe anche una più accurata selezione dei canali di reazione, potendo disporre delle informazioni di tutte le parti del rilevatore (rilevatori Cherenkov e calorimetri). Il flusso di dati prodotto dall'apparato di CLAS12 è pari a $50\,\mathrm{GBps}$, da ridurre almeno di un fattore 10 perché sia possibile salvarne il contenuto in memoria.

L'approccio *triggerless* all'acquisizione dati è stato selezionato anche per il futuro esperimento EIC (Electron-Ion Collider) al Brookhaven National Laboratory (New York). Trattandosi di una tipologia inedita di esperimento di collisione di fascio, nuovi tipi di sensori verranno implementati aprendo possibilità di integrazione più stretta con il sistema di acquisizione dati. In particolare, si avranno vantaggi quali una calibrazione *online* dei sensori per compensare le perdite di segnale dovute alle cascate elettromagnetiche, implementazione di algoritmi di intelligenza artificiale per un migliore riconoscimento dei pattern e delle tracce e più accurata distinzione tra cascate acroniche e elettromagnetiche. In più, possono essere implementati livelli di *trigger* superiori per cercare particolari eventi di interesse. Ai Jefferson Lab è stato realizzato un prototipo composto da calorimetri dell'esperimento EIC, il quale è stato utilizzato per testare le prestazioni di TriDAS nel contesto della catena di acquisizione dati (schede elettroniche di interfaccia, rete di network, CPU etc.) e confrontarne le prestazioni con l'attuale sistema di acquisizione *triggered*. Il prototipo è formato da una matrice 3×3 di cristalli $PbWO_4$ ed è stato installato nella *Hall-D* dei Jefferson Lab, all'uscita del fascio secondario di elettroni prodotto dal *Pair Spectrometer* (Figura 2.5 e Figura 2.6). Fasci secondari di elettroni e positroni sono prodotti dall'interazione del fascio primario di fotoni con un convertitore in berillio di spessore $750\,\mu\mathrm{m}$ e separati da un campo magnetico di $1.5\,\mathrm{T}$. Ciascun cristallo ($2.05\,\mathrm{cm} \times 2.05\,\mathrm{cm} \times 20\,\mathrm{cm}$) è collegato a un fotomoltiplicatore di $19\,\mathrm{mm}$ di diametro. Il prototipo è stato posizionato in modo che il centro del fascio elettronico illumini il cristallo centrale della matrice, con inclinazione perpendicolare alle facce dei cristalli. Il test è stato svolto durante la presa dati dell'esperimento *GlueX* con fascio di fotoni da $350\,\mathrm{nA}$ e un fascio secondario di elettroni da $4.7\,\mathrm{GeV}$. Il segnale di ciascun PMT è stato digitalizzato e inviato a TriDAS, settando una soglia di $\sim 2\,\mathrm{GeV}$ per gli eventi L1 (Figura 2.7). I risultati ottenuti con con TriDAS sono compatibili con quelli ottenuti utilizzando il sistema *triggered*, mostrando l'affidabilità del sistema di acquisizione *triggerless* [3].

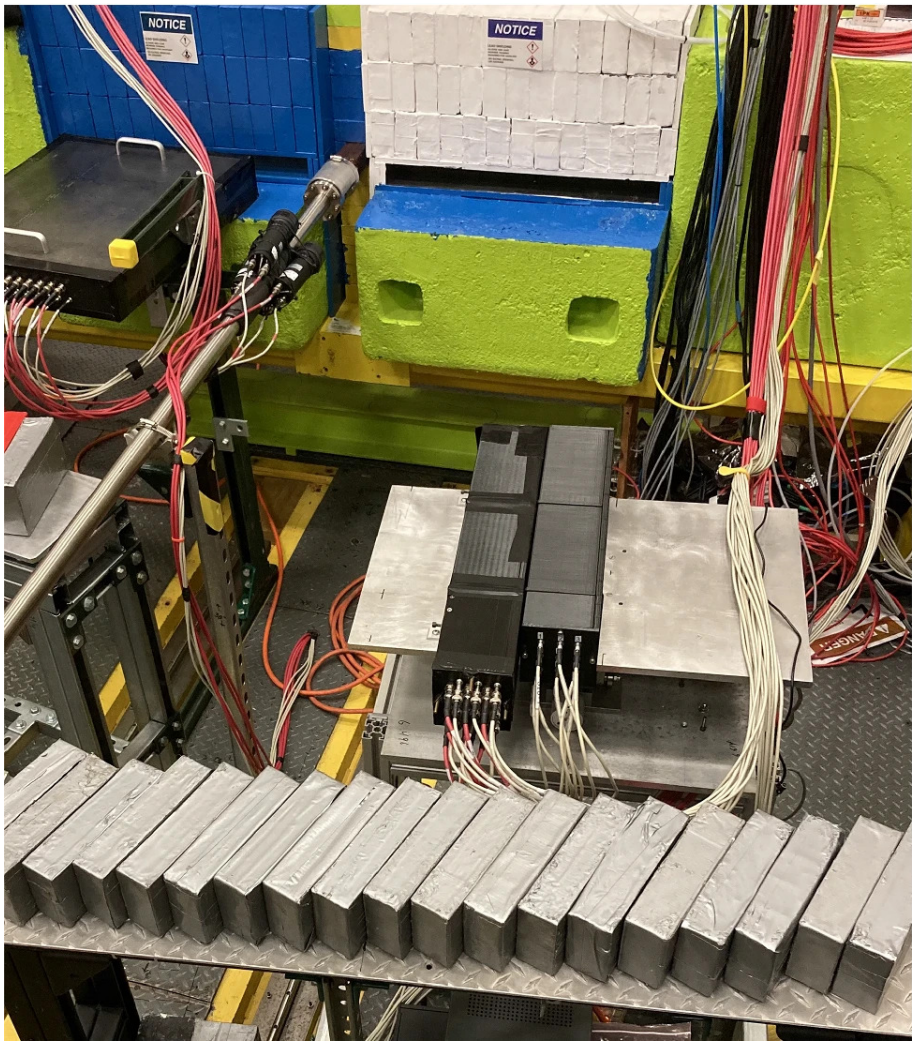

Figura 2.5 Il prototipo installato nella *Hall-D* dei Jefferson Lab all'uscita del fascio secondario di elettroni prodotto per produzione di coppia dal fascio primario di fotoni. [3]

