Università degli Studi di Pisa

FACOLTÀ DI SCIENZE MATEMATICHE FISICHE E NATURALI
Corso di Laurea Magistrale in Fisica Delle Interazioni Fondamentali

Tesi di laurea Magistrale

Algoritmi di Trigger Paralleli per la Ricerca di Decadimenti Rari del Mesone K

Candidato
Jacopo Pinzino

Relatore
Chiar.mo Prof. M. Sozzi

Anno Accademico 2010-2011

INDICE

ELENCO DELLE FIGURE

ELENCO DELLE TABELLE

INTRODUZIONE

Il lavoro di tesi in questione è stato svolto nell'ambito della collaborazione per l'esperimento NA62 presso i laboratori del CERN di Ginevra. L'obiettivo finale di questa tesi ha riguardato la creazione di un algoritmo di trigger per le camere a *straw* dell'esperimento NA62 e la possibilità di una sua implementazione su una GPU (scheda grafica).

La tesi presenta all'inizio (primo capitolo) un introduzione riguardante l'esperimento NA62 in cui viene brevemente descritto il suo obiettivo, la misura del *branching ratio* del decadimento ultra raro $K^+ \to \pi^+\nu\bar{\nu}$ con un rapporto segnale su fondo pari a 10 per ottenere un test stringente sul Modello Standard delle particelle elementari: sono esposti cenni della fisica del decadimento con le motivazioni teoriche che hanno portato alla sua scelta e le strategie sperimentali che verranno utilizzate nella reiezione dei fondi e nella costruzione dei rivelatori.

Nel secondo capitolo viene poi descritto l'apparato sperimentale, suddiviso in una parte a monte della regione di decadimento, in cui sono utilizzati i rivelatori per identificare i K^+ all'interno del fascio e misurarne il tempo, la direzione e l'impulso, ed un parte a valle della regione di decadimento. In tale regione è presente un sistema di rivelatori di veto per i fotoni che permette una copertura angolare totale per angoli inferiori a 50 mrad, un rivelatore Čerenkov, un sistema di veto per i muoni, un odoscopio carico ed uno spettrometro magnetico composto da camere a deriva di tipo STRAW, che sarà descritto con particolare attenzione in quanto direttamente coinvolto nel lavoro di tesi.

Successivamente, nel terzo capitolo, vengono descritte le GPU, le loro modalità di programmazione, il linguaggio usato (OpenCL) e i vantaggi del loro utilizzo al posto di *trigger hardware* (basati solitamente su FPGA) come i loro minori costi, la maggiore flessibilità e la loro comodità ai fini delle manutenzioni e riprogrammazioni.

Nel quarto capitolo viene descritto il lavoro da me svolto per l'ottimizzazione di un algoritmo di *trigger* di livello 0 per la ricostruzione di singoli cerchi nel RICH per una GPU in modo da ridurre al minimo il suo tempo di esecuzione.

Infine nell'ultima parte della tesi, il quindo capitolo, viene presentato lo studio, la simulazione e l'implementazione di un possibile algoritmo di *trigger* per le camere a STRAW con lo scopo di rigettare gli eventi di fondo $K^+ \to \pi^+\pi^+\pi^-$ perdendo una minima quantità di segnale. Lo studio dell'algoritmo di *trigger* si è sviluppato in due parti: nella prima si è considerato ogni evento di segnale $K^+ \to \pi^+\nu\bar{\nu}$ e di fondo $K^+ \to \pi^+\pi^+\pi^-$ singolarmente, mentre nella seconda parte si è considerato l'effetto del *pileup*, ovvero della sovrapposizione di eventi avvenuti a tempi diversi all'interno del tempo risolutivo del rivelatore. Dopo aver esposto i risultati dell'algoritmo simulato vengono descritti i cambiamenti apportati in seguito alla sua parallelizzazione e implementazione sulla GPU. I risultati dell'algoritmo e il suo tempo di esecuzione sulla GPU mostrano l'utilità dell'utilizzo di questo algoritmo e la possibilità dell'utilizzo delle GPU nel trigger dell'esperimento.

CAPITOLO 1

OBIETTIVI FISICI DELL'ESPERIMENTO NA62

Indice

Il decadimento ultra raro $K^+ \to \pi^+\nu\bar{\nu}$ è un processo di grande utilità per lo studio della fisica del *flavour* e per ottenere un test decisivo del Modello Standard. Questo decadimento, insieme a $K_L \to \pi^0\nu\bar{\nu}$, è importante perché il suo *branching ratio* può essere calcolato con un alto grado di precisione usando il Modello Standard [17, 30].

$$BR\left(K^+ \to \pi^+\nu\bar{\nu}\right)(SM) = (8.5 \pm 0.7) \times 10^{-11} \tag{1.1}$$

L'incertezza nel *branching ratio* è dominata dalla precisione con cui sono conosciuti i parametri della matrice CKM piuttosto che da errori teorici che sono attualmente dell'ordine di 2 %. Per questo motivo l'esperimento NA62 [3] al CERN SPS (fig. 1.1) mira a raccogliere dell'ordine di 100 eventi di $K^+ \to \pi^+\nu\bar{\nu}$ in ~ 2 anni di presa dati con un rapporto segnale su fondo (S/B) di circa 10:1 in modo da tenere bassa l'incertezza sistematica totale, per ottenere una precisione del 10% sulla misura del *branching ratio*. Con un *branching ratio* dell'ordine di 10^{-10} dovranno essere prodotti almeno 10^{12} decadimenti, quindi per ottenere il rapporto S/B 10:1 è necessario un fattore di reiezione dell'ordine di 10^{11} sui decadimenti generici del K^+ che hanno un *branching ratio* dell'ordine di 10^{-1}.

1.1 Introduzione Teorica

Tra i molti decadimenti del K e del B, i decadimenti ultra-rari soppressi dal meccanismo GIM quadratico $K^+ \to \pi^+\nu\bar{\nu}$ e $K_L \to \pi^0\nu\bar{\nu}$ sono molto speciali perché il loro *branching ratio* può essere calcolato ad un alto grado di precisione, a differenza degli altri processi a corrente neutra con scambio di sapore (FCNC: *Flavour-Changing Neutral-Current*). I decadimenti

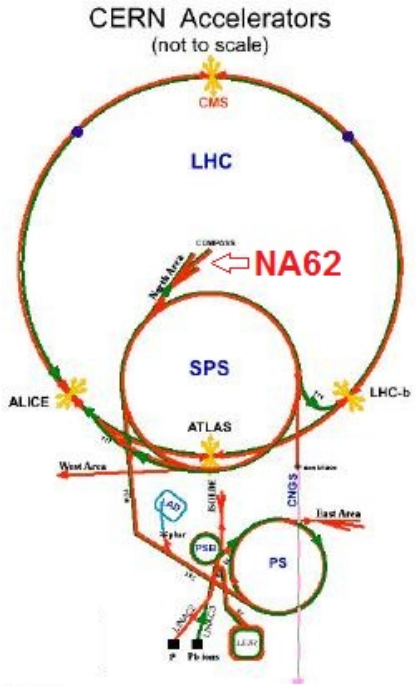


Figura 1.1: Il complesso degli acceleratori del CERN con in evidenza la posizione di NA62.

come $B \to X_s\mu^+\mu^-$ e $B_s \to \mu^+\mu^-$ permettono il calcolo del loro *branching ratio* con un incertezza teorica superiore al 10% [18], mentre il *branching ratio* di $K_L \to \pi^0\nu\bar{\nu}$ può essere calcolato con un incertezza tra $1-2\%$ [15, 17, 30]. Nel caso del decadimento $K^+ \to \pi^+\nu\bar{\nu}$, la presenza del contributo del *charm* all'interno del diagramma Z^0 pinguino e del diagramma a box (fig. 1.2) comporta un incertezza perturbativa teorica di $\sim 7\%$ al livello NLO [16, 17] che successivamente è stata ridotta al $1-2\%$ usando un calcolo al NNLO [19, 20].

La precisione teorica dei *branching ratio* di questi due decadimenti è dovuta al fatto che gli elementi di matrice adronici, che in genere costituiscono la più importante fonte di incertezza, possono essere ottenuti dalle misure sperimentali del decadimento $K^+ \to \pi^0 e^+\nu$ con l'inclusione delle correzioni di violazione dell'*isospin*. Ulteriori correzioni dovute a contributi a lunga distanza, insieme ad effetti elettrodeboli di ordine superiore, possono essere trascurate.

Con la combinazione delle misure dei *branching ratio* di questi due decadimenti è possibile determinare l'angolo β del triangolo di unitarietà o in modo equivalente determinare la fase dell'elemento V_{td} della matrice CKM (Cabibbo-Kobayashi-Maskawa). Il confronto tra diverse misure dell'angolo β permette di avere un potente test sul Modello Standard e sull'unitarietà della matrice CKM.

1.1.1 La Matrice CKM

La matrice CKM (V_{CKM}) descrive il mescolamento dei quark nelle interazioni deboli del Modello Standard, infatti lega gli autostati di sapore d, s, b a quelli che diagonalizzano l'hamiltoniana debole d', s', b' (eq. 1.2).

$$\begin{pmatrix} d' \\ s' \\ b' \end{pmatrix} = \begin{pmatrix} V_{ud} & V_{us} & V_{ub} \\ V_{cd} & V_{cs} & V_{cb} \\ V_{td} & V_{ts} & V_{tb} \end{pmatrix} \begin{pmatrix} d \\ s \\ b \end{pmatrix} \tag{1.2}$$

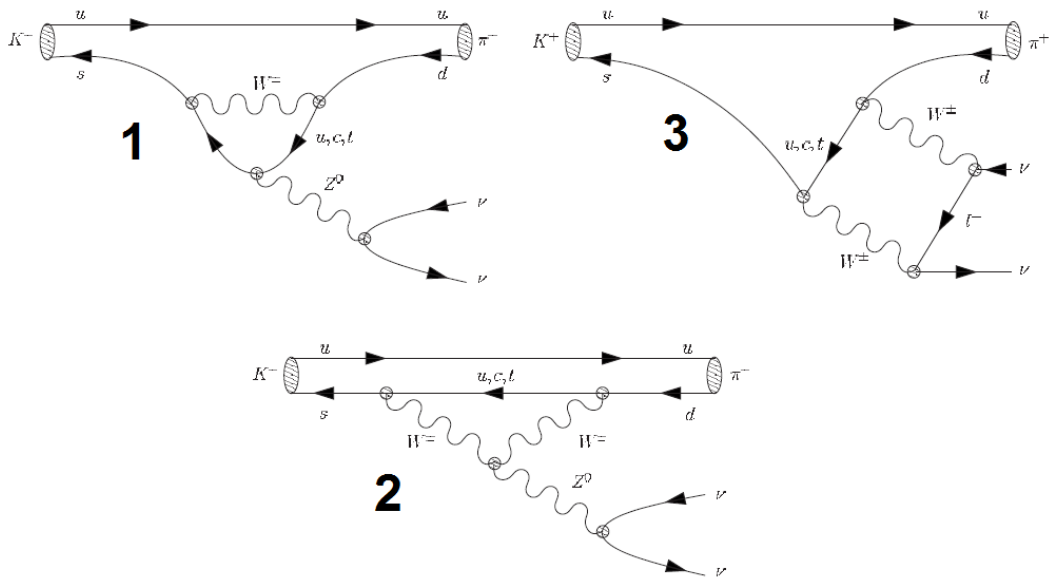


Figura 1.2: Diagrammi di Feynman Z pinguino (1-2) e a box (3) per il processo $K^+ \to \pi^+ \nu\bar{\nu}$.

Il meccanismo GIM (Glashow-Iliopoulos-Maiani) [27] impone la diagonalità della matrice $V_{CKM}^{\dagger} V_{CKM}$ in modo che gli autostati dell'interazione debole siano ortogonali come gli autostati di massa.

$$\text{Sia } D \equiv (d, s, b) \text{ e } D^{'} \equiv (d', s', b')$$

$$\bar{D}_i^{'} \gamma^{\mu}(1-\gamma_5) D_j^{'} = \bar{D}_i \gamma^{\mu}(1-\gamma_5) \left(V_{CKM}^{\dagger} V_{CKM} \right)_{ij} D_j = 0 \text{ se } i \neq j \Leftrightarrow V_{CKM}^{\dagger} V_{CKM} = \begin{pmatrix} \lambda_1 & 0 & 0 \\ 0 & \lambda_2 & 0 \\ 0 & 0 & \lambda_3 \end{pmatrix} \tag{1.3}$$

La diagonalità della matrice $V_{CKM}^{\dagger} V_{CKM}$ implica anche la soppressione dei processi FCNC rispetto ai processi a correnti cariche, in quanto proibisce contributi da diagrammi ad albero per essi.

Un'ulteriore condizione sulla matrice $V_{CKM}^{\dagger} V_{CKM}$ è aggiunta dall'universalità delle interazioni deboli, comportando che gli elementi sulla diagonale siano tutti uguali ad 1 ($\lambda_1 = \lambda_2 = \lambda_3 = 1$) e quindi che la matrice sia uguale all'identità.

$$V_{CKM}^{\dagger} V_{CKM} = I_3 \tag{1.4}$$

Quindi la matrice CKM (V_{CKM}) risulta essere unitaria.

Le matrici complesse $n \times n$ hanno $2n^2$ parametri ma imponendo le condizione di unitarietà, composte da n^2 equazioni, restano n^2 parametri. Le matrici unitarie possono essere considerate come il prodotto di una matrice ortogonale reale O per una matrice complessa A: visto che le matrici ortogonali reali hanno $\frac{n(n-1)}{2}$ parametri reali indipendenti, alla matrice A restano $n^2 - \frac{n(n-1)}{2} = \frac{n(n+1)}{2}$ parametri complessi. Il numero delle fasi indipendenti della matrice complessa può essere ridotto di $2n-1$ con una ridefinizione delle fasi arbitrarie dei quark, lasciando infine $\frac{(n-1)(n-2)}{2}$ parametri. Riassumendo, una matrice unitaria $n \times n$ possiede $\frac{n(n-1)}{2}$ parametri reali indipendenti e $\frac{(n-1)(n-2)}{2}$ fasi complesse indipendenti. Nel caso della matrice CKM con 3 famiglie di quark, $n = 3$ e abbiamo 3 parametri reali ed 1 fase complessa, quest'ultima è responsabile della violazione della simmetria CP.