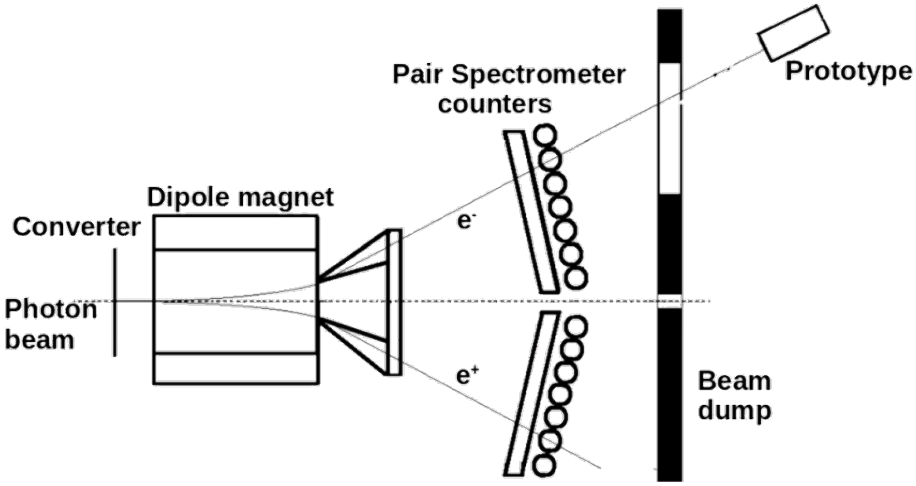

Figura 2.6 Schema dell'apparato sperimentale realizzato nella *Hall-D* dei Jefferson Lab. Un prototipo costituito da rilevatori a scintillazione è installato all'uscita del fascio di elettroni prodotto per produzione di coppia dal fascio primario di fotoni. Un magnete separa e indirizza i fasci secondari di elettroni e positroni. [3]

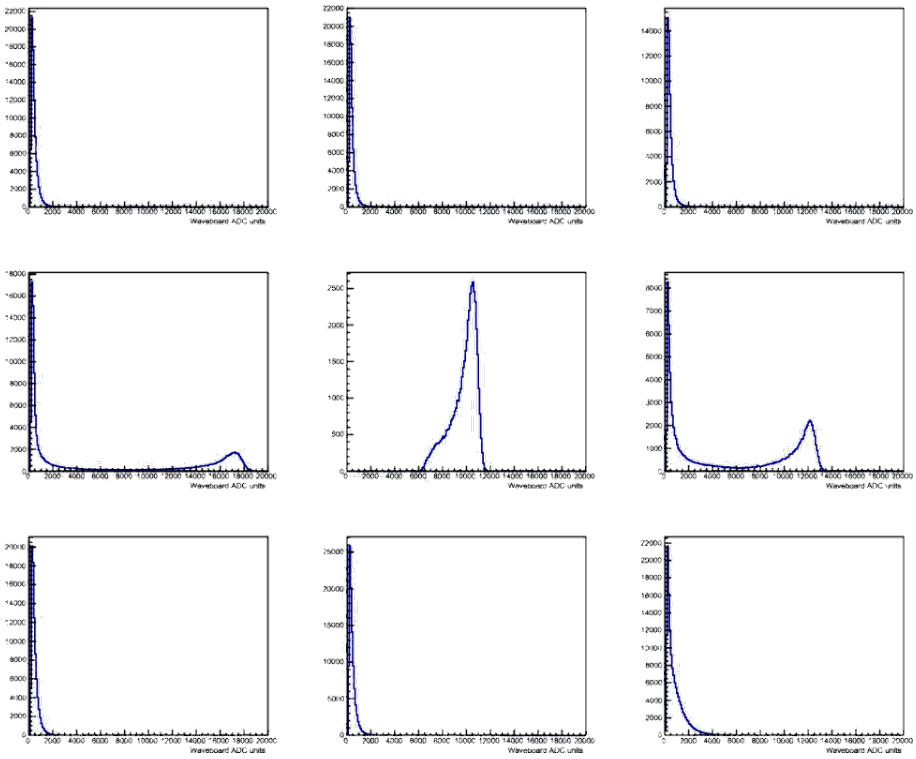

Figura 2.7 Risposta in energia dei nove sensori del prototipo acquisita usando TriDAS. Il fascio era centrato nel sensore centrale della matrice. [3]

Capitolo 3

Algoritmo di *tagging* di raggi cosmici

3.1 Trigger *TrigFromBeam*

I raggi cosmici sono una fonte naturale di particelle cariche ad alta energia che, interagendo con l'atmosfera, giungono alla superficie terrestre. Tutti gli esperimenti che utilizzano rilevatori di particelle cariche presentano un segnale di fondo generato dalla presenza di raggi cosmici. In alcuni casi questo può minacciare la riuscita dell'esperimento, per esempio nel caso si voglia rilevare eventi meno energetici e rari come i neutrini [1]. A seconda delle caratteristiche dell'esperimento e del fenomeno fisico di interesse, è possibile simulare questo segnale di fondo per poi andare a rimuoverlo. Altrimenti è possibile sviluppare algoritmi di identificazione degli eventi generati dai raggi cosmici.