In seguito sarà utile utilizzare la parametrizzazione di Wolfenstein della matrice CKM [38]. Visto che l'elemento di matrice V_{us} è ben determinato ($|V_{us}| \approx 0.22$) e che la matrice

CKM sembra differire dall'unità solo per piccole quantità, Wolfestein decise di svilupparla in serie di potenze dell'elemento $\lambda = V_{us}$. All'ordine λ^2 dello sviluppo si ottiene la matrice (eq. 1.5) in cui i restanti elementi sono stati scelti in base all'ordine di grandezza stimato sperimentalmente come ad esempio $V_{cb} = A\lambda^2$ con $A \approx \frac{5}{4}$ è determinato dalla misura della vita media del mesone B.

$$V_{CKM} = \begin{pmatrix} 1-\frac{1}{2}\lambda^2 & \lambda & 0 \\ -\lambda & 1-\frac{1}{2}\lambda^2 & A\lambda^2 \\ 0 & -A\lambda^2 & 1 \end{pmatrix} + O(\lambda^3) \tag{1.5}$$

All'ordine λ^3 dello sviluppo nella matrice (eq. 1.6) devono essere introdotti due nuovi parametri reali $\rho \approx 0.135$ e $\eta \approx 0.349$.

$$V_{CKM} = \begin{pmatrix} 1-\frac{1}{2}\lambda^2 & \lambda & A\lambda^3(\rho - i\eta) \\ -\lambda & 1-\frac{1}{2}\lambda^2 & A\lambda^2 \\ A\lambda^3(1-\rho-i\eta) & -A\lambda^2 & 1 \end{pmatrix} + O(\lambda^4) \tag{1.6}$$

In questa parametrizzazione la violazione di CP viene descritta dal parametro η. Per una accuratezza maggiore è necessario aggiungere alla matrice (eq. 1.6) i seguenti termini dello sviluppo agli ordini successivi [10]:

$$V_{ud} = 1 - \frac{1}{2}\lambda^2 - \frac{1}{8}\lambda^4 + O(\lambda^6) \tag{1.7}$$

$$V_{us} = \lambda + O(\lambda^7) \tag{1.8}$$

$$V_{ub} = A\lambda^3(\rho - i\eta) \tag{1.9}$$

$$V_{cd} = -\lambda + \frac{1}{2}A^2\lambda^5[1 - 2(\rho + i\eta)] + O(\lambda^7) \tag{1.10}$$

$$V_{cs} = 1 - \frac{1}{2}\lambda^2 - \frac{1}{8}\lambda^4(1 + 4A^2) + O(\lambda^6) \tag{1.11}$$

$$V_{cb} = A\lambda^2 + O(\lambda^8) \tag{1.12}$$

$$V_{td} = A\lambda^3[1 - (\rho + i\eta)(1 - \frac{1}{2}\lambda^2)]O(\lambda^7) \tag{1.13}$$

$$V_{ts} = -A\lambda^2 + \frac{1}{2}A\lambda^4[1 - 2(\rho + i\eta)] + O(\lambda^6) \tag{1.14}$$

$$V_{tb} = 1 - \frac{1}{2}A^2\lambda^4 + O(\lambda^6) \tag{1.15}$$

1.1.2 I Triangoli Di Unitarietà

La condizione di unitarietà applicata sulla matrice CKM dà origine a 9 equazioni (eq. 1.4), tre per gli elementi lungo la diagonale e sei per gli elementi fuori diagonale, di cui solo tre sono indipendenti.

$$|V_{uq}|^2 + |V_{cq}|^2 + |V_{tq}|^2 = 1 \text{ per } q = d, s, b \tag{1.16}$$

$$V_{us}V_{ud}^* + V_{cs}V_{cd}^* + V_{ts}V_{td}^* = 0 \tag{1.17}$$

$$V_{ub}V_{us}^* + V_{cb}V_{cs}^* + V_{tb}V_{ts}^* = 0 \tag{1.18}$$

$$V_{ud}V_{ub}^* + V_{cd}V_{cb}^* + V_{td}V_{tb}^* = 0 \tag{1.19}$$

Le equazioni degli elementi fuori diagonale possono essere rappresentate ciascuna come un triangolo nel piano complesso in cui i vettori, che individuano i vertici, corrispondono agli addendi delle equazioni. Utilizzando gli elementi della matrice CKM mostrati nelle equazioni (1.7-1.15) per studiare gli ordini di grandezza degli addendi delle equazioni dei triangoli di unitarietà, si nota che i triangoli corrispondenti alle prime due equazioni (eq. 1.17 e 1.18) sono degeneri. Gli addendi della terza equazione (eq. 1.19) sono tutti dello

stesso ordine di grandezza ($O(\lambda^3)$) rendendoli più adatti per essere sottoposti ad una verifica sperimentale tramite una misura di tutti e tre gli addendi; gli altri triangoli invece rendono più complicata la misura sperimentale di un lato. Quindi utilizzando le equazioni (1.7-1.15) si ricavano gli addendi che compongono l'equazione del terzo triangolo (eq. 1.19).

$$V_{ud}V_{ub}^* = A\lambda^3(\bar{\rho} + i\bar{\eta}) + O(\lambda^7) \tag{1.20}$$

$$V_{cd}V_{cb}^* = -A\lambda^3 + O(\lambda^7) \tag{1.21}$$

$$V_{td}V_{tb}^* = A\lambda^3[1 - (\bar{\rho} + i\bar{\eta})] + O(\lambda^7) \tag{1.22}$$

In queste equazioni è stata usata la notazione (eq. 1.23) in modo da avere espressioni più semplici per gli addendi.

$$\bar{\rho} = \rho(1 - \frac{1}{2}\lambda^2), \quad \bar{\eta} = \eta(1 - \frac{1}{2}\lambda^2) \tag{1.23}$$

Visto che il termine $V_{cd}V_{cb}^*$ è reale all'ordine $O(\lambda^7)$, è possibile riscalare gli altri addendi in modo da ottenere un triangolo con base sull'asse reale e di valore unitario (fig. 1.3).

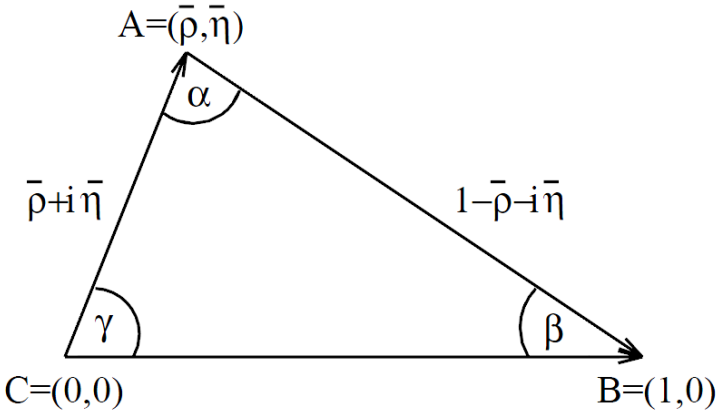


Figura 1.3: Triangolo di unitarietà corrispondente all'equazione 1.19.

L'angolo β del triangolo (fig. 1.3) è stato determinato accuratamente negli esperimenti alle *B-factory* dalla violazione di CP nel decadimento $B \to \psi K^0$ mentre la lunghezza del lato destro con vertici $(\bar{\rho}, \bar{\eta})$ e (1,0) è stata determinata attraverso l'analisi delle oscillazioni $B^0\bar{B}^0$ [14]. La misura del *branching ratio* del decadimento $K^+ \to \pi^+\nu\bar{\nu}$ può offrire una valida alternativa alla misura delle oscillazioni $B^0\bar{B}^0$ con un minore errore teorico, quindi comparando le due misure è possibile ottenere un test significativo sul Modello Standard.

1.1.3 $K^+ \to \pi^+\nu\bar{\nu}$ e il triangolo di unitarietà

Il *branching ratio* del decadimento $K^+ \to \pi^+\nu\bar{\nu}$ si può ricavare dall'equazione (eq. 1.24) [17], in cui G_l è la costante di accoppiamento efficace che mostra i contributi dovuti al *top* e al *charm* ed il termine $(\bar{s}d)_{V-A}(\bar{\nu}_l\nu_l)_{V-A}$ descrive le linee fermioniche esterne.

$$B(K^+ \to \pi^+\nu\bar{\nu}) = \sum_{l=e,\mu,\tau} \frac{|G_l|^2}{4m_K\Gamma_{tot}} \int |(\bar{s}d)_{V-A}(\bar{\nu}_l\nu_l)_{V-A}|^2 \, d\Phi \tag{1.24}$$

$$G_l = \frac{\alpha G_F}{2\pi \sin^2\vartheta_W} [V_{ts}V_{td}^* X_t + V_{cs}V_{cd}^* X_c] \tag{1.25}$$

$$X_t = 1.51 \pm 0.05, \quad X_c = \lambda^4 P_0(X) \quad \text{e} \quad P_0(X) = 0.42 \pm 0.06 \tag{1.26}$$

$$(\bar{s}d)_{V-A}(\bar{\nu}_l\nu_l)_{V-A} \equiv \left\langle \pi^+\nu\bar{\nu} \right| \bar{s}\gamma_\mu(1-\gamma_5)d\bar{\nu}_l\gamma^\mu(1-\gamma_5)\nu_l \left| K^+ \right\rangle \tag{1.27}$$

Nelle equazioni m_K è la massa del K^+, Γ_{tot} è la sua larghezza totale di decadimento, G_F è la costante di Fermi, α è la costante di struttura fine, ϑ_W è l'angolo di Weinberg e Φ è lo spazio delle fasi. L'elemento di matrice adronico (eq. 1.27) è in generale la più grande fonte di incertezza teorica nei decadimenti deboli dei mesoni e non può essere calcolato con precisione perché la QCD a bassa energia non è perturbativa. In questo caso, pero, l'elemento $\langle(\bar{s}d)_{V-A}\rangle$ può essere ottenuto dalla misura del *branching ratio* del decadimento $K^+ \to \pi^0 e^+ \nu$, che ha un errore del 0.79% [32], e si riflette in una relazione tra i *branching ratio* dei due decadimenti (eq. 1.28) dove $r_{K^+} = 0.901$ è un parametro che tiene conto di tutte le correzioni dovute alla rottura della simmetria dell'isospin.

$$B(K^+ \to \pi^+ \nu\bar{\nu}) = 6 r_{K^+} \frac{|G_l|^2}{G_F^2 |V_{us}|^2} B(K^+ \to \pi^0 e^+ \nu) \tag{1.28}$$

Scrivendo gli elementi della matrice CKM nella parametrizzazione di Wolfenstein (eq. 1.7-1.15) si ottiene l'equazione(eq. 1.29).

$$\frac{B(K^+ \to \pi^+ \nu\bar{\nu})}{B(K^+ \to \pi^0 e^+ \nu)} \frac{G_F^2}{6 r_{K^+}} \simeq \left(\frac{\alpha G_F}{2\pi \sin^2 \vartheta}\right)^2 A^4 \lambda^8 X_t^2 \left[(\rho_0 - \bar{\rho})^2 + \bar{\eta}^2\right] \tag{1.29}$$

$$\rho_0 \overset{def}{=} 1 + \frac{X_c}{\lambda^4 A^2 X_t} \tag{1.30}$$

Dall'equazione (1.29) si nota che il *branching ratio* del decadimento $K^+ \to \pi^+ \nu\bar{\nu}$ è proporzionale al quadrato dell'ipotenusa (fig. 1.4) di un triangolo rettangolo che ha un cateto uguale all'altezza $\bar{\eta}$ del triangolo di unitarietà e l'altro pari a $\rho_0 - \bar{\rho}$, quindi supera il vertice (1,0) del triangolo di unitarietà di $\frac{X_c}{\lambda^4 A^2 X_t}$. Tale quantità è dovuta al contributo del *quark charm* al decadimento.

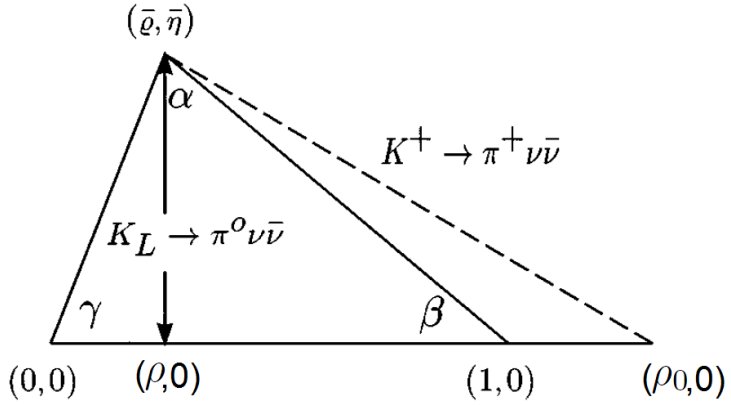


Figura 1.4: Triangolo di unitarietà corrispondente all'equazione 1.19.

Con lo stesso procedimento per il decadimento neutro $K_L \to \pi^0 \nu\bar{\nu}$ si ottiene l'equazione (eq. 1.31).

$$\frac{B(K_L \to \pi^0 \nu\bar{\nu})}{B(K^+ \to \pi^0 e^+ \nu)} \frac{G_F^2}{6 r_{K^+}} \frac{\tau_{K^+}}{\tau_{K_L}} \simeq \left(\frac{\alpha G_F}{2\pi \sin^2 \vartheta}\right)^2 A^4 \lambda^8 X_t^2 \bar{\eta}^2 \tag{1.31}$$

Dall'equazione si nota che il *branching ratio* del decadimento $K_L \to \pi^0 \nu\bar{\nu}$ è proporzionale all'altezza $\bar{\eta}$ del triangolo di unitarietà (fig. 1.4) e che non sono presenti contributi dovuti al *quark charm*. L'assenza di tali contributi permette una maggiore precisione teorica ed inoltre, visto che il *branching ratio* è proporzionale a $\bar{\eta}$, il decadimento non esisterebbe in assenza di violazione di CP quando il triangolo di unitarietà sarebbe degenere.

1.2 Misure Esistenti

Le misure più precise (eq. 1.32) di questo decadimento sono state ottenute dall'esperimento E787 e dal suo aggiornamento E949 al *Brookhaven National Laboratory* basate su sette eventi interpretati come segnale[7].

$$BR\left(K^+ \to \pi^+ \nu\bar{\nu}\right)(E787 + E949) = \left(17.3^{+11.5}_{-10.5}\right) \cdot 10^{-11} \tag{1.32}$$

Questi esperimenti utilizzavano K^+ prodotti da fasci di protoni a bassa energia ($p = 21.5$ GeV$/c$), che decadevano *a riposo*.

Erano considerati segnale $K^+ \to \pi^+ \nu\bar{\nu}$ gli eventi che presentavano solo una traccia carica che veniva identificata come un pione dallo studio della sua catena di decadimento ($\pi^+ \to \mu^+ \to e^+$) all'interno dell'apparato sperimentale. Inoltre gli eventi candidati venivano studiati sul piano (E_+, r_+) dove E_+ è l'energia cinetica della particella carica rivelata e considerata un π^+ mentre r_+ è il range della sua traccia. Su questo piano sono state definite due regioni fiduciali, all'interno delle quali gli eventi trovati sono stati considerati come eventi di segnale.

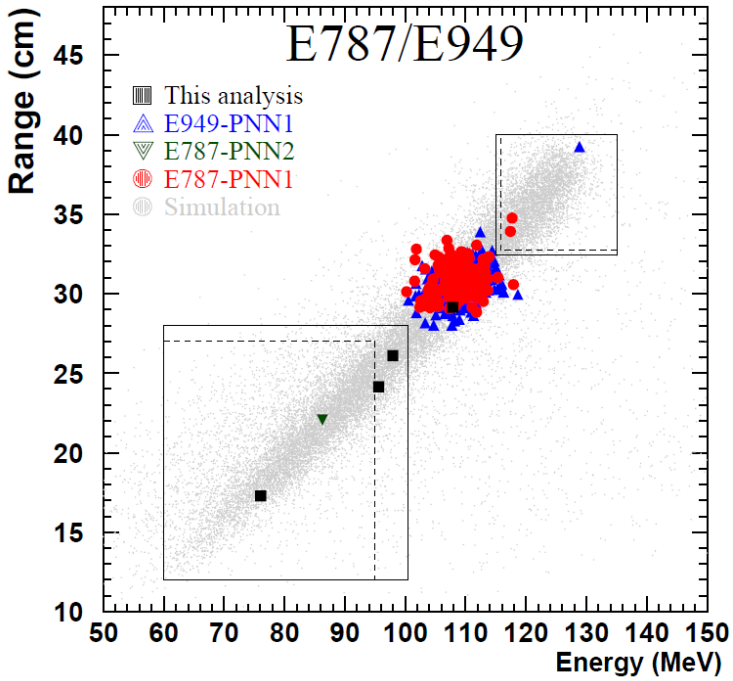


Figura 1.5: Piano (E_+, r_+) in cui sono mostrate le regioni fiduciali e gli eventi $K^+ \to \pi^+ \nu\bar{\nu}$ rivelati dagli esperimenti E787+E949.

La misura del *branching ratio* del decadimento $K_L \to \pi^0 \nu\bar{\nu}$ risulta difficile sperimentalmente per la difficoltà di identificare il K_L e il π^0. L'esperimento E246a a KEK [1] e' stato il primo specificatamente dedicato alla ricerca del decadimento $K^0 \to \pi^0 \nu\bar{\nu}$. Visto che non è stato osservato nessun evento di segnale, è stato messo un limite superiore al *branching ratio* del decadimento, al 90% di livello di confidenza e basato sulla statistica di Poisson: (eq. 1.33) da confrontare con il valore teorico $BR\left(K^0 \to \pi^0 \nu\bar{\nu}\right)(SM) = (2.49 \pm 0.39) \times 10^{-11}$.

$$BR\left(K^0 \to \pi^0 \nu\bar{\nu}\right) < 6.7 \times 10^{-8} \tag{1.33}$$

1.3 La Strategia Sperimentale e i Principali fondi di NA62

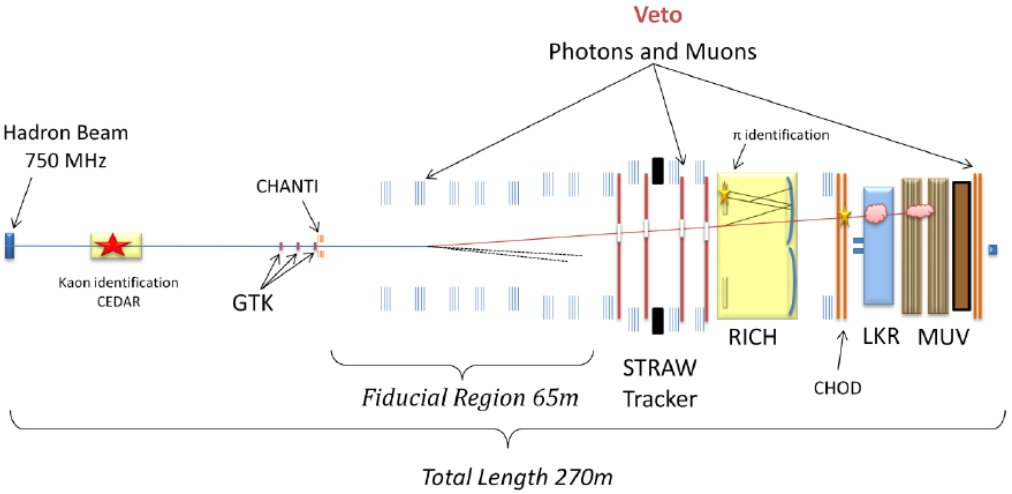


Figura 1.6: Vista schematica di NA62 che mostra i principali rivelatori.

NA62 (fig. 1.6) si servirà di fasci di protoni a 400 GeV/c provenienti dal SPS in modo da utilizzare K^+ di impulso intorno a 75 GeV/c che decadranno *in volo*. Ci sono due principali vantaggi[22] nell'uso di alte energie:

1. la sezione d'urto di produzione dei K^+ aumenta al crescere dell'energia dei protoni comportando un maggiore flusso di K^+ a parità di flusso di protoni;

2. La rivelazione di fotoni prodotti da decadimenti di fondo risulta più semplice ad alta energia. Si studierà la regione cinematica in cui l'impulso dei π^+ nel laboratorio è inferiore a 35 GeV/c; in questo modo, nel caso del fondo $K^+ \to \pi^+\pi^0$ i fotoni depositeranno almeno 40 GeV/c nei veti, riducendo la possibilità che entrambi i fotoni non siano rivelati per inefficienze o reazioni fotonucleari.

L'utilizzo di protoni ad alta energia porta anche ad uno svantaggio: i protoni e i pioni non possono essere separati efficientemente dai K^+ nel fascio secondario con la conseguenza che i rivelatori, che misurano la direzione e l'impulso dei K^+ a monte della regione di decadimento, sono esposti ad un flusso (tab. 1.1) 17 volte superiore a quello dei soli K^+. I rivelatori a valle

P	K^+	π^+
171×10^6	53×10^6	532×10^6

Tabella 1.1: Flusso alla produzione in 42 μsr $\Delta\Omega$ /10^{12} protoni incidenti per s.

della regione di decadimento non subiscono questo aumento di flusso in quanto i protoni, i π e i K^+ non decaduti restano nella *beam pipe* senza illuminarli. Allo stesso modo i muoni provenienti dal decadimento dei pioni restano principalmente all'interno della *beam pipe* a causa del loro piccolo impulso trasverso rispetto alla loro energia. Un ulteriore conseguenza dell'uso di alte energie consiste nel fatto che non si possono arrestare i K^+, quindi si lavora nel vuoto in modo da eliminare le difficoltà legate alla presenza di materiale.

Per una buona riuscita dell'esperimento NA62, la sua collaborazione deve fronteggiare alcune sfide:

- la tracciatura delle particelle del fascio con una frequenza di ∼1 GHz, con il minimo tempo morto ed un elevata risoluzione spaziale;
- l'utilizzo di un contatore Čerenkov differenziale (CEDAR)insensibile a protoni e pioni per una positiva identificazione dei K^+ nel flusso ad alta frequenza di particelle con il minimo errore di identificazione accidentale;
- la costruzione di un sistema ermetico di veti per i fotoni (LAV) in modo da provvedere ad un reiezione di $\sim 10^8$ dei π^0;
- l'utilizzo di un sistema di rivelatori di veto per i muoni con un efficienza di rigetto di almeno $\sim 10^5$;
- la separazione dei π dai μ fino ad impulsi di 35 GeV/c a due deviazioni standard attraverso l'uso del RICH;
- l'uso di misure ridondanti dell'impulso entrante del K^+ e di quello uscente del π^+ per sopprimere le code nella ricostruzione della massa mancante dei decadimenti a due corpi $K^+ \to \pi^+\pi^0$;
- riconoscere e rigettare le particelle cariche provenienti da decadimenti a tre o quattro corpi del K^+;
- minimizzare l'attività accidentale dei decadimenti di particelle diverse[1] da K^+.

Per questo esperimento ci aspettiamo un *rate* complessivo integrato sui rivelatori di ~ 10 MHz. Per raggiungere un rapporto segnale su fondo (S/B) di 10 è necessario impiegare una combinazione di differenti tecniche di soppressione dei fondi; questo ci permette anche di misurare il fattore di reiezione dai dati invertendo un taglio per volta.

Dato che la coppia neutrino-antineutrino non è rivelabile, la segnatura di un evento di segnale $K^+ \to \pi^+\nu\bar{\nu}$ consiste in una singola traccia dovuta al π^+ a valle della regione di decadimento collegata temporalmente, spazialmente e angolarmente alla traccia di un K^+ a monte della regione di decadimento. Quindi gli eventi di decadimento di un K^+ con una segnatura simile fanno parte del fondo; anche decadimenti con più particelle cariche possono imitare eventi di segnale se una traccia non viene rivelata per inefficienze o perché è fuori dall'accettanza dei rivelatori. Si possono riscontrare contributi ai fondi dovuti a particelle del fascio che interagiscono con particelle di gas residuo o con gli ultimi rivelatori del fascio a monte della regione di decadimento, se queste accidentalmente corrispondono in tempo ad un K^+ entrante. Quindi, per ridurre al minimo i fondi, la costruzione dei rivelatori dell'esperimento ha seguito quattro linee guida: una ricostruzione cinematica accurata, una precisa misura temporale delle particelle, un efficiente sistema di veto e di riconoscimento delle particelle.

Visto che i *branching ratio* dei principali canali di decadimento del K^+ (tab. 1.2) sono fino a 10^{10} volte superiori a quello teorico del segnale, per raggiungere un rapporto segnale su fondo pari a 10 è necessario che le inefficienze di reiezione degli altri canali di decadimento siano inferiori a 10^{-11}.

La ricostruzione della cinematica del decadimento è una tecnica di reiezione molto utile per i canali di decadimento a due corpi nei casi in cui i fotoni provenienti dal π^0 del decadimento $K^+ \to \pi^+\pi^0$ non siano rivelati o se il μ del decadimento $K^+ \to \mu^+\nu$ viene scambiato per un π^+; vengono utilizzate in congiunzione le tecniche di reiezione cinematiche e l'identificazione delle particelle.

Le uniche quantità misurabili per la cinematica del decadimento $K^+ \to \pi^+\nu\bar{\nu}$ (fig. 1.7) sono l'impulso del K^+ entrante $\vec{P}_K$, l'impulso del π^+ uscente $\vec{P}_\pi$ e l'angolo formato dai due vettori $\vartheta_{K\pi}$. Con solo queste tre quantità, è conveniente usare la massa mancante quadrata

[1] Come ad esempio i muoni provenienti dal bersaglio di produzione, i protoni del fascio primario, i pioni e i K che decadono a monte della regione di decadimento.

canale di decadimento	branching ratio (%)	modalità di reiezione
$K^+ \to \mu^+\nu$	63.54	cinematica due corpi + identificazione μ
$K^+ \to \pi^+\pi^0$	20.66	cinematica due corpi + veto fotoni
$K^+ \to \pi^+\pi^+\pi^-$	5.59	cinematica + veto particelle cariche
$K^+ \to \pi^0 e^+\nu$	5.07	E/P + veto fotoni
$K^+ \to \pi^0\mu^+\nu$	3.35	identificazione μ + veto fotoni
$K^+ \to \pi^+\pi^0\pi^0$	1.76	cinematica + veto fotoni

Tabella 1.2: *Branching ratio* e modalità di reiezione dei principali canali di decadimento del K^+. La reiezione E/P si basa sulla misura dell'energia rilasciata dalla particella carica nel calorimetro elettromagnetico e del suo impulso con lo spettrometro magnetico, se la particella rivelata è un elettrone la quantità E/P deve essere ~ 1.

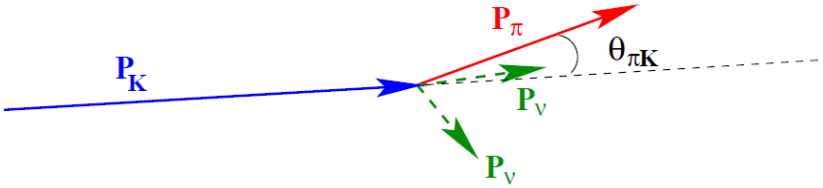


Figura 1.7: Cinematica del decadimento $K^+ \to \pi^+\nu\bar{\nu}$.

m^2_{miss} (eq. 1.34) definita come il quadrato della differenza tra il quadrimpulso del K^+ e della traccia carica rivelata a valle della regione di decadimento sotto l'ipotesi che questa particella sia un π^+.

$$m^2_{miss} \stackrel{def}{=} (P_K - P_\pi)^2 \tag{1.34}$$

$$= m^2_K + m^2_\pi - 2E_K E_\pi + 2\left|\vec{P}_K\right|\left|\vec{P}_\pi\right|\cos\vartheta_{K\pi} \tag{1.35}$$

Nel limite in cui $\vec{P}_K >> m_K$ e $\vec{P}_\pi >> m_\pi$ si ottiene l'equazione:

$$m^2_{miss} \simeq m^2_K\left(1 - \frac{\left|\vec{P}_\pi\right|}{\left|\vec{P}_K\right|}\right) + m^2_\pi\left(1 - \frac{\left|\vec{P}_K\right|}{\left|\vec{P}_\pi\right|}\right) - \left|\vec{P}_K\right|\left|\vec{P}_\pi\right|\vartheta^2_{K\pi} \tag{1.36}$$

Come si vede dalle strategie di reiezione mostrate nella tabella 1.2, non tutti i decadimenti sono rigettabili cinematicamente; è comunque possibile utilizzare la cinematica nella reiezione del 92% dei decadimenti del $K^+(fig.1.8)$.

Consideriamo ora i decadimenti più importanti; $K^+ \to \pi^+\pi^0$ essendo un decadimento a due corpi, se ignoriamo gli effetti della risoluzione, viene visualizzato in un istogramma di massa mancante come una linea posizionata nel punto con $m^2_{miss} = m^2_\pi$. Per i due decadimenti a tre π, $K^+ \to \pi^+\pi^+\pi^-$ e $K^+ \to \pi^+\pi^0\pi^0$, il valore della massa mancante quadrata è sempre superiore ad un limite inferiore chiamato in seguito $min\ m^2_{miss}(\pi\pi\pi)$. Il decadimento $K^+ \to \mu^+\nu$, nonostante sia un decadimento a due corpi, non appare come una linea a $m^2_{miss} = 0$ perché l'assunzione utilizzata, cioè che la particella rivelata sia un π, è sbagliata. La forma invece dipende dall'impulso della particella nello stato finale ed ha $m^2_{miss} = 0$ come estremo superiore.

Come si può vedere dalla figura (1.8) la linea del decadimento $K^+ \to \pi^+\pi^0$ si trova all'interno della regione del segnale. Questo decadimento divide quindi la regione di accettanza del segnale in due parti, chiamate *regione I* e *regione II*, separate dalla linea del decadimento $K^+ \to \pi^+\pi^0$. Queste due regioni sono delimitate rispettivamente dalla regione del decadimento $K^+ \to \mu^+\nu$ e da quella dei due decadimenti in tre π.

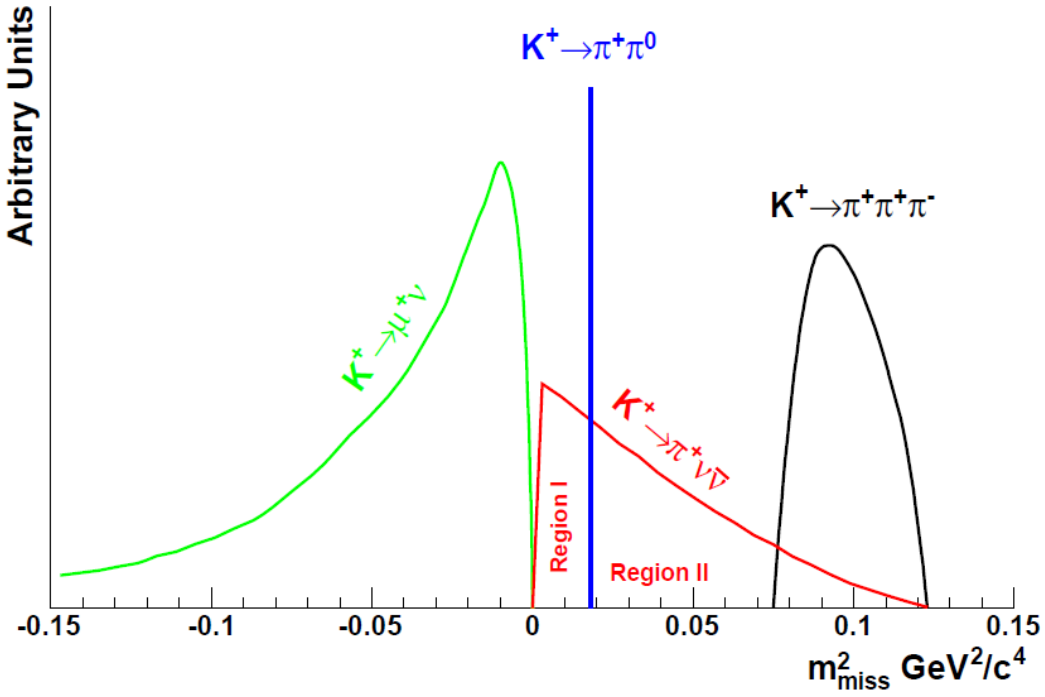


Figura 1.8: Distribuzione della m^2_{miss} per il segnale $K^+ \to \pi^+\nu\bar{\nu}$ e per i fondi principali rigettabili attraverso la cinematica per $\left|\vec{P}_K\right| = 75$ GeV/c.

- Regione I: $0 < m^2_{miss} < m^2_\pi - (\Delta m)^2$
- Regione II: $m^2_\pi + (\Delta m)^2 < m^2_{miss} < min\ m^2_{miss}(\pi\pi\pi) - (\Delta m)^2$

Il termine $(\Delta m)^2$ dipende dalla risoluzione della massa mancante quadrata. Si è stimato che per ottenere un rapporto segnale su fondo pari a 10 con un accettanza del segnale del 10%, un'inefficienza di veto dei π^0 dell'ordine di 10^{-8} e un'inefficienza di veto dei μ dell'ordine di 10^{-6} si deve avere una risoluzione minima sulla massa mancante quadrata di 0.008 GeV$^2/c^4$ [22]. Per raggiungere questa risoluzione su m^2_{miss} sono stati imposti dei requisiti per i due rivelatori adibiti alla ricostruzione delle tracce della particella entrante e di quella uscente, rispettivamente il Gigatracker e lo spettrometro magnetico composto da camere a STRAW. Oltre alle condizioni sulla risoluzione spaziale, il Gigatracker necessita di una elevata risoluzione temporale, in quanto il grande *rate* di particelle può portare ad una situazione in cui un π misurato dai rivelatori a valle della regione di decadimento viene associato ad un K sbagliato.