Si presenta l'algoritmo di identificazione di eventi generati da raggi cosmici sviluppato per il prototipo di rivelatore descritto nel capitolo precedente (matrice 3×3 di calorimetri e fotomoltiplicatori, sezione 2.2). L'acquisizione dati è gestita da TriDAS e l'algoritmo è inserito come *trigger* L2. Per prima cosa si mostra l'interfaccia definita da TriDAS per un *trigger* L2. Il *trigger* L2 (o *trigger plugin*), in questo caso *TrigFromBeam*, è dichiarato come una funzione che prende come argomento un oggetto di tipo *PluginArgs*

```
void TrigFromBeam(PluginArgs const& args);
```

con

```
struct PluginArgs
{
  int id;
  EventCollector* evc;
  Geometry const* geom;
  Configuration const* params;
};
```

Quindi l'algoritmo dispone degli eventi appartenenti a una certa *TimeSlice* (TS), della geometria dell'apparato e del file di configurazione fornito dall'utente. L'ener-

gia media di un muone cosmico alla superficie terrestre (~4 GeV) è tale da superare la soglia del *trigger* L1 (2 GeV). Quindi, le *hit*, i.e. picchi di segnale, degli eventi generati da un muone cosmico vengono selezionati dal *trigger* L1. Per ciascun evento della TS, l'algoritmo riempie un vettore chiamato `seeds` con i *seed* del *trigger* L1, ovvero le *hit* che il *trigger* ha selezionato.

```
    std::vector<PMTHit const*> seeds;
```

Le *hit* sono rappresentate dal tipo `PMTHit`, che contiene l'indice identificativo del sensore che ha prodotto la *hit*, il tempo di arrivo, la carica elettrica del picco di segnale, etc. Quindi, per ciascun *seed* `this_seed` del vettore `seeds`, si verifica la presenza di altri *seed* correlati spazio-temporalmente a `this_seed`. In caso affermativo, i *seed* corrispondenti sono inseriti in un vettore chiamato `cosmic_trace`.

Listato 3.1 Algoritmo di selezione di *hit* generate da raggi cosmici implementato dalla funzione `TrigFromBeam`.

```
for (auto const& this_seed : seeds) {
    std::vector<PMTHit const*> cosmic_trace;
    cosmic_trace.reserve(seeds.size());

    std::copy_if(seeds.begin(), seeds.end(), std::back_inserter(cosmic_trace),
          [&geom, &this_seed](auto const& other_seed) {
            return this_seed != other_seed
            && spaceCorrelation(*this_seed, *other_seed, geom)
            && timeCorrelation(*this_seed, *other_seed, geom);
          });

    if (!cosmic_trace.empty()) { // seed is from cosmic ray
        // get information about cosmic ray
    } else { // seed is from beam
        // select seed to be saved in the PT file
    }
}
```

La correlazione spaziale e temporale dipende dalla geometria dell'apparato e dal fenomeno fisico in questione. Nel caso dei raggi cosmici, possiamo considerare in prima approssimazione muoni che volano in direzione perpendicolare alla superficie terrestre a velocità prossime a c. L'apparato del rivelatore è il prototipo descritto nel capitolo precedente (sezione 2.2). Consideriamo quindi una matrice 3×3 di rilevatori, ciascuno di sezione quadrata di lato d. Quindi, nel caso di passaggio di un muone cosmico all'interno del rivelatore, ci aspettiamo di trovare *hit* successive in una colonna di rilevatori, a distanza temporale di $\sim \frac{d}{c}$.

Le correlazioni spaziale e temporale sono definite dalle funzioni `timeCorrelation` e `spaceCorrelation`. La funzione `timeCorrelation` prende come argomento due *hit* e restituisce `vero` o `falso` a seconda che siano soddisfatti i criteri di correlazione temporale. Per tenere conto di possibili variazioni statistiche del valore della velocità di volo e di traiettorie non propriamente verticali, la funzione verifica che la distanza temporale tra le due *hit* cada all'interno di un intervallo. La distanza temporale minima `shortest_correlation_time_frame` tra due *hit* è $\frac{d}{c}$, ovvero quella

generata da un muone che cade perpendicolare alla superficie terrestre a velocità c. La distanza temporale massima `longest_correlation_time_frame` è definita come $\frac{\sqrt{2}d}{0.9\ c}$, ovvero quella tra due *hit* generate da un muone che cade con inclinazione di $45°$ rispetto alla superficie terrestre a velocità $0.9c$.

Listato 3.2 Algoritmo di correlazione temporale implementato dalla funzione `timeCorrelation`. L'unità di misura `fine_time` corrisponde a 10^{-9} s.

```
fine_time real_time_frame{
    seed.IsOlderThan(other_seed)
        ? seed.get_fine_time() - other_seed.get_fine_time()
        : other_seed.get_fine_time() - seed.get_fine_time()};

return shortest_correlation_time_frame < real_time_frame
    && real_time_frame < longest_correlation_time_frame;
```

La funzione `spaceCorrelation`, analogamente alla precedente, prende come argomento due *hit* e verifica i criteri di correlazione spaziale. Sostanzialmente c'è correlazione quando la traiettoria tracciata dalle due hit è compatibile con un'inclinazione massima di $45°$ rispetto alla verticale. Per controllare questa condizione si sfrutta la disposizione dei sensori all'interno della matrice. Ciascuna *hit* contiene l'indice del sensore che l'ha prodotta. I sensori sono identificati da indici crescenti, in questo caso da `0` a `8`. Quindi l'algoritmo fa uso delle funzioni `row` e `col` per ottenere gli indici riga e colonna di ciascun sensore all'interno della matrice 3×3. Per esempio, il sensore con indice `6` sarà il primo dell'ultima riga, ovvero indice riga `2` e indice colonna `0`. Ora, supponiamo che la *hit* `this_seed` sia prodotta dal sensore nella posizione `(1,0)` della matrice (Figura 3.1). Le possibili traiettorie percorse da un muone cosmico per arrivare al sensore `(1,0)` attraversano il sensore soprastante `(0,0)` o uno di quelli adiacenti ai vertici `(0,1)`. Quelli sottostanti di conseguenza. Si tratta quindi di implementare una regola sugli indici in modo da selezionare i sensori corretti. Preso un elemento della matrice $e_{i,j}$, allora le due diagonali che lo comprendono sono definite da

$$d_a = \{e_{i',j'} \mid i' + j' = a\} \tag{3.1}$$

e

$$d_b = \{e_{i',j'} \mid i' - j' = b\} \tag{3.2}$$

con

$$a := i + j,\ b := i - j \tag{3.3}$$

Gli elementi soprastanti $e_{i,j}$ compresi tra le due diagonali d_a e d_b sono definiti da

$$I_{up} = \{e_{i',j'} \mid i' + j' \leq a \ \wedge \ i' - j' \leq b\} \tag{3.4}$$

Gli elementi sottostanti $e_{i,j}$ compresi tra le due diagonali sono definiti da

$$I_{down} = \{e_{i',j'} \mid i' + j' \geq a \ \wedge \ i' - j' \geq b\} \tag{3.5}$$

Se un *seed* è stato prodotto da un sensore la cui posizione appartiene all'insieme unione $I_{up} \ \cup \ I_{down}$, allora può essere correlato al seed `this_seed`, prodotto dal

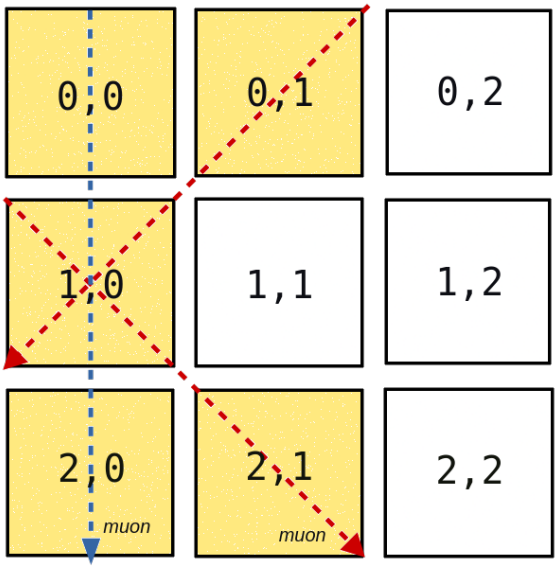

Figura 3.1 Schema della sezione trasversale dell'apparato sperimentale. Le linee tratteggiate rosse e blu rappresentano le possibili traiettorie più lunghe e più corte rispettivamente con cui i muoni cosmici possono attraversare il sensore `(1,0)` dell'apparato, attivando di conseguenza i sensori indicati in giallo. All'interno di ciascun sensore sono indicati gli indici riga e colonna.

sensore che occupa l'elemento $e_{i,j}$ della matrice. L'algoritmo verifica anche che una *hit* prodotta da un sensore soprastante il sensore che ha prodotto `this_seed`, sia precedente temporalmente a `this_seed`. Analogamente, una *hit* prodotta da un sensore sottostante il sensore che ha prodotto *this_seed*, deve essere anche successiva temporalmente a `this_seed`.

Listato 3.3 Algoritmo di correlazione spaziale implementato dalla funzione `spaceCorrelation`.

```
if (seed_row > other_seed_row
    && seed.IsOlderThan(other_seed)) // space and time order respected
  return other_seed_row + other_seed_col <= seed_row + seed_col
     && other_seed_row - other_seed_col <= seed_row - seed_col;

if (seed_row < other_seed_row
    && other_seed.IsOlderThan(seed)) // space and time order respected
  return other_seed_row + other_seed_col >= seed_row + seed_col
     && other_seed_row - other_seed_col >= seed_row - seed_col;

return false; // here if other_seed_row == seed_row
```

Infine, se un *seed* non risulta correlato spazio-temporalmente con nessun altro *seed* dell'evento significa che è stato prodotto da una particella del fascio, e viene quindi selezionato per essere salvato in memoria all'interno del PT file. Inoltre tramite la funzione `getParticleEnergy` viene calcolata la somma della carica del *seed* con le cariche di tutte le *hit* dello stesso evento distanti al più `8 ns` dal *seed*, fornendo così la carica totale dell'evento, proporzionale all'energia della particella.

Si vede come l'algoritmo sia strutturato in modo da sfruttare una logica generale

tramite le funzioni `timeCorrelation` e `spaceCorrelation`, lasciando a queste l'implementazione dei criteri specifici di correlazione spazio-temporale per l'esperimento in questione. In questo modo si è cercato di rendere l'algoritmo il più possibile astratto e adattabile a diverse configurazioni dell'apparato sperimentale.

L'algoritmo fa uso del sistema di *logging* `TRIDAS_LOG` implementato in TriDAS per comunicare risultati e errori.

Listato 3.4 Esempio di uso del sistema di *log*. Prima di terminare l'esecuzione, viene riportato il numero di *seed* selezionati dal *trigger* L2.

```
TRIDAS_LOG(INFO) << "TrigFromBeam triggered " << evc.stats_for_plugin(id)
                 << " of " << evc.used_trig_events() << " events in TTS "
                 << evc.ts_id();
```

Il programma è stato integrato nel sistema di compilazione *CMake* di TriDAS.

3.2 Ambiente di test

TriDAS prevede un algoritmo di esecuzione *offline* con lo scopo di sottoporre i dati contenuti in un PT file precedentemente acquisito a un *trigger plugin* (L2). Questo algoritmo è stato perfezionato e reso funzionante. Inoltre è stato completata la configurazione di un ambiente *docker compose* con cui eseguire localmente TriDAS in fase di test.

3.2.1 Applicazione *offline* del *trigger* a PT file

Il programma `trig_sim` esegue in locale una TCPU e un EM, sottoponendo ai *trigger* i dati contenuti in un PT file e salvando i risultati in un nuovo PT file. Il programma prende quattro argomenti da riga di comando: la porzione dell'eseguibile della TCPU, il file di configurazione di TriDAS, il PT file da analizzare, il nome del nuovo PT file in cui salvare i dati. L'interfaccia fa uso della libreria `boost::program_options`.

Per prima cosa si avvia la TCPU come sottoprocesso. L'eseguibile della TCPU richiede due argomenti da riga di comando: identificativo della TCPU e file di configurazione di TriDAS.

```
std::string const tcpu_cmd_line = tcpu_path + " -i 0 -d " + cardfile;
Subprocess tcpu_process(tcpu_cmd_line);
```

Quindi, vengono create le socket necessarie a realizzare le connessioni tra i diversi moduli e connesse alle porte della rete locale indicate nel file di configurazione.