L'8% del *branching ratio* dei decadimenti del K^+ è composto da decadimenti radiativi e semi-leptonici che non possono essere rigettati cinematicamente (fig. 1.9). Per questi canali di decadimento l'identificazione delle particelle insieme al sistema di veti per i fotoni e per le particelle cariche sono gli unici strumenti che si possono usare. Per sopprimere tutti i decadimenti che potrebbero imitare il segnale $K^+ \to \pi^+\nu\bar{\nu}$ è quindi necessario rendere ermetici i rivelatori specialmente per quanto riguarda i fotoni provenienti da π^0 originati nella regione di decadimento fiduciale lunga 65 m e che inizia a $\sim$ 100 m dal bersaglio di produzione. Per ottenere questo risultato la copertura angolare deve essere totale per angoli inferiori a 50 mrad: un sistema di veti anulari per fotoni composto da 12 rivelatori a grande angolo (LAV) provvede alla copertura degli angoli compresi tra 8.5 mrad e 50 mrad; tra 8.5 e 2 mrad la copertura è data dal calorimetro a kripton liquido (LKr) mentre per angoli

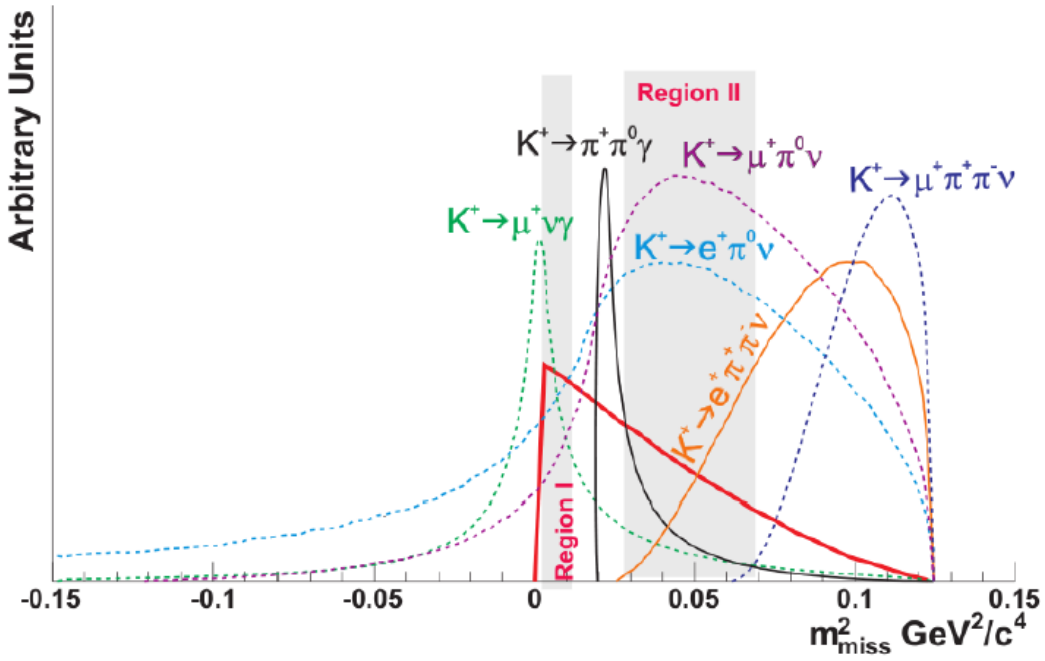


Figura 1.9: Distribuzione della m^2_{miss} per il segnale $K^+ \to \pi^+\nu\bar{\nu}$ e per i fondi non rigettabili attraverso la cinematica per $\left|\vec{P}_K\right| = 75$ GeV/c.

inferiori è stato previsto un calorimetro intermedio ad anello (IRC) che coprirà la corona circolare intorno al fascio e un calorimetro per piccoli angoli (SAC) che sarà posizionato sull'asse del fascio alla fine dell'apparato sperimentale; un dipolo magnetico si occuperà di deviare le particelle cariche del fascio in modo da allontanarle dal SAC.

Infine esistono alcuni decadimenti (con *branching ratio* $\geq 10^{-5}$) come $K^+ \to \pi^+\pi^- e^+\nu$ e $K^+ \to \pi^+\pi^-\mu^+\nu$ nei quali il leptone carico può non essere rivelato perché se viaggiasse, ad esempio, in prossimità della *beam line* sarebbe impossibile distinguerlo dalle altre particelle del fascio; similarmente ai decadimenti con il π^0, è necessario che il π^- sia rivelato per non scambiare l'evento per segnale. Per questo motivo è necessario che i rivelatori siano ermetici per particelle cariche con impulso inferiore a 60 GeV/c; a questo scopo può essere usato lo spettrometro magnetico composto da quattro camere a STRAW e posizionato a valle della regione di decadimento.

1.4 Altri Esperimenti

Altri due esperimenti, con lo scopo di misurare il *branching ratio* dei decadimenti $K^+ \to \pi^+\nu\bar{\nu}$ e $K_L \to \pi^0\nu\bar{\nu}$, sono in fase di approvazione o progettazione. Il primo è l'esperimento KOTO[29] in costruzione a JPARC che ha lo scopo di ottenere la prima osservazione del decadimento $K_L \to \pi^0\nu\bar{\nu}$, utilizzando fasci primari di protoni da 30 GeV/c, ed in seguito arrivare a raccoglierne fino a 100 riutilizzando e aggiornando i rivelatori dell'esperimento KEK-E391a. L'esperimento KOTO dovrebbe iniziare la presa dati nel 2013. Il secondo appartiene ad un complesso di esperimenti che dovrebbero aver luogo al Fermilab sfruttando un futuro acceleratore di protoni di grande intensità noto come *Project X* [13] e mira a raccogliere 1000 eventi di $K^+ \to \pi^+\nu\bar{\nu}$ e 1000 eventi di $K_L \to \pi^0\nu\bar{\nu}$ utilizzando fasci di protoni da 3 GeV/c. La costruzione dell'acceleratore Project X non è ancora approvata e quindi tali esperimenti potrebbero essere realizzati al più presto dopo il 2020.

CAPITOLO 2

APPARATO SPERIMENTALE

Indice

La composizione del fascio di NA62 e la disposizione dei suoi rivelatori fanno uso dell'esperienza guadagnata dalla collaborazione in seguito al precedente esperimento NA48 [6] eseguito al CERN SPS. Dopo $\sim$ 100 m dal bersaglio di produzione in berillio inizia la regione di decadimento, lunga 170 m, in cui sono posti i principali elementi per la rivelazione dei prodotti di decadimento dei K^+. Per il resto del capitolo poniamo l'asse z lungo la direzione su cui è disposto l'apparato sperimentale e la *beam pipe*. Al suo interno si trova la regione fiduciale lunga 65 m a cui ci si restringerà quando verranno analizzati i dati dell'esperimento. L'apparato sperimentale di NA62 (fig. 2.1) comprende i seguenti rivelatori:

- il *Čerenkov Differential counter with Achromatic Ring focus* (CEDAR), un contatore differenziale Čerenkov, ha lo scopo di identificare i K^+ all'interno del fascio di particelle;
- il Gigatracker (GTK) è composto da tre stazioni di micro-pixel di silicio adatte a misurare tempo, direzione ed impulso delle particelle del fascio prima che entrino nella regione di decadimento;
- Il *CHarged ANTIcounter* (CHANTI) è un set di anelli di materiale scintillante, che circondano l'ultima stazione del Gigatracker, utili come veto per le particelle cariche prima che entrino nella regione di decadimento.
- Un sistema di rivelatori di veto per i fotoni provvede ad una copertura da 0 fino a $\sim$50 mrad attraverso l'utilizzo di una serie di 12 veti anulari a grande angolo (LAV), un

calorimetro elettromagnetico a krypton liquido (LKr), un calorimetro intermedio ad anello (IRC) per coprire la corona circolare intorno al fascio ed un calorimetro per piccoli angoli (SAC) posizionato in fondo alla regione di decadimento;

- uno spettrometro magnetico a camere a STRAW che verrà utilizzato per misurare la posizione dei vertici di decadimento, la direzione e l'impulso delle particelle cariche secondarie;
- un Ring Imaging Čerenkov (RICH) adatto a distinguere i π^+ dai μ^+ nell'intervallo di impulsi tra 15 e 35 GeV/c e a misurare la direzione e la velocità delle particelle;
- il Charged Hodoscope (CHOD) che è un odoscopio carico segmentato trasversalmente concepito per funzioni di *trigger* e di misura temporale del passaggio di particelle cariche;
- un sistema di veto per i muoni (MUV) composto da tre moduli;

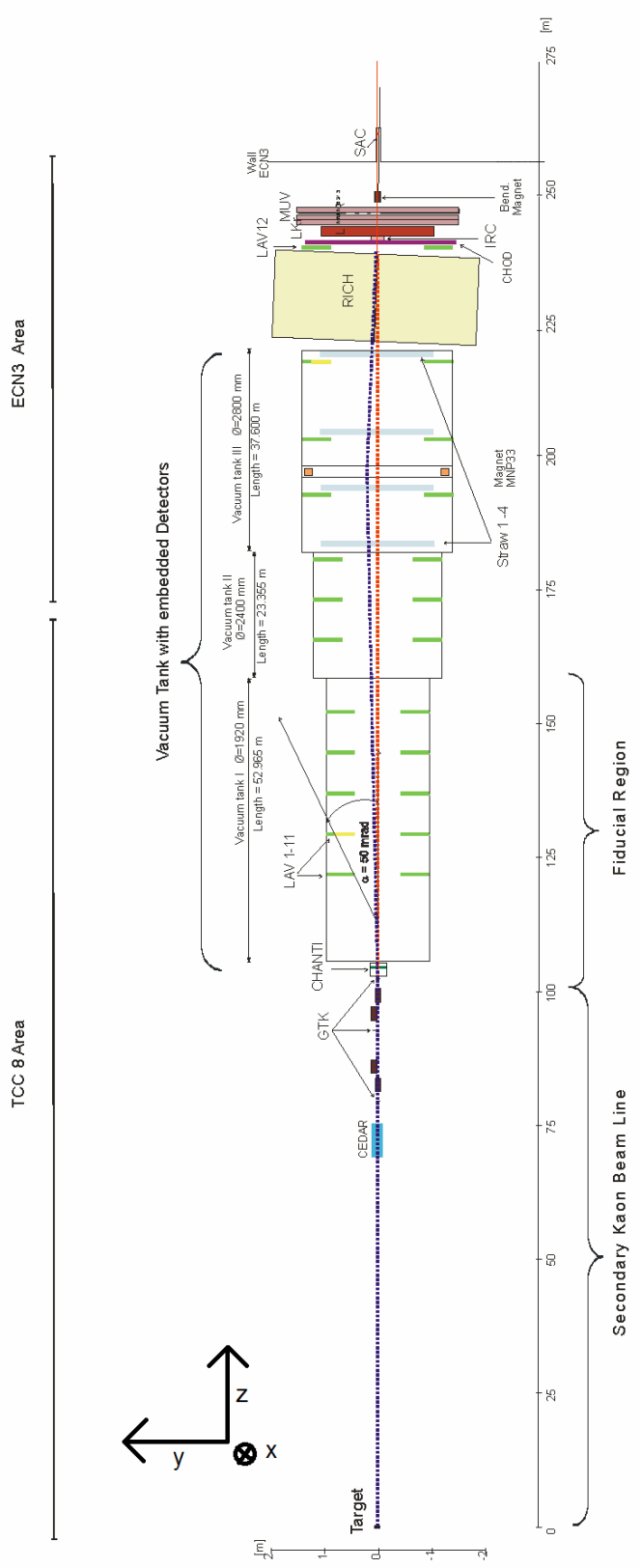


Figura 2.1: Vista longitudinale dell'apparato sperimentale di NA62.

2.1 La Linea del Fascio

Ci sono numerosi vantaggi nell'utilizzo di K carichi e quindi protoni ad alta energia (400 GeV/c). Da una semplice formula empirica, ottenuta con un fit di dati sperimentali sulla produzione di particelle secondarie [8], si ottiene che la massima produzione di K^+, per un fissato intervallo di impulsi $\Delta p_K/p_K$ per i K^+ e con protoni primari di impulso p_0, avviene per $p_K \sim 0.35p_0$ e che, tenendo costante il rapporto p_K/p_0, la produzione di K^+ aumenta come p_K^2 e quindi come p_0^2. Il numero di K^+ che decadono all'interno della regione fiduciale è massimo per $p_K \sim 0.23p_0$ e, tenendo costante il rapporto p_K/p_0, aumenta come p_K e quindi come p_0. Un ulteriore vantaggio dovuto all'utilizzo di fasci ad alta energia è legato al miglioramento dell'accettanza, della risoluzione e della capacità di reiezione di alcuni rivelatori come i veti per i fotoni e i muoni e per il calorimetro.

L'utilizzo di fasci ad alta energia comporta anche uno svantaggio, ovvero la difficoltà di separare i K dalle altre particelle del fascio in quanto non è possibile utilizzare la separazione a radio-frequenza. Visto che la distanza tra le due cavità a radio-frequenza aumenta come p_K^2 con $p_K = 75$ GeV/c, sarebbe necessario far percorrere più di 2.5 km alle particelle per separarle con il risultato, però, di avere un flusso di K^+ ridotto al 4%.

Dai due andamenti mostrati precedentemente segue che il rapporto tra il numero di K^+ che decadono nella regione fiduciale ed il numero di particelle del fascio è inversamente proporzionale a p_K. L'impulso dei K è stato scelto pari a 75 GeV/c come compromesso tra questo rapporto e la dipendenza del numero di K^+ prodotti da p_K^2 in modo da massimizzare il numero di decadimenti $K^+ \to \pi^+\nu\bar{\nu}$ rispetto al flusso totale di particelle.

Sono stati scelti i K positivi piuttosto che quelli negativi perché, con fasci primari di protoni da 400 GeV/c e K da 75 GeV/c, il rapporto tra i *rate* di produzione dei K^+ e quelli dei K^- è uguale a 2.1 [3] e perché nei fasci positivi c'è una maggiore concentrazione di K rispetto ai π ($(K^+/\pi^+)/(K^-/\pi^-)$). Sono invece uguali le concentrazioni dei K positivi o negativi rispetto al flusso totale delle particelle della stessa carica ($K^+/totale^+ \approx K^-/totale^-$).

Il fascio primario composto da protoni è focalizzato e diretto con angolo zero[1] su un bersaglio di berillio di 2 mm di diametro e 400 mm di lunghezza sospeso tra due lamine di alluminio e raffreddato ad aria. Il fascio uscente dal bersaglio passa attraverso un collimatore di rame ad apertura variabile lungo 950 mm raffreddato ad acqua che assorbe le particelle con angolo maggiore di 6 mrad prima che decadano in muoni. Successivamente all'entrata nella camera a vuoto, il fascio passa attraverso un ulteriore collimatore, lungo 1.6 m e con un foro di diametro 28 mm, che riduce il suo angolo di apertura a $\sim$ 4 mrad.

Dopo il primo collimatore il fascio viene focalizzato da tre quadrupoli e poi deflesso verticalmente da quattro dipoli che formano un *front-end achromat*. I primi due dipoli, di polarità opposta l'uno all'altro, selezionano la componente positiva del fascio con momento $\sim$ 75 GeV/c. In seguito il fascio passa attraverso un radiatore di tungsteno con spessore 1.06 X_0 con lo scopo di far perdere ai positroni, per *Bremsstrahlung*, un energia sufficiente a farne rigettare il 99.6% nel passaggio attraverso il terzo e quarto dipolo che ripristinano la direzione iniziale del fascio successivamente rifocalizzato da un tripletto di quadripoli. Il fascio risultante possiede un impulso medio di 75 GeV/c, una deviazione standard di 0.9 GeV/c ed è formato solo da particelle di carica positiva. Successivamente il fascio viene reso più largo da una coppia di quadrupoli in modo da ottimizzare l'identificazione dei K^+ da parte del CEDAR e dopo il suo attraversamento un ulteriore coppia di quadrupoli ripristina la collimazione precedente. Il fascio, poi, passa da un secondo *achromat*, dalle tre stazioni del Gigatracker e dal CHANTI, collimatore attivo ad anello che veta le particelle periferiche del fascio. Il fascio finale ha un *rate* di 750 MHz e risulta composto [21] per il 70% da π^+, il 23% da protoni e per il 6% da K^+.

[1] La scelta dell'angolo uguale a zero comporta una maggiore produzione di K^+.

2.2 Rivelatori a Monte della Regione di Decadimento

2.2.1 Il CEDAR

A causa dell'alto rate di particelle (750 MHz), dovuto all'alta energia del fascio secondario e quindi all'impossibilità di separare i K dai pioni e dai protoni, l'identificazione dei K^+ risulta un aspetto critico dell'esperimento. Per identificare i K^+ all'interno del fascio e vetare i pioni e i protoni, verrà usato un contatore Čerenkov differenziale riempito con idrogeno, il CEDAR, che rivela solo la luce Čerenkov emessa ad un determinato angolo rispetto alla direzione del fascio.

Quando una particella attraversa un gas radiatore, di indice di rifrazione n, con velocità β (in unità della velocità della luce c) emette un cono di luce Čerenkov con angolo ϑ_c ($\cos\vartheta_c = \frac{1}{n\beta}$) rispetto alla direzione della particella. Visto che nell'effetto Čerenkov l'angolo di emissione è collegato alla velocità della particelle e all'indice di rifrazioni e che il fascio è selezionato in impulso dall'angolo di emissione possiamo conoscere la massa delle particelle. Tenendo l'angolo costante è possibile selezionare le diverse particelle variando la pressione del gas e di conseguenza il suo indice di rifrazione. Nella figura 2.2 è mostrato un esempio di selezione di protoni, pioni e K usando il CEDAR riempito con azoto e variando la pressione del gas.

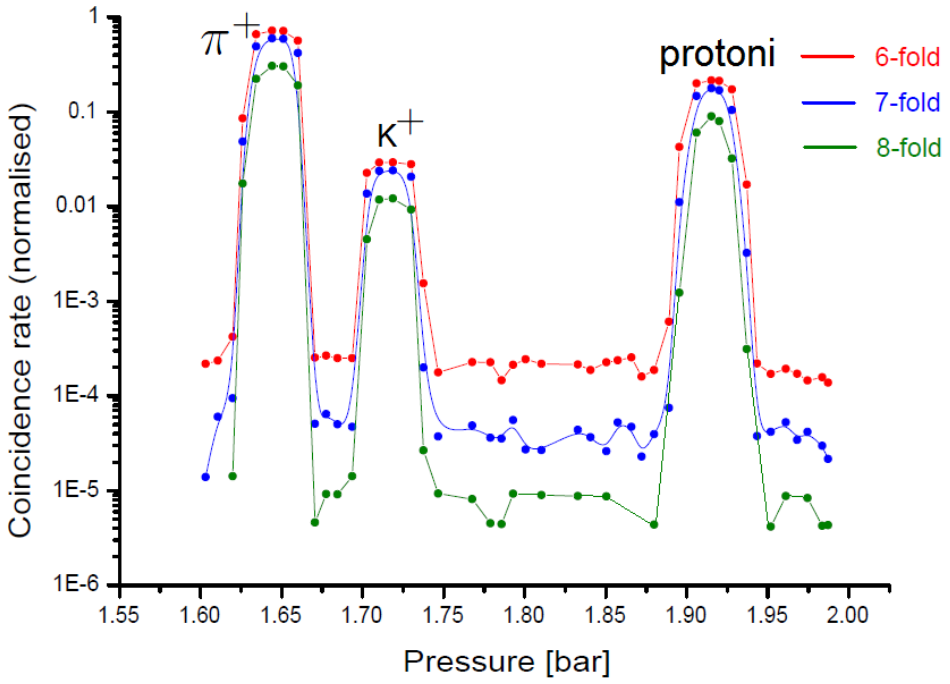


Figura 2.2: *Rate* misurato nel CEDAR in funzione della pressione del gas (N_2) e normalizzato al *rate* totale del fascio [21]. I tre picchi corrispondono ai π^+, ai K^+ e ai protoni. Le tre serie di punti e curve corrispondono a diverse richieste di molteplicità.

Utilizzando l'idrogeno la pressione necessaria a selezionare i K^+ e di ~ 3.6 bar.

Il CEDAR utilizzato in questo esperimento è una versione aggiornata di un rivelatore già esistente al SPS chiamato West CEDAR [12]. Sono state eseguite due principali modifiche: l'utilizzo dell'idrogeno come radiatore al posto dell'azoto in modo da minimizzare la diffusione multipla, e la sostituzione degli 8 fotomoltiplicatori con 8 *cluster* di dispositivi multi-anodo, composti ognuno da 32 fotomoltiplicatori, al fine di ridurre il *rate* in ciascun canale e quindi l'inefficienza dovuta al tempo morto.

La luce Čerenkov emessa dal fascio viene riflessa da uno specchio sferico di Mangin[2], posizionato alla fine del rivelatore, verso una lente di correzione cromatica che la focalizza in direzione dei fotomoltiplicatori (fig. 2.3). Tra la lente di correzione cromatica e i fotomoltiplicatori si trova un diaframma ad anello di 100 mm di raggio con apertura variabile che permette l'arrivo ai fotomoltiplicatori solo della luce emessa ad un determinato angolo.

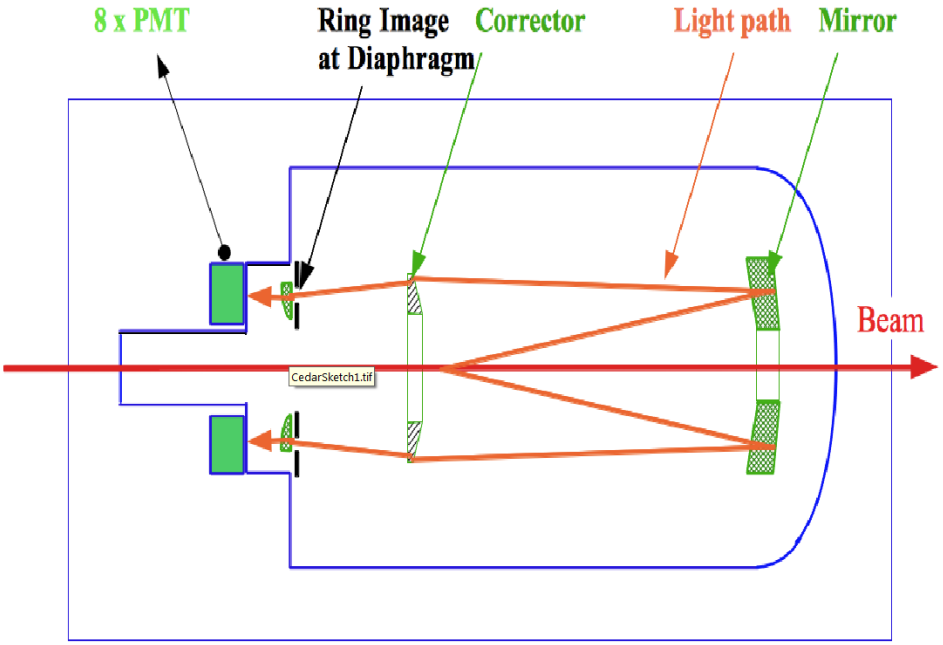


Figura 2.3: Schematizzazione del CEDAR e del cammino ottico percorso dalla luce Čerenkov.

2.2.2 Il Gigatracker

Il Gigatracker è uno spettrometro composto da tre stazioni di rivelatori a micro-pixel di silicio (fig. 2.4) il cui scopo è di misurare con precisione la direzione del K^+, il suo impulso e il tempo del suo passaggio. Queste misure di precisione sono necessarie per associare con sicurezza il K^+ alle tracce dei prodotti di decadimento misurate a valle della regione di decadimento dallo spettrometro magnetico a camere a STRAW e per la reiezione dei fondi attraverso tagli cinematici basati sulla massa mancante. La possibilità di un errata associazione tra la traccia del pione e quella del K^+ comporterebbe, quindi, una reiezione cinematica meno efficace. Per evitare che avvengano queste errate associazioni è necessario che il Gigatracker abbia una risoluzione temporale di $\sim$ 150 ps e, tenendo contro della risoluzione delle STRAW, che il momento della particella sia misurato con una risoluzione relativa $\sigma(p_K)/p_K \sim 0.2\%$ e la sua direzione con una risoluzione di $\sim$ 15 μrad.

Ciascuna stazione (fig. 2.5 e fig. 2.6) contiene 18000 pixel di dimensioni 300x300 μm^2 per un totale di 60x27 mm^2 ed è spessa 300 μm di cui 200 μm sono occupati dal sensore al silicio e 100 μm dall'elettronica di lettura composta da 5x2 *chip*. Il rivelatore ha uno spessore totale di 0.5% X_0 risultante dalla somma dei contributi del sensore (0.22% X_0), dei *chip* e del supporto. La progettazione si è basata su un buon compromesso tra il costo del rivelatore e la risposta al passaggio di una MIP ($\sim$ 15000 elettroni).

Il nome Gigatracker deriva dalla possibilità che le tre stazioni siano sottoposte ad un flusso di particelle con picchi di 1 GHz. A causa di questo è necessario che le stazioni siano poco deteriorabili e che abbiano un'elevata efficienza anche a questo *rate*. Il Gigatracker

[2]Lo specchio di Mangin è un menisco di vetro alluminizzato sulla superficie posteriore.

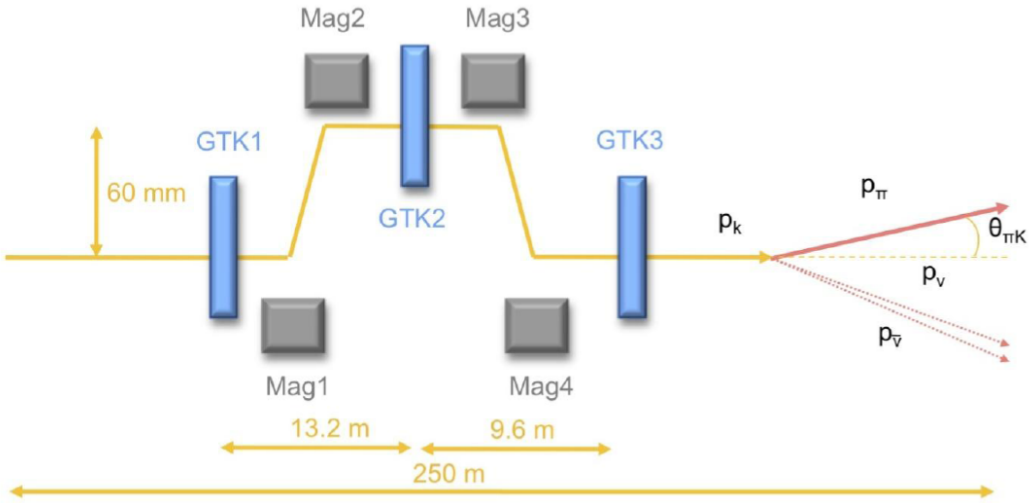


Figura 2.4: Schema delle tre stazioni del Gigatracker e schema del decadimento di interesse.

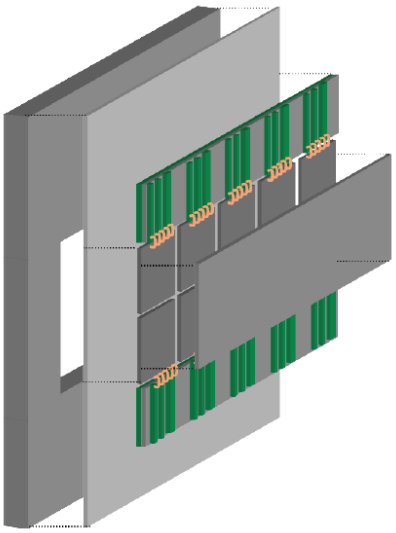


Figura 2.5: Vista espansa di una stazione del Gigatracker in cui si possono riconoscere i 5x2 *chip*, la tavoletta rettangolare con i 18000 pixel e l'elettronica di lettura.

ha richiesto un intenso lavoro di sviluppo, non essendo mai stati realizzati in precedenza rivelatori al silicio con risoluzioni temporali paragonabili. L'alto *rate* comporta anche una grande dissipazione di calore(32 W) da parte dei 10 *chip* dell'elettronica di lettura, per cui è necessario utilizzare un sistema di raffreddamento composto da una struttura di supporto in fibra di carbonio, su cui sono montate le stazioni del Gigatracker, che viene raffreddata con un sistema di refrigerazione posto fuori dalla portata del fascio.

2.2.3 Il CHANTI

Subito dopo l'ultima stazione del Gigatracker, all'entrata della camera a vuoto, è posizionato il CHANTI, un rivelatore che ha lo scopo di ridurre il fondo causato dal'interazione del fascio con il Gigatracker e con l'ultimo collimatore e dall'alone di muoni che circonda il fascio. Gli eventi causati dall'interazione anelastica del fascio coll'ultima stazione del Giga-

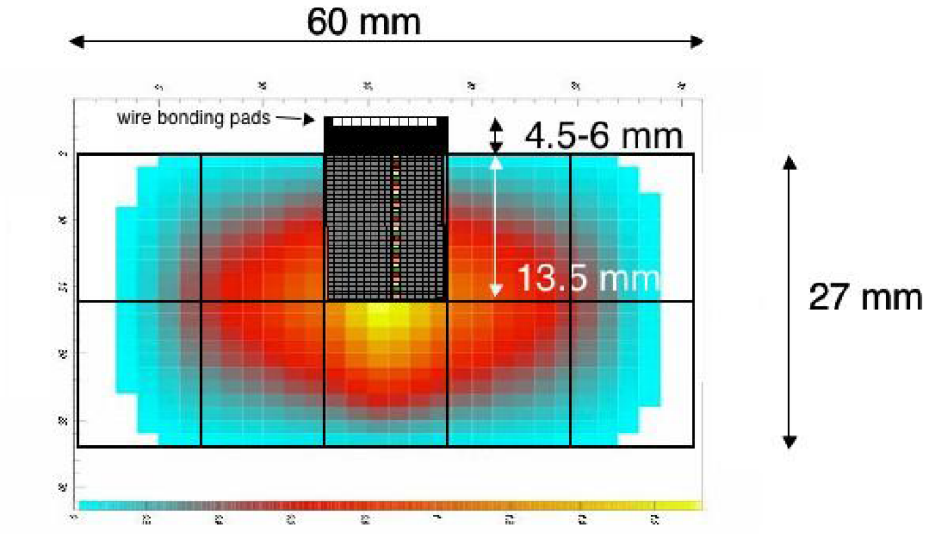


Figura 2.6: Profilo del fascio su una stazione del Gigatracker.

tracker sono molto pericolosi in quanto le particelle prodotte nell'interazione, in particolare i pioni emessi a piccolo angolo, possono raggiungere lo spettrometro magnetico con camere a STRAW e simulare il decadimento di un K^+ nella regione fiduciale di decadimento. Inoltre se nessuna altra traccia viene rivelata, questo evento può sembrare proprio un evento di segnale $K^+ \to \pi^+ \nu\bar{\nu}$.

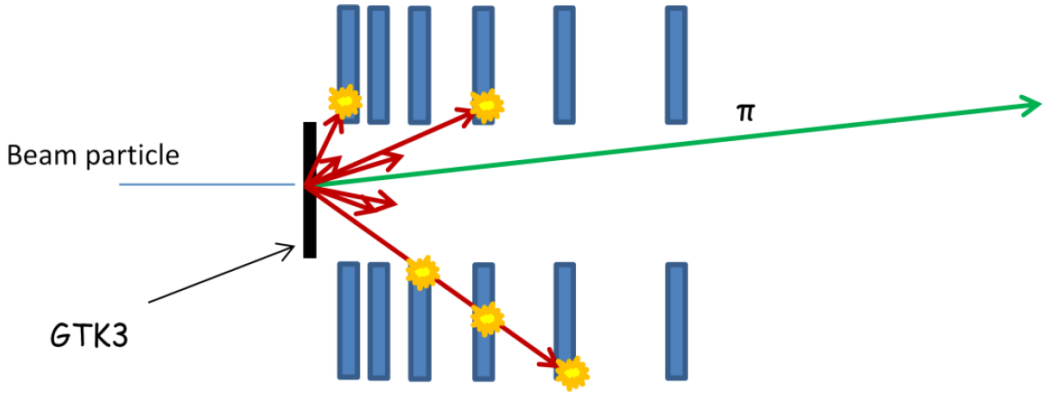


Figura 2.7: Schema delle 6 stazioni del CHANTI ed esempio di interazione del fascio con il Gigatracker.

Il CHANTI è formato da 6 stazioni quadrate (fig. 2.7) con 30 cm di lato e con un foro rettangolare per il fascio, di dimensioni 9 cm x 5 cm che coprono ermeticamente la regione angolare tra 34 mrad e 1.38 rad dall'asse del fascio. In ogni stazione ci sono due piani composti da 22 e 24 barre di scintillatore plastico parallele rispettivamente alla direzione x e y e perpendicolari all'asse del fascio. La luce è raccolta usando delle fibre *wavelength shifter* posizionate all'interno delle barre ed è letta ad un lato da un fotomoltiplicatore.

2.3 Rivelatori a Valle della Regione di Decadimento

2.3.1 Sistema di Veto per i Fotoni

Un sistema di veto per i fotoni è necessario per rigettare un fondo dominante originato dal decadimento $K^+ \to \pi^+\pi^0$ che ha un *branching ratio* del 20.7%. I sistema di veto deve avere un'inefficienza media di reiezione inferiore a 10^{-8}, e la sua copertura geometrica deve essere ermetica fino ad angoli di 50 mrad dall'asse del fascio per i fotoni provenienti dai decadimenti avvenuti nella regione fiduciale. Con questa configurazione solo nello $\sim 0.2\%$ dei casi uno dei due fotoni del π^0 viene perso ad angoli maggiori. Il sistema è composto da quattro rivelatori che coprono di tre differenti regioni angolari:

- i Large Angle Vetoes (LAV) coprono la regione angolare tra 8.5 mrad e 50 mrad;
- il calorimetro a krypton liquido (LKr) copre la regione tra 1 mrad e 8.5 mrad;
- l'Inner Ring Calorimeter (IRC) copre la corona circolare intorno al fascio lasciata scoperta dal LKr;
- lo Small Angle Calorimeter (SAC) è posto in fondo alla regione di decadimento e completa il sistema di veto coprendo fino ad'angolo zero.