Listato 3.5 Parte del file di configurazione in cui si specificano alcuni parametri richiesti dalla TCPU, come le porte della rete locale a cui connettersi.

```
"TCPU": {
    "LOG_LEVEL": "DEBUG",
    "LOG_TO_SYSLOG": "false",
    "BASE_CTRL_PORT": "6200",
    "BASE_DATA_PORT": "6300",
    "PLUGINS_DIR": "\/tridas\/build-test\/DAQ\/TCPU\/trigger_plugins",
```

```
    "PARALLEL_TTS": "10",
    "HOSTS": [
        {
            "CTRL_HOST": "localhost",
            "DATA_HOST": "localhost",
            "N_INSTANCES": "1"
        }
    ],
...
}
```

Quindi, in un nuovo *thread*, viene avviato il `FileWriter`, componente dell'EM, il quale resta in attesa di ricevere eventi da scrivere nel PT file. Il PT file viene letto utilizzando la classe `PTFileReader` di TriDAS, la quale permette di scorrere le le TS, gli eventi di ciascuna TS e le *hit* di ciascun evento. Quindi, per ciascuna TS viene costruita una TTS e inviata alla TCPU tramite la socket `hm2tcpu`. La TCPU quindi analizza la TTS, costruisce gli eventi e li invia al `FileWriter` utilizzando la socket `tcpu2em`.

3.2.2 Ambiente *docker* di esecuzione

Docker compose è una tecnologia che permette di definire e eseguire applicazioni che richiedono la connessione di diversi *container*. I *container* dell'applicazione, detti "servizi", sono definiti in un file di configurazione formato `YAML`. Ciascun servizio esegue un modulo di TriDAS e si connette agli altri tramite una rete locale creata da *compose*. Quindi *compose* prevede comandi per: avviare tutti i servizi, vedere l'output e il *log* di ciascun servizio, aprire una *shell* in qualsiasi *container* per eseguire dei comandi, interrompere l'esecuzione di tutti i servizi. Si riporta come esempio la definizione del servizio `tcpu` (Listato 3.6). Si fa uso di un *dockerfile* esterno per costruire e configurare l'ambiente del container. Viene montato un volume esterno, dove salvare dati di log. Il servizio attende l'avvio dei servizi `hm` e `em` in modo che, quando avviata, la TCPU disponga di un flusso di dati in entrata proveniente dall'HM e possa inviare dati all'EM.

Listato 3.6 Definizione del container `tcpu` all'interno del file di configurazione di *docker compose*.

```
tcpu:
  build:
    context: .
    dockerfile: Dockerfile-tridas
  image: tridas/tridas
  init: true
  depends_on:
    - syslog
  logging:
    driver: syslog
    options:
      syslog-address: "udp://127.0.0.1:5514"
```

```
      tag: "tcpu"
  volumes:
    - /tmp/tridas:/tmp/tridas
  links:
    - hm
    - em
```

3.3 Strumenti di analisi dei file di output

Sono stati sviluppati strumenti di analisi dei PT file in modo da leggerne e osservarne graficamente il contenuto.

3.3.1 Estrazione in formato testuale dei dati dei PT file

Il PT file contiene i dati in formato binario. Per poter analizzare e rendere graficamente i dati, è stato implementato un algoritmo che estrae i dati da un PT file e li scrive su file in formato testuale. Il programma `pt2txt` prende come argomenti da riga di comando il nome del PT file da leggere e il nome del file di testo da scrivere. Quindi, utilizzando la classe `PtFileReader` di TriDAS, scorre le TS del file, gli eventi di ciascuna TS e le *hit* di ciascun evento. Per ciascuna *hit* l'algoritmo scrive una riga del file di testo contente nell'ordine e separandoli con uno spazio: le proprietà della *hit*, le proprietà dell'evento a cui appartiene la hit, le proprietà della TS a cui appartiene l'evento. Nella prima riga del file vengono scritti i nomi identificativi dei vari attributi. In questo modo i valori di un certo attributo occupano tutti una certa colonna, identificata dal nome dell'attributo. Il programma è stato integrato nel sistema di compilazione *CMake* di TriDAS.

Listato 3.7 Parte di un file di testo prodotto da un PT file con `pt2txt`. In ordine sono elencati la carica della *hit*, l'indice identificativo del sensore (PMT) che l'ha prodotta, il tempo di arrivo, l'indice identificativo della "torre" (gruppo di rilevatori) in cui risiede il sensore, l'indice identificativo dell'evento di cui la *hit* fa parte. Gli altri attributi sono omessi dal listato per brevità.

```
Charge PMTID T1ns TowerID EventID
105 4 568 22 0
422 5 568 22 0
```

3.3.2 Graficare dati numerici da file di testo

Un PT file di ~500 MB contiene circa 19×10^6 *hit*. Perciò è utile rendere graficamente i valori delle varie proprietà delle *hit* per osservarne l'andamento. Si è sviluppato un algoritmo che legge dati numerici disposti in colonna e li usa per riempire dei grafici. In particolare, si fa uso del *framework* di analisi dati e grafica ROOT. L'algoritmo può essere eseguito come macro dalla linea di comando di ROOT o essere compilato in un eseguibile. Per la compilazione e l'esecuzione è fornito un ambiente *docker* tramite *dockerfile* di configurazione [11].

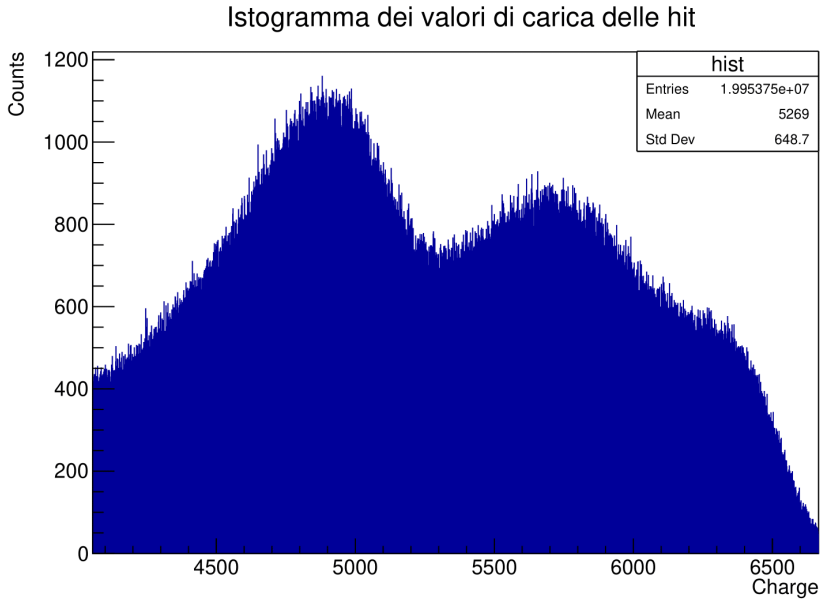

Figura 3.2 Istogramma dei valori di carica delle *hit* generato da un PT file utilizzando `pt2txt` (vedi sottosezione 3.3.1) e `plotxt`. Si nota il picco di carica in corrispondenza dell'energia degli elettroni del fascio.