I LAV

I LAV sono composti da 12 contatori anulari detti anche ANTI(1-12) (fig. 2.8) che coprono la regione angolare tra 8.5 mrad e 50 mrad dall'asse del fascio. I primi 11 LAV sono posizionati all'interno della regione di decadimento tra 120 m e 220 m dal bersaglio di produzione mentre l'ANTI12 è situato tra il RICH e il calorimetro LKr a 240 m dal bersaglio di produzione.

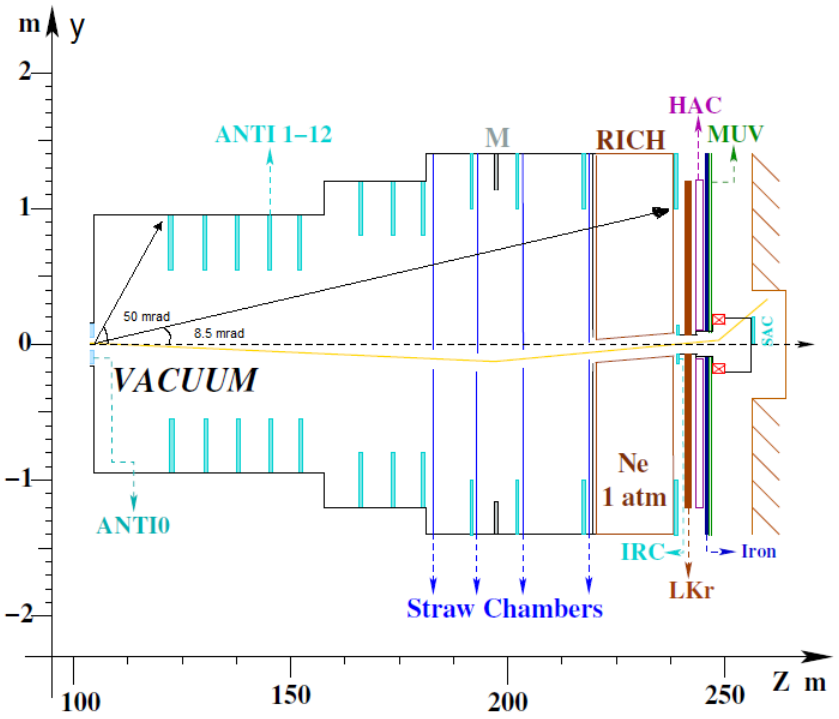


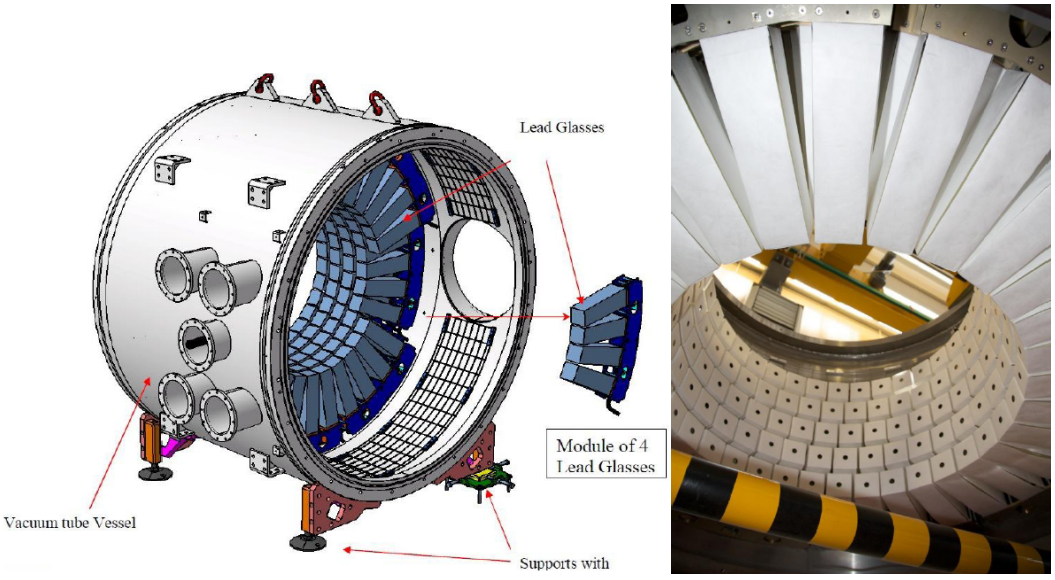
Figura 2.8: Disposizione e copertura angolare dei LAV.

Non è stata riutilizzata la struttura dei veti per i fotoni dell'esperimento NA48, con scintillatori plastici posti all'esterno della regione in vuoto, perché avrebbe avuto un'eccessiva inefficienza di rivelazione per i fotoni a bassa energia ($\sim 4.5\%$). Per questo motivo è stato deciso di istallare i contatori all'interno del tubo a vuoto che deve avere una pressione di $\sim 3 \times 10^{-7}$ mbar, in modo da ridurre le interazioni tra il fascio e le particelle di gas residuo ad un livello accettabile.

Ogni modulo è formato da una struttura a corona circolare che sostiene il materiale attivo costituito da cristalli di vetro al piombo (con il 74% di PbO) a forma di piramide troncata con altezza 37 cm e con basi quadrate di dimensioni 11x11 cm^2 e 9x9 cm^2, originariamente costituenti il calorimetro elettromagnetico dell'esperimento OPAL al CERN.

Ogni cristallo è collegato ad un fotomoltiplicatore attraverso una guida ottica cilindrica in plexiglass. Ogni stazione dei LAV (fig. 2.9) è composta da 4 o 5 anelli ruotati azimutalmente l'uno rispetto all'altro per avere una copertura completa lungo la direzione longitudinale, perché il raggio di curvatura interno è diverso dal raggio di curvatura dell'anello e in ogni anello tra i blocchi resta uno spazio vuoto.

La rivelazione dei fotoni avviene attraverso la produzione di uno sciame elettromagnetico nella loro interazione con i contatori; le coppie e^+ e^- dello sciame generano luce Čerenkov che viene poi rivelata dal fotomoltiplicatore.



(a) (b)

Figura 2.9: Struttura di una stazione completa dei lav (a) e vista interna degli anelli (b).

LKr

Il calorimetro elettromagnetico a Krypton liquido (LKr) è lo stesso precedentemente utilizzato nell'esperimento NA48 al CERN anche se si è reso necessario aggiornare il sistema di lettura in quanto in NA62 ci si aspetta un *rate* L0 di due ordini più grande rispetto a quello di NA48.

Il calorimetro è composto da una camera a ionizzazione di forma approssimativamente ottagonale riempita con 9 m^3 di Krypton liquefatto, scelto per la sua risposta lineare con l'energia depositata, per la sua durata nel tempo e per la sua lunghezza di radiazione (X_0 = 4.7 cm). La sua parte attiva ha uno spessore di 127 cm ($\simeq 27\ X_0$), che permette di contenere interamente sciami elettromagnetici di 50 GeV, e la sua struttura è formata di 13248 celle

quadrate di 2 cm di lato separate da strisce che agiscono da catodo e con all'interno una striscia che agisce da anodo (fig. 2.10). Il calorimetro presenta un eccellente risoluzione in energia ($\frac{\sigma_E}{E} = \frac{0.032}{\sqrt{E}} \oplus \frac{0.09}{E} \oplus 0.0042$ GeV), in posizione ($\sigma_{X,Y} = \frac{0.42}{\sqrt{E}} \oplus 0.06$ cm) e in tempo ($\sigma_t = \frac{2.5}{\sqrt{E}}$ ns)[3] [6].

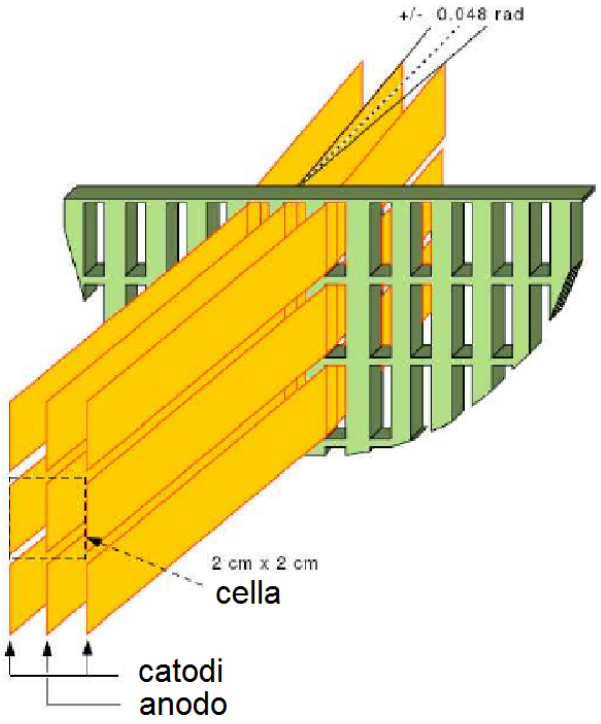


Figura 2.10: Schema delle celle del calorimetro elettromagnetico a Krypton liquido.

Al centro il calorimetro è attraversato da un foro di 8 cm di raggio per far passare il fascio di particelle non decadute; esso è racchiuso in un criostato di alluminio e acciaio necessario a mantenere il Krypton alla temperatura di 121 K.

Lo scopo principale del LKr è di rivelare i fotoni nella regione angolare tra 1 mrad e 8.5 mrad, con un'inefficienza di rivelazione di 10^{-5} sopra 1 GeV, ma può anche essere usato per rigettare il fondo $K^+ \to \pi^0 e^+ \nu$ misurando l'energia dell'elettrone con elevata precisione.

IRC e SAC

Il calorimetro ad anello IRC è posizionato di fronte allo LKr intorno al fascio in modo da coprire la regione lasciata scoperta dal calorimetro elettromagnetico mentre il SAC si trova in fondo all'apparato sperimentale in modo da coprire gli angoli minori fino a zero. Prima del SAC si trova un dipolo magnetico utilizzato per deviare le particelle cariche del fascio in modo che non colpiscano il calorimetro. IRC e SAC sono esposti a fotoni di energie superiori a 5 GeV, e le loro inefficienze di rivelazione devono essere inferiori a 10^{-4} per raggiungere la soppressione del fondo $K^+ \to \pi^+\pi^0$ necessaria all'esperimento. L'IRC e il SAC utilizzano la tecnologia *shashlyk* basata su strati alternati di piombo e materiale plastico scintillante. Il fotone o elettrone entrante interagiscono con il piombo e sviluppano una cascata elettromagnetica; gli elettroni prodotti nella cascata producono una luce di scintillazione nel materiale plastico che può essere assorbita e riemessa con una lunghezza d'onda maggiore dalle fibre *wavelengths shifter* e rivelata da fotomoltiplicatori.

[3] L'energia E è espressa in GeV.

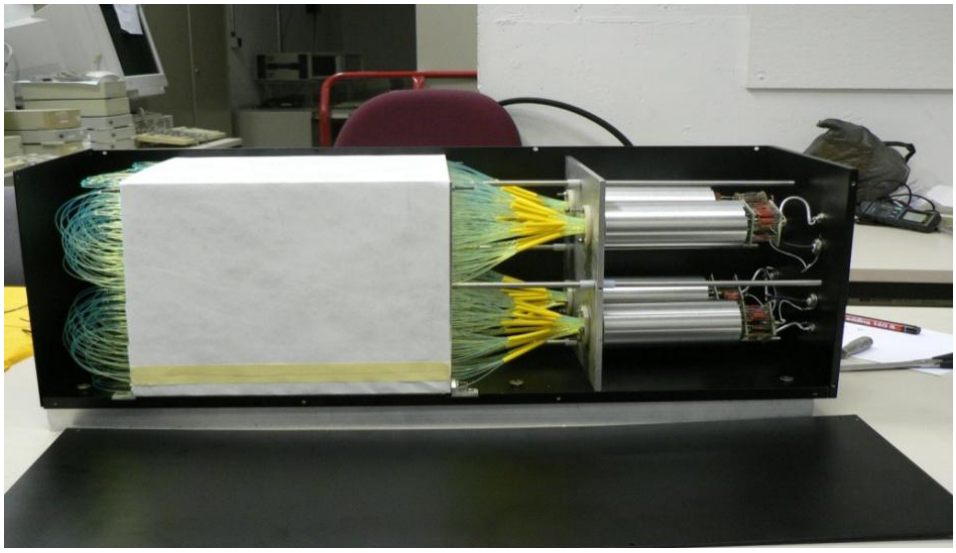


Figura 2.11: Prototipo di SAC realizzato nel 2006.

2.3.2 Il CHOD

Il CHOD (fig. 2.12) è un rivelatore di particelle cariche utilizzato principalmente per funzioni di *trigger* ovvero per misurare i tempi e i punti in cui sono passate le particelle cariche, per completare il RICH nella selezione delle tracce cariche nel *trigger* di livello 0 e per rivelare le reazioni foto-nucleari che possono avvenire all'interno dello specchio del RICH. La rivelazione di queste tracce cariche dovute alla reazioni foto-nucleari nel RICH è importante perché esse rischiano di ridurre l'efficienza di veto dei fotoni del LKr. Il CHOD può essere utilizzato anche durante l'analisi per garantire l'associazione della particella carica rivelata con la traccia del K^+ nel Gigatracker. A questo scopo deve avere un alta efficienza di rivelazione delle particelle cariche ed un eccellente risoluzione temporale (200 ps).

Temporaneamente verrà utilizzato come CHOD l'odoscopio già esistente in NA48, formato da un sistema di contatori a scintillazione su due piani ottagonali di superficie ~ 5.8 m^2 e spessi circa 0.15 lunghezze di radiazione e posizionato tra il RICH e il calorimetro LKr. I due piani contengono 64 contatori uno con le strisce posizionate in direzione orizzontale, l'altro in direzione verticale.

Come per il calorimetro LKr è risultato necessario rifare l'elettronica di lettura per permettere al CHOD di supportare un rate di particelle stimato intorno a 11 MHz.

2.3.3 Il MUV

Il MUV (Muon Veto) è un sistema di veto per i muoni composto da tre rivelatori chiamati MUV1, MUV2, MUV3. Oltre alla reiezione cinematica, alle camere a STRAW e al RICH sono necessari sistemi calorimetrici e di veto per i muoni per un'ulteriore reiezione dell'ordine di 10^5 del decadimento $K^+ \to \mu^+\nu$ che ha un *branching ratio* del 63.54 %. La maggior parte di questa reiezione può essere ottenuta con un veto sulle particelle che non depositano un significativo quantitativo di energia nei calorimetri e che attraversano una lastra sufficientemente spessa di materiale assorbitore. Per ottenere la reiezione richiesta però bisogna riconoscere anche i muoni che depositano una maggiore energia nel calorimetro per *Bremsstrahlung*. Per riconoscere questi eventi è necessario distinguere le cascate dovute ai muoni da quelle adroniche dei pioni attraverso misure della forma della cascata, che richiede una sufficiente segmentazione del sistema calorimetrico.

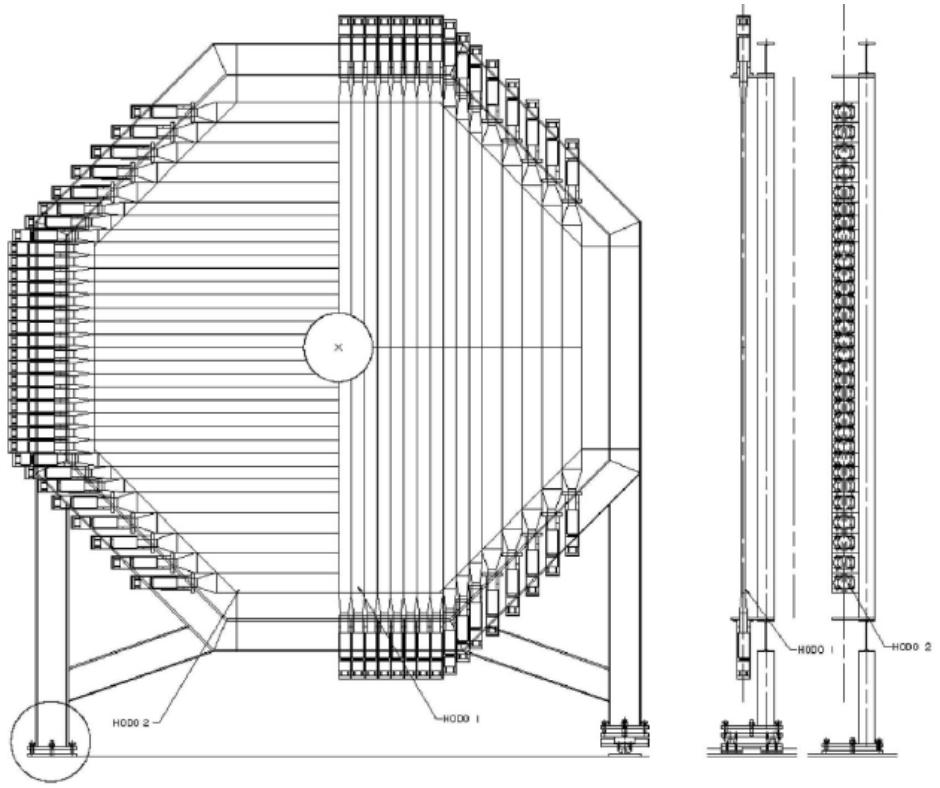


Figura 2.12: Vista schematica del CHOD di NA48, utilizzato nei test di NA62.