L'algoritmo prende come argomenti il nome del file contenente i dati, i nomi degli attributi di cui graficare i dati o gli indici delle colonne contenenti i dati da graficare, il tipo di grafico da riempire (tra quelli supportati) e eventuali opzioni da passare alla funzione `Draw` di ROOT. I tipi di grafici supportati sono `TGraph` per disegnare grafici sul piano cartesiano e `TH1I` per istogrammi di numeri interi. Per esempio, per rappresentare graficamente la prima e la terza colonna di un file di testo come coordinate x e y di punti sul piano cartesiano

```
$ ./pltxt -d data.txt -c 0 2 -p TGraph -o AL
```

o in alternativa da linea di comando di ROOT

```
$ pltxt<int>("data.txt", {0, 2}, {"TGraph", "AL"})
```

La funzione `pltxt` riceve il tipo di dato da leggere come argomento *template*. I dati supportati sono `int` e `double`.

Conclusioni

L'algoritmo di identificazione di eventi generati da raggi cosmici implementato nel trigger L2 *TrigFromBeam* è integrato con successo nel *framework* di TriDAS.

Allo stato attuale risultano delle incompatibilità nei sistemi di indici utilizzati dai diversi componenti software. Infatti, i PT file disponibili al momento per l'analisi sono quelli acquisiti ai Jefferson Lab utilizzando il rivelatore di cui nel capitolo 2 (matrice 3×3 di calorimetri). Il formato di dati utilizzato da TriDAS per l'acquisizione prevede una geometria di 23 "torri", ciascuna composta da 5 "piani" di 16 sensori ciascuno, per un totale di 1840 sensori. Al momento dell'acquisizione quindi sono stati assegnati ai soli 9 sensori dell'esperimento degli indici nell'intervallo $1 - 1840$. Inoltre gli indici non sono associati ai sensori secondo la disposizione dell'apparato sperimentale, per cui ad esempio i primi tre indici non corrispondono alla prima riga della matrice del rivelatore. È necessario quindi realizzare ulteriori analisi per rendere noto all'algoritmo l'associazione tra gli indici utilizzati da TriDAS e la posizione dei sensori all'interno della matrice del rivelatore.

La continuazione di questo lavoro prevede la verifica dell'efficienza dell'algoritmo sfruttando il software `trig_sim` descritto nel capitolo 3, utilizzando PT file precedentemente acquisiti e confrontandone il contenuto con quello dei nuovi PT file prodotti da `trig_sim`. A questo scopo possono essere utilizzati i programmi `pt2txt` e `plotxt`, anch'essi descritti nel capitolo 3.

Appendice A

Listato completo dell'algoritmo

Listato A.1 TrigFromBeam.h

```
#ifndef DAQ_TCPU_TRIGGER_PLUGINS_TRIGFROMBEAM_TRIGFROMBEAM_H
#define DAQ_TCPU_TRIGGER_PLUGINS_TRIGFROMBEAM_TRIGFROMBEAM_H

#include "TriggerInterface.h"

namespace tridas {
namespace tcpu {

extern "C" void TrigFromBeam(PluginArgs const& args);

}
} // namespace tridas
#endif
```

Listato A.2 TrigFromBeam.cpp

```
/*  This plugin triggers only if the signal comes from the beam and not from
 * other sources (e.g. noise, cosmic rays). L0 trigger just reports and builds
 * hits. L1 trigger builds events. TimeSlice is TriggeredEvent.
 *
 */
#include "TrigFromBeam.h"
#include "Event_Classes.h"
#include "tridas_dataformat.hpp"// needed for fine_time
#include "Geometry.h"
#include "PMTHit.h"
//#include "f_dataformat_clas12.hpp" //needed for fine_time
#include "log.hpp"
#include <algorithm>
#include <cmath>

namespace tridas {
namespace tcpu {