Per sopprimere gli eventi $K^+ \to \mu^+\nu$ di un fattore 20 già con il *trigger* di livello 0 è necessario anche un detector veloce di veto per i muoni con una risoluzione temporale inferiore ad 1 ns, in modo da rigettare gli eventi che presentano in coincidenza anche un segnale nel CHOD o nel RICH.

I primi due rivelatori (MUV1 e MUV2) (fig. 2.13) sono posizionati subito dopo il calorimetro elettromagnetico LKr e sono utilizzati come calorimetro adronico in modo da misurare l'energia rilasciata dalle particelle e la forma degli sciami. Mentre il MUV1 è un rivelatore nuovo, il MUV2 è il primo modulo del calorimetro adronico utilizzato nell'esperimento NA48. Non sono stati riutilizzati entrambi i moduli del calorimetro di NA48 perché il secondo modulo presentava scintillatori e PMT precedenti risalenti all'esperimento NA31 degli anni '80 ormai inefficienti; inoltre era necessaria una segmentazione trasversale più fine per il MUV1 per ottenere una maggiore distinzione tra le cascate elettromagnetiche dei muoni e quelle adroniche dei pioni. Infine lo spazio a disposizione per i MUV1 e MUV2 non era sufficiente a contenere entrambi i moduli preesistenti. Nel nuovo MUV1 la luce è raccolta usando delle fibre *wavelengths shifter* mentre nel MUV2 esistente la luce di scintillazione è incanalata attraverso delle guide di luce verso i fotomoltiplicatori che si trovano ai lati del modulo e che fuoriescono longitudinalmente di 80 cm. Per motivi di spazio il blocco del MUV2 sarà inserito ruotato di 180° in modo che le guide di luce e i fotomoltiplicatori circonderanno il MUV1. I due moduli hanno le facce ortogonali al fascio quadrate con lato di 2.6 m con un foro centrale di 10 cm di raggio per il passaggio della *beam pipe*, e sono composti rispettivamente da 25 e 23 lastre di ferro di 2.5 cm di spessore alternate con 24 e 22 strati di fibre scintillanti di 5 mm di raggio orientate sia orizzontalmente che verticalmente.

Il MUV3 (fig. 2.13) è il rivelatore veloce utilizzato per il *trigger* di livello 0 che abbiamo introdotto precedentemente: è posizionato dopo un blocco di ferro lungo 80 cm ed è composto

da due strati di materiale scintillante, per una larghezza totale di 5 cm, affacciate ai PMT per ottenere una buona risoluzione temporale.

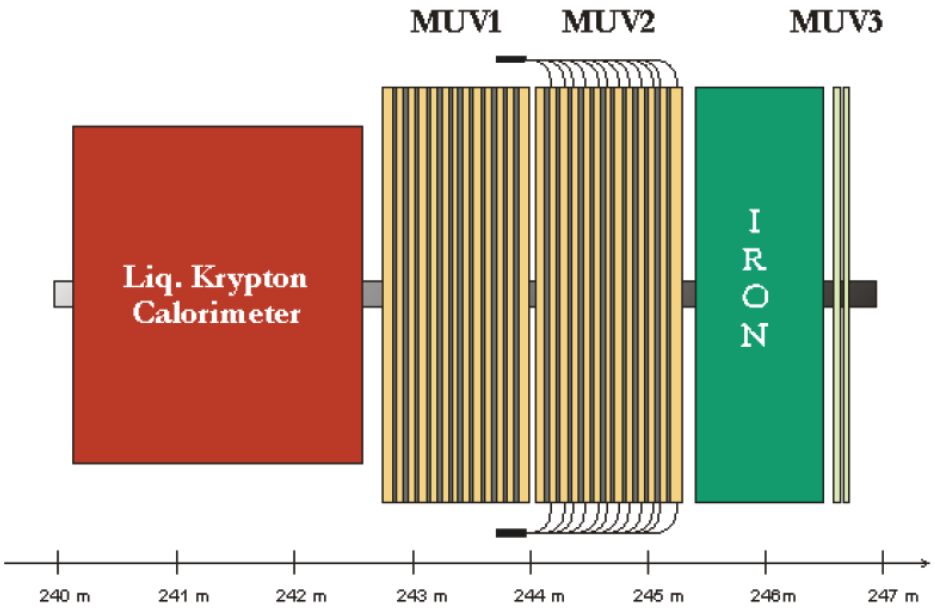


Figura 2.13: Struttura e posizione dei MUV.

2.3.4 Il RICH

Il RICH (Ring Imaging Čerenkov Counter) (fig. 2.14) è un rivelatore che può misurare la velocità e la direzione delle particelle cariche sfruttando la luce Čerenkov e combinato ad altri rivelatori permette l'identificazione delle particelle. Attraverso queste misure risulta utile a diversi scopi:

- separare i muoni dai pioni nell'intervallo di impulso tra 15 e 35 GeV/c che comporta una soppressione dei muoni almeno di un fattore 10^{-2};
- misurare il tempo di passaggio dei pioni con una risoluzione dell'ordine dei 100 ps;
- produrre un trigger di livello 0 per le particelle cariche;
- aiutare nella reiezione cinematica permettendo di eliminare gli effetti dovuti alle code non gaussiane dello spettrometro magnetico a camere a STRAW;
- aiutare nella reiezione di fotoni vicini alla *beam pipe*.

Per questi scopi, come compromesso tra il numero di fotoelettroni prodotti[4] e lo spazio disponibile per il RICH (18 m di lunghezza e 2.8 m di larghezza), è stato scelto come radiatore per la luce Čerenkov il neon a pressione atmosferica; per ottenere la risoluzione temporale richiesta saranno utilizzati veloci fotomoltiplicatori (PMT) a singolo anodo.

La soglia di impulso di particelle con massa m per emettere radiazione Čerenkov è $p_t = \frac{m}{\sqrt{n^2-1}}$ (dove n è l'indice di rifrazione del gas); per avere piena efficienza per pioni con 15 GeV/c di impulso è necessario che la soglia di emissione sia circa il 20% più piccola

[4] Il numero di fotoelettroni prodotti dipende dalla lunghezza L del radiatore secondo la formula $N_{p.e.} = N_0 L \sin^2 \vartheta_c$, in cui N_0 riassume il rendimento del RICH ed è chiamato fattore di qualità e ϑ_c è l'angolo a cui viene emessa la radiazione Čerenkov rispetto alla direzione della particella.

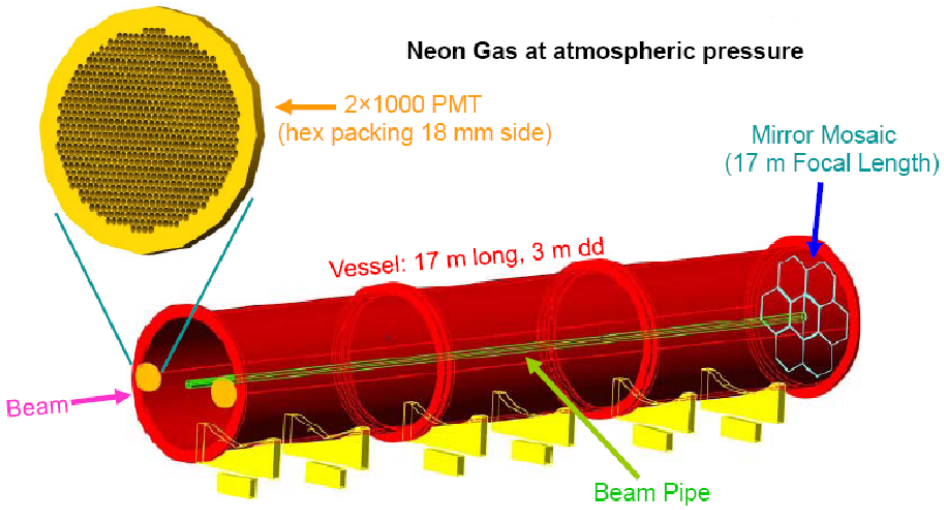


Figura 2.14: Schema della struttura del RICH.

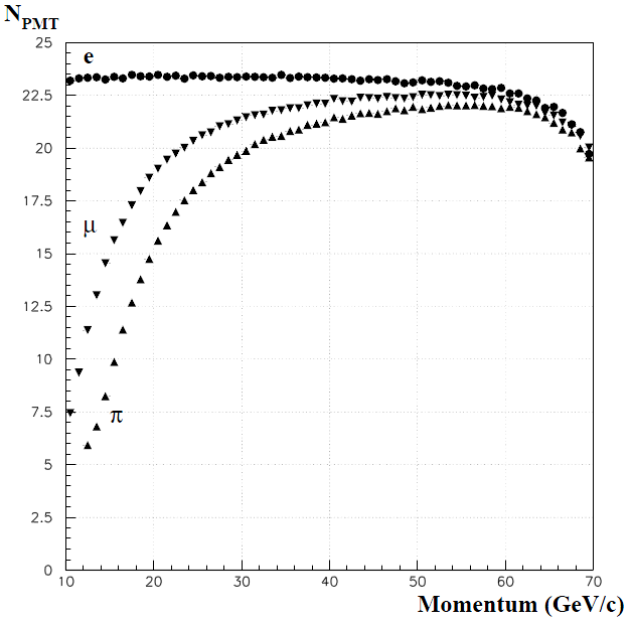


Figura 2.15: Valor medio del numero di fotomoltiplicatori colpiti per evento in funzione dell'impulso dei pioni, dei muoni e degli elettroni prodotti nel decadimento di un K.

ovvero 12.5 GeV/c che corrisponde ad un indice di rifrazione per il mezzo radiatore di $n = 1 + 62 \times 10^{-6}$, circa uguale all'indice di rifrazione del Neon a pressione atmosferica.

Nel RICH il cono di luce viene riflesso da uno specchio sferico, di lunghezza focale $f = 17$ m posto alla fine del rivelatore. L'immagine del cono sul piano focale, se la particella viaggia ortogonalmente a tale piano, è un cerchio di raggio $r_c = f \tan \vartheta_c$, quindi il raggio massimo del cerchio immagine si ha per l'angolo Čerenkov massimo ($\cos \vartheta_{max} = 1/n$) e vale $r_{max} = f\sqrt{n^2 - 1}$. Le particelle con traiettorie parallele e velocità diverse formano cerchi concentrici mentre se la direzione della particella è inclinata di un angolo α rispetto al piano focale il cerchio viene distorto in un ellisse ed il suo centro viene spostato di una quantità

pari a $f \tan\alpha$. Per questo motivo dalla posizione del centro del cerchio è possibile misurare la direzione delle particelle mentre dal suo raggio, e quindi dall'angolo Čerenkov, si ottiene la sua velocità che ci permette di ricavare la massa della particella se conosciamo l'impulso, oppure il suo impulso (fig. 2.16) se supponiamo che la particella sia un pione.

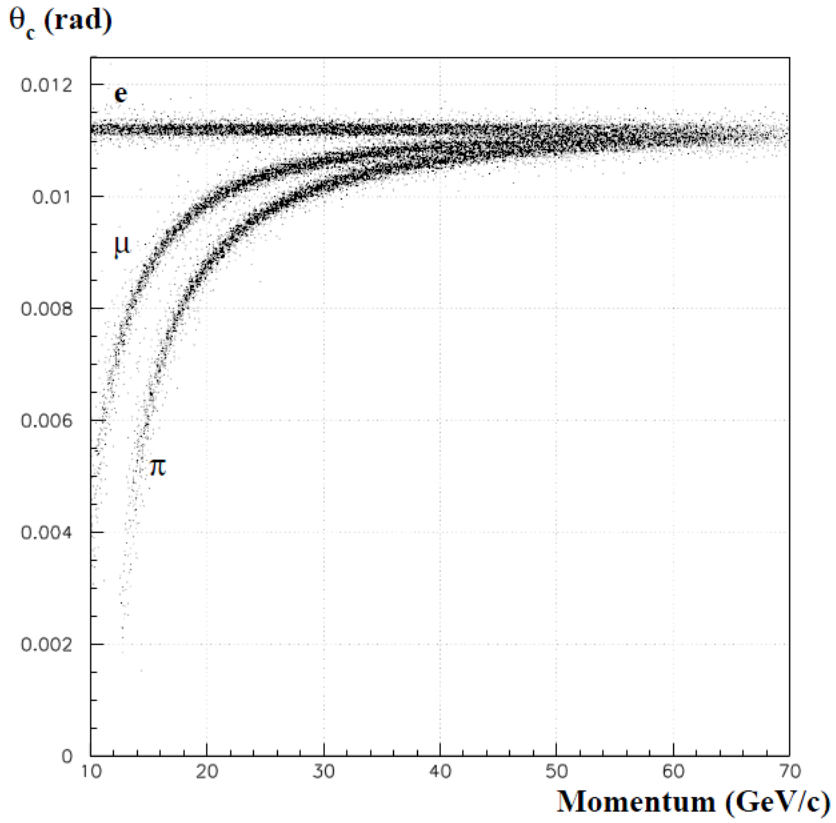


Figura 2.16: Angoli Čerenkov in funzione del momento dei pioni, muoni o elettroni per il RICH di NA62.

Utilizzando lo spettrometro magnetico a camere a STRAW per ottenere l'impulso p delle particelle, dalla misura dell'angolo Čerenkov possiamo calcolare la massa delle particelle per identificarle.

$$m^2_{rec} = p^2 \left(\frac{\cos^2 \vartheta_c}{\cos^2 \vartheta_{max}} - 1 \right)$$

In figura 2.17 sono mostrate le masse ricostruite per elettroni, muoni e pioni utilizzando tre impulsi (15, 25 e 35 GeV/c). Attraverso dei tagli tra i picchi dei pioni e delle altre particelle è possibile determinare il fattore di reiezione dei fondi contenenti muoni o elettroni e le perdite di pioni. Ad esempio con una selezione dei pioni, integrata negli impulsi tra 15 e 35 GeV/c, si ha un fattore di soppressione dei muoni di $\sim 1.3 \times 10^{-3}$.

Sul piano focale sono poste due regioni (fig. 2.18), contenenti ciascuna circa 1000 fotomoltiplicatori, per raccogliere la radiazione Čerenkov riflessa dallo specchio. Ogni fotomoltiplicatore ha un area attiva larga circa 8 mm con davanti un cono di Winston per incanalare nel PMT la radiazione Čerenkov che colpisce una zona di 18 mm. I centri delle due regioni contenenti i fotomoltiplicatori sono distanti 1.2 m dall'asse del fascio.