```

```
19 // Returns the row of the 3x3 matrix to which the PMT belongs. Counting
       starts
20 // at 0. the indexes of the first row (row == 0) are in the range [0,
       columns[
21 // the indexes of the second row (row == 1) are in the range [columns,
22 // 2*columns[ etc... columns should be equal to geometry.pmts_per_floor rows
23 // should be equal to geometry.floors_per_tower
24 int row(PMTHit const& seed, const int columns)
25 {
26   const int seed_PMTID{seed.getAbsPMTID()};
27
28   int row{0};
29   while (seed_PMTID < row * columns || seed_PMTID >= (row + 1) * columns) {
30     ++row;
31   }
32
33   return row;
34 }
35
36 // Returns the column of the 3x3 matrix to which the PMT belongs. Counting
37 // starts at 0. the indexes of the first column (col == 0) are divisible by
       the
38 // number of columns. the indexes of the second column (col == 1) minus 1 are
39 // divisible by the number of columns. etc... columns should be equal to
40 // geometry.pmts_per_floor
41 int col(PMTHit const& seed, const int columns)
42 {
43   const int seed_PMTID{seed.getAbsPMTID()};
44
45   int col{0};
46   while ((seed_PMTID - col) % columns) {
47     ++col;
48   }
49
50   return col;
51 }
52
53 // Two seeds are space correlated if the trace they draw lies at an angle of
       at
54 // most 45 degree to the vertical Also because cosmic rays comes from above,
55 // space correlation requires coherent space AND time order e.g. a seed in
       row 0
56 // must be earlier than a seed in row 1
57 bool spaceCorrelation(PMTHit const& seed, PMTHit const& other_seed,
58                       Geometry const& geom)
59 {
60   const int rows{3}; // geom.floors_per_tower;
61   const int cols{3}; // geom.pmts_per_floor;
62   const int seed_PMTID{seed.getAbsPMTID()};
63   assert(0 <= seed_PMTID && seed_PMTID < cols * rows && "PMTID not possible");
```

```
64
65    const int seed_row{row(seed, cols)};
66    const int other_seed_row{row(other_seed, cols)};
67    const int seed_col{col(seed, cols)};
68    const int other_seed_col{col(other_seed, cols)};
69
70    // If seed is from a PMT lower (higher) than other_seed, the row index is
71    // bigger (smaller) and seed must be older (earlier) Also if the sum and
72    // subtraction of row and col index of other_seed are less (greater) or
          equal
73    // to the ones of seed then the two seeds draw an angle to the vertical
74    // smaller than 45 degree.
75    if (seed_row > other_seed_row
76        && seed.IsOlderThan(other_seed)) // space and time order respected
77      return other_seed_row + other_seed_col <= seed_row + seed_col
78          && other_seed_row - other_seed_col <= seed_row - seed_col;
79
80    if (seed_row < other_seed_row
81        && other_seed.IsOlderThan(seed)) // space and time order respected
82      return other_seed_row + other_seed_col >= seed_row + seed_col
83          && other_seed_row - other_seed_col >= seed_row - seed_col;
84
85    return false; // here if other_seed_row == seed_row
86  }
87
88  // Two seeds are time correlated if the time between them is equal to the
        time
89  // spent by the cosmic ray to cover the distance between to PMTs
90  bool timeCorrelation(PMTHit const& seed, PMTHit const& other_seed,
91                       Geometry const& geom)
92  {
93    assert(geom.positions.size() >= 9
94           && "Not correct number of PMTs to evaluate time correlation of
                 seeds");
95    // The side of the square face of each of the 9 scintillators arranged in
          the
96    // 3x3 matrix i.e. the distance between a PMT and the nearest one in
          orizontal
97    // and vertical directions
98    const double front_cell_side{
99        std::abs(geom.positions[0].x - geom.positions[1].x)};
100   // The diagonal of the square face of each of the 9 scintillators arranged
101   // i.e. the distance between a PMT and the nearest one in diagonal direction
102   const double front_cell_diagonal{std::sqrt(2) * front_cell_side};
103   const double light_speed{2.99792458e8};      // m/s
104   const double low_cosmic_speed{0.9 * light_speed}; // m/s
105
106   // Cosmic rays can come from right above or diagonal and in range of speeds
          of
107   // about [0.9c, c] So the shortest time between two seeds is given by a
```

```
    cosmic
  // ray that comes from right above at the speed of light The shortest path
  // possible between two seeds is the side of a single cell The longest time
  // between two seeds is given by a comsic ray coming diagonal at the lowest
  // speed possible. In a 3x3 matrix the longest path possible is going
      through
  // two cells in diagonal directions
  fine_time shortest_correlation_time_frame{
      static_cast<int>(std::round(front_cell_side / light_speed))};
  fine_time longest_correlation_time_frame{
      static_cast<int>(std::round(2 * front_cell_diagonal /
          low_cosmic_speed))};
  // Absolute value of the subtraction
  fine_time real_time_frame{
      seed.IsOlderThan(other_seed)
          ? seed.get_fine_time() - other_seed.get_fine_time()
          : other_seed.get_fine_time() - seed.get_fine_time()};

  return shortest_correlation_time_frame < real_time_frame
      && real_time_frame < longest_correlation_time_frame;
}

// Returns the energy of the particle that generated a seed.
// The hits +/- 8 ns from seed belongs to the same particle
// TODO: make half_time_frame a function parameter
double getParticleEnergy(PMTHit const& seed, TriggeredEvent const& tev)
{
  fine_time seed_time{seed.get_fine_time()};
  fine_time half_time_frame{8};

  double particle_energy{seed.getCaliCharge()};

  PMTHit const* this_hit = seed.previous();
  // Go back in time from seed to find the previous hits that fall in a 8 ns
  // time frame
  for (PMTHit const* start_hit = reinterpret_cast<PMTHit const*>(tev.sw_hit_);
       this_hit != start_hit
       && seed_time - this_hit->get_fine_time() <= half_time_frame;
       this_hit = this_hit->previous()) {
    particle_energy += this_hit->getCaliCharge();
  }

  this_hit = seed.next();
  // Go forward in time from seed to find the next hits that fall in a 8 ns
      time
  // frame
  for (PMTHit const* end_hit = reinterpret_cast<PMTHit const*>(tev.ew_hit_);
       this_hit != end_hit
       && this_hit->get_fine_time() - seed_time <= half_time_frame;
       this_hit = this_hit->next()) {
```

```
154     particle_energy += this_hit->getCaliCharge();
155   }
156
157   return particle_energy;
158 }
159
160 void TrigFromBeam(PluginArgs const& args)
161 {
162   const int id       = args.id;
163   EventCollector& evc = *args.evc;
164   Geometry const& geom = *args.geom;
165
166   // Initialise the logger
167   tridas::Log::init("TrigFromBeam", tridas::Log::INFO);
168
169
170   unsigned plug_events{0};
171
172   // for each event
173   for (int i{0}; i < evc.used_trig_events(); ++i) {
174     TriggeredEvent& tev = *(evc.trig_event(i));
175     assert(tev.nseeds_[L1TOTAL_ID] && "Triggered event with no seeds");
176
177     TRIDAS_LOG(INFO) << "In triggered_event " << tev.EventID_ << " there are
          " << tev.nHit_ << " hits.";
178
179     // search for seeds, i.e. PMTHits of the TriggeredEvent that triggered the
180     // L1
181     std::vector<PMTHit const*> seeds;
182
183     PMTHit const* end_hit = reinterpret_cast<PMTHit const*>(tev.ew_hit_);
184     for (PMTHit const* this_hit = tev.L1_first_seed_; this_hit != end_hit;
185          this_hit             = this_hit->next()) {
186       assert(this_hit && "invalid current hit address");
187       assert(end_hit && "invalid end hit address");
188
189       TRIDAS_LOG(INFO) << "Hit ";
190       if (this_hit->isSeed()) {
191         seeds.push_back(this_hit);
192         TRIDAS_LOG(INFO) << "[SEED] ";
193       }
194       TRIDAS_LOG(INFO) << *this_hit;
195     }
196     assert(!seeds.empty() && "Seeds not found");
197
198
199     // Check if this_seed is from a cosmic ray, i.e. if there are other seeds
200     // (~same energy) spacetime correlated to this_seed in other PMTs.
201     for (auto const& this_seed : seeds) {
202       std::vector<PMTHit const*> cosmic_trace;
```

```
203       cosmic_trace.reserve(seeds.size());
204
205       std::copy_if(seeds.begin(), seeds.end(),
              std::back_inserter(cosmic_trace),
206                   [&geom, &this_seed](auto const& other_seed) {
207                     return this_seed != other_seed
208                         && spaceCorrelation(*this_seed, *other_seed, geom)
209                         && timeCorrelation(*this_seed, *other_seed, geom);
210                   });
211
212       /* Check that are all correlated together in a coherent way to avoid
213        * accidental correlations e.g. if the first two are from the same
              column
214        * then the third must be from the same column too. However with only 3
215        * columns it would be too difficult to say which is the real one coming
216        * from the cosmic ray.
217        */
218
219       if (!cosmic_trace.empty()) { // seed is from cosmic ray
220         // Add the original seed to fill the trace
221         cosmic_trace.push_back(this_seed);
222
223         double cosmic_energy{0.};
224         TRIDAS_LOG(INFO) << "Detected cosmic ray:";
225         for (auto const& cosmic_seed : cosmic_trace) {
226           TRIDAS_LOG(INFO)
227               << "\nin PMT[" << cosmic_seed->getAbsPMTID() << "] at time "
228               << cosmic_seed->get_fine_time() << " with energy "
229               << cosmic_seed->getCaliCharge() << '\n';
230           cosmic_energy += cosmic_seed->getCaliCharge();
231         }
232         TRIDAS_LOG(INFO) << "\nTotal energy is " << cosmic_energy;
233
234
235       } else { // seed is from beam
236
237         TRIDAS_LOG(INFO) << "Detected particle in PMT["
238                          << this_seed->getAbsPMTID() << "] at time "
239                          << this_seed->get_fine_time() << " with energy "
240                          << getParticleEnergy(*this_seed, tev) << '\n';
241
242         // increment seeds counter of triggered event
243         ++tev.plugin_nseeds_[id];
244         tev.plugin_ok_ = true;
245       }
246     }
247
248     if (tev.plugin_ok_) {
249       ++plug_events;
250     }
```

```
251   }
252
253   // Set plugin results
254   evc.set_stats_for_plugin(id, plug_events);
255   TRIDAS_LOG(INFO) << "TrigFromBeam triggered " << evc.stats_for_plugin(id)
256                    << " of " << evc.used_trig_events() << " events in TTS "
257                    << evc.ts_id();
258 }
259
260 } // namespace tcpu
261 } // namespace tridas
```

Bibliografia

[1] P. Abratenko et al. “Cosmic Ray Background Rejection with Wire-Cell LArTPC Event Reconstruction in the MicroBooNE Detector”. In: *Phys. Rev. Applied* 15 (6 giu. 2021), p. 064071. DOI: 10.1103/PhysRevApplied.15.064071. URL: https://link.aps.org/doi/10.1103/PhysRevApplied.15.064071.

[2] A. Acker et al. “The CLAS12 Forward Tagger”. In: *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment* 959 (2020). ISSN: 0168-9002. DOI: 10.1016/j.nima.2020.163475. URL: https://www.sciencedirect.com/science/article/pii/S0168900220300838.

[3] Fabrizio Ameli et al. “Streaming readout for next generation electron scattering experiments”. In: *The European Physical Journal Plus* 137.958 (2022). DOI: 10.1140/epjp/s13360-022-03146-z.

[4] Alessandro Bettini. *Introduction to Elementary Particle Physics*. Second. Cambridge University Press, 2014. ISBN: 978-1-107-05040-2.

[5] Angela Bracco. “Al cuore della materia. Nel nucleo l’origine della varietà degli elementi.” In: *Asimmetrie* 9 (2009). URL: https://www.asimmetrie.it/al-cuore-della-materia.

[6] Sylvie Braibant, Giorgio Giacomelli e Maurizio Spurio. *Particles and Fundamental Interactions. An introduction to particle physics*. Second. Springer, 2012. DOI: 10.1007/978-94-007-2464-8.

[7] T. Chiarusi et al. “The Trigger and Data Acquisition System for the KM3NeT-Italy neutrino telescope”. In: *Journal of Physics: Conference Series* 898.Track 1: Online Computing (2017). DOI: 10.1088/1742-6596/898/3/032042.

[8] Tommaso Chiarusi. “The NEMO trigger and data acquisition system”. In: *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment* 725 (2013), pp. 129–132. ISSN: 0168-9002. DOI: 10.1016/j.nima.2012.11.168. URL: https://www.sciencedirect.com/science/article/pii/S016890021201515X.

[9] KM3NeT collaboration. *KM3Net website*. URL: https://www.km3net.org/.

[10] TriDAS developers. *TriDAS-core official repository*. URL: https://baltig.infn.it/tridas/tridas-core.

[11] Antonio Ghinassi. *Plotxt official repository*. URL: https://github.com/ghinanto/plotxt.

[12] Kenneth S. Krane. *Introductory Nuclear Physics*. John Wiley & Sons, 1988. ISBN: 0-471-80553-X.

[13] Paolo Lipari. “Voci dell’universo”. In: *Asimmetrie* 10 (2010). URL: https://www.asimmetrie.it/voci-dell-universo.

[14] Pia De Simone. “Plato’s use of the term stoicheion. Origin and implications”. In: *Archai* 30 (2020). DOI: 10.14195/1984-249X_30_5.