Lo specchio sferico utilizzato nel RICH non è composto da un unico pezzo di superficie 6 m^2 ma da un mosaico di 20 specchi in modo da avere la possibilità di riflettere la radiazione

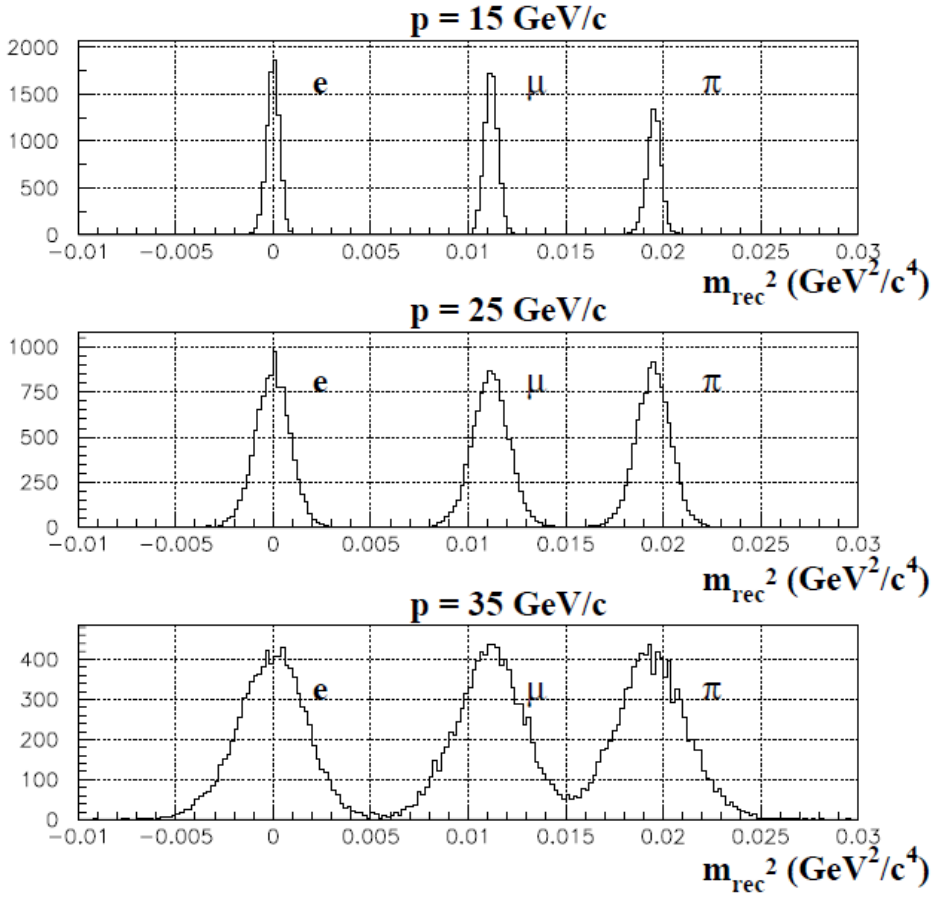


Figura 2.17: Massa ricostruita al quadrato per tre impulsi, con i picchi dovuti agli elettroni, muoni e pioni ben visibili.

Čerenkov nelle due zone contenenti i PMT e di evitare la *beam pipe*. Ogni specchio è allineato in modo indipendente dagli altri, ha una lunghezza focale di 17m, è formato da una strato di 25 mm di vetro rivestito da un sottostrato di alluminio ed è rivestito da una pellicola fine dielettrica a scopo di protezione e per aumentare la riflettività.

2.3.5 Spettrometro Magnetico a Camere a STRAW

Lo scopo dello spettrometro magnetico è di misurare accuratamente l'impulso e la direzione delle particelle secondarie prodotte nel decadimento dei K^+. Per utilizzare i vincoli cinematici utili a rigettare la maggior parte del fondo, è necessario che le tracce del K^+ e delle particelle secondarie siano ricostruite accuratamente. Con lo spettrometro magnetico, quindi, si deve ottenere una buona risoluzione sia sull'impulso delle particelle ($\frac{\Delta p}{p} \leq 1\%$) che sulla loro direzione, in modo da ottenere una precisione sull'angolo tra la direzione del pione e del K^+ inferiore a 60 μrad. Inoltre per una buona riuscita dell'esperimento lo spettrometro magnetico deve rispettare alcune necessità sperimentali:

- l'uso di materiale ultra-leggero lungo le traiettorie delle particelle per minimizzare la diffusione multipla;

- integrazione dello spettrometro magnetico all'interno del tubo a vuoto;

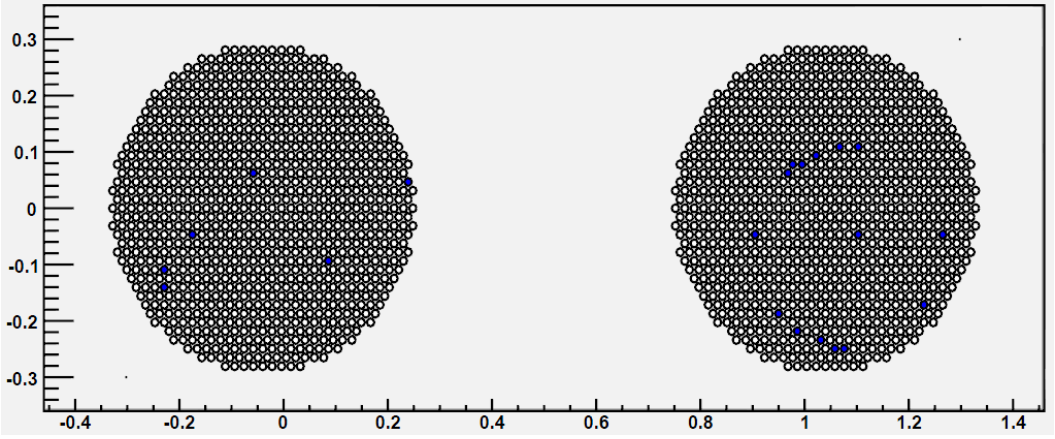


Figura 2.18: Esempio di cerchio Čerenkov come visto nelle due regioni contenenti i fotomoltiplicatori, ottenuto da una simulazione Monte Carlo.

- una risoluzione spaziale che permetta una ricostruzione precisa del vertice di decadimento;
- un efficienza media sulle tracce vicina al 100%;
- la capacità di vetare gli eventi con più particelle cariche.

Queste condizioni hanno delineato le principali richieste per il rivelatore:

- la risoluzione spaziale delle coordinate del punto di impatto delle particelle deve essere inferiore a 130 μm;
- ogni camera deve avere uno spessore equivalente inferiore a 0.5% della lunghezza di radiazione;
- deve essere istallato all'interno del tubo a vuoto, che ha una pressione di 10^{-5} mbar, con la minima fuoriuscita di gas;
- le regioni vicine al fascio devono sopportare un alto rate di *hit* (fino a 500 kHz/*straw*);
- deve poter contribuire al *trigger* come veto di molteplicità.

Le camere dello spettrometro sono costituite da tubi a deriva (*straw*) posizionati nel vuoto. La scelta dell'utilizzo di camere a STRAW come spettrometro magnetico è dovuta alla necessità di avere un buona risoluzione spaziale del vertice di decadimento e, quindi, di minimizzare la diffusione multipla delle particelle provenienti dal decadimento del K^+. Per questo motivo non è stato progettato uno spettrometro magnetico simile a quello adoperato in NA48 [9, 24] costituito da 4 camere a deriva, che avevano un efficienza di rivelazione molto vicina al 100% e una risuzione spaziale nella ricostruzione delle posizioni degli hit di circa 100μm.

Utilizzando le camere a STRAW come spettrometro magnetico in NA62 non si va incontro alla diffusione multipla delle particelle ad opera del gas presente nelle camere e delle flange, che in NA48 separavano la *beam pipe* dalle camere. Inoltre un altro importante vantaggio di operare nel vuoto è dovuto alla conseguente riduzione del fondo che poteva originarsi dalle particelle dell'alone del fascio in seguito all'attraversamento delle flange e del gas nelle camere.

Lo spettrometro (fig. 2.19) è posizionato prima del RICH, a 183 m dal bersaglio di produzione, e composto da quattro camere, le prime tre distanti 10 m tra loro mentre la

quarta distante 15 m dalla terza, con un dipolo magnetico posto tra la seconda e la terza camera che genera un campo magnetico verticale di 0.36 T e che fornisce alle particelle un impulso trasverso di 270 MeV/c.

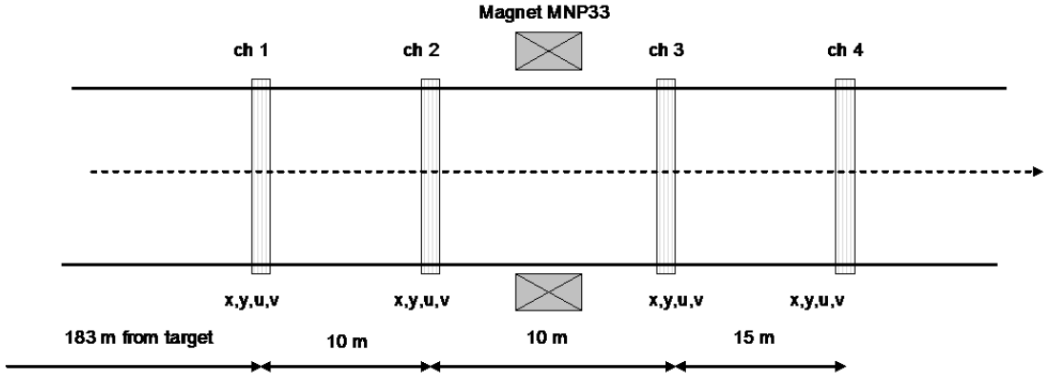


Figura 2.19: Vista schematica dello spettrometro magnetico e delle sue quattro camere.

In ogni camera sono presenti 1792 tubi *straw* distribuiti su quattro viste, in modo da misurare la posizione della traccia con quattro coordinate (x,y e u,v orientate a 45° rispetto alle prime due (fig. 2.20)). Tali camere sono misurate in ogni camera da quattro piani (112 *straw* per piano) per risolvere l'ambiguità destra-sinistra nella ricostruzione della coordinata dell'*hit*. In ogni vista la regione centrale, larga circa 12 cm e corrispondente alla zona di passaggio del fascio di particelle, non presenta fili (fig. 2.20) comportando una perdita di accettanza inferiore al 10 %. La presenza di questa zona centrale senza *straw* implica una suddivisione del piano in quattro zone distinte a seconda del numero di viste (fig. 2.21) che le coprono. Sono presenti, infatti, alcune regioni intorno al foro centrale per il fascio che sono coperte da 1 sola vista, mentre altre sono coperte da 2-3 o 4 viste. Questo è un aspetto importante da tenere in conto durante la ricostruzione delle tracce, in quanto con una sola vista, o con due se una non presenta *hit* per inefficienze, non è possibile ricostruire la posizione del passaggio della particella nel piano; per questo motivo è stato necessario utilizzare 4 viste e non solo 2. Da una simulazione Monte Carlo (fig. 2.22) risulta che nella quarta camera la frazione di eventi di segnale in cui il π^+ attraversa la zona coperta da 1 sola vista è il 2%, da 2 viste è 10%, da 3 viste è il 50% mentre da 4 viste è il 38 %. Le *straw* sono lunghe 2.1 m, hanno un raggio di 4.9 mm e i tubi sono costruiti con sottili fogli di PET (polyethylene terephthalate) da 36 μm di spessore rivestiti all'interno da due strati sottili di metallo[5] necessari a garantire la conduttività elettrica al catodo. All'interno della *straw* è posto un filo di tungsteno placcato d'oro di 15 μm di raggio che svolge la funzione di anodo.

Le *straw* di ogni piano sono distanziate tra loro di 17.6 mm (fig. 2.23); i primi due piani e gli ultimi due sono posizionati a 11 mm di distanza mentre tra il secondo e il terzo ci sono 15 mm. Inoltre in ogni vista le *straw* del secondo, terzo e quarto piano sono sfasate rispetto a quelle del primo piano rispettivamente di +8.8, +4.4 e -4.4 mm. Questa configurazione delle *straw* è necessaria per tenere sotto controllo eventuali deformazioni dovute alla pressione e per avere sempre almeno due *straw* attraversate dalla particella incidente.

Per la scelta del gas da utilizzare all'interno delle *straw* è stata utilizzata una simulazione in cui venivano confrontate due misture: la prima è composta dal 90% di CO_2, 5% di isoC_4H_{10} e 5% di CF_4 mentre la seconda è costituita dal 70% di Ar ed il 30% di CO_2 [3]. La risoluzione spaziale delle due misture di gas è stata calcolata nel punto di lavoro, scelto in modo che l'efficienza per entrambe le *straw* non fosse inferiore al 99% a una distanza dal filo di 3 mm, ed è risultata $\sim 40\mu$m per il primo gas e $\sim 80\mu$m per il secondo. Ai fini della ricostruzione delle tracce nello spettrometro è importante considerare una altro

[5]Il primo strato è formato da 0.05 μm di rame ed il secondo da 0.02 μm d'oro.

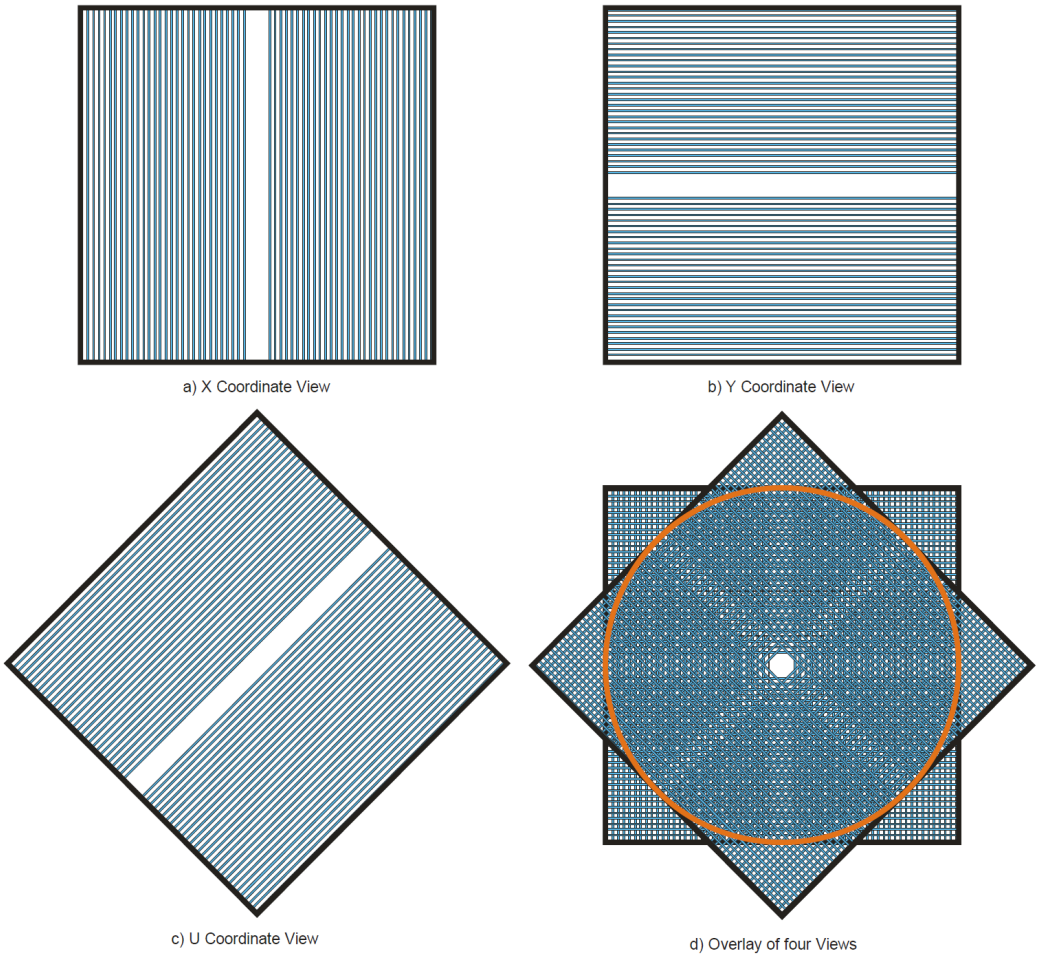


Figura 2.20: Schema delle viste che compongono le camere: (a) la vista della coordinata x con *straw* verticali, (b) la vista della coordinata y con *straw* orizzontali, (c) la vista della coordinata u (la vista della coordinata v è uguale ma ruotata di 90°) con *straw* oblique e la proiezione di tutte le viste di una camera in cui si nota il buco centrale necessario a lasciare libero il passaggio del fascio.

parametro per il confronto tra le due misture di gas, ovvero la risposta temporale delle *straw* al passaggio di una particella.

Una particella carica, al suo passaggio all'interno delle *straw*, ionizza il gas lungo il suo cammino. Gli elettroni creati lungo la traccia della particella si muovono verso l'anodo con una velocità media di deriva di ~ 3 cm/μs. Quando gli elettroni arrivano in una regione intorno al filo entro un raggio dell'ordine di 300 μm hanno guadagnato un energia sufficiente dal campo elettrico accelerante da riuscire a ionizzare le molecole del gas e creare una valanga. Il segnale sul filo (anodo) inizia con la formazione delle prime valanghe, dovute agli elettroni provenienti dalla regione meno distante dal filo in cui è passata la particella, e finisce dopo l'arrivo delle ultime valanghe, generate dagli elettroni provenienti dalle regioni più lontane dal filo ovvero vicino alle pareti della *straw*. La fine del segnale quindi è sempre dovuta agli elettroni che percorrono una distanza pari al raggio della *straw*; il tempo impiegato da questi elettroni ad arrivare sull'anodo è chiamato *trailing time* (fig. 2.24) ed equivale alla differenza di tempo tra il passaggio della particella ed il tempo finale del segnale. Visto che il *trailing time* è il tempo di arrivo degli elettroni all'anodo da una distanza pari al raggio, il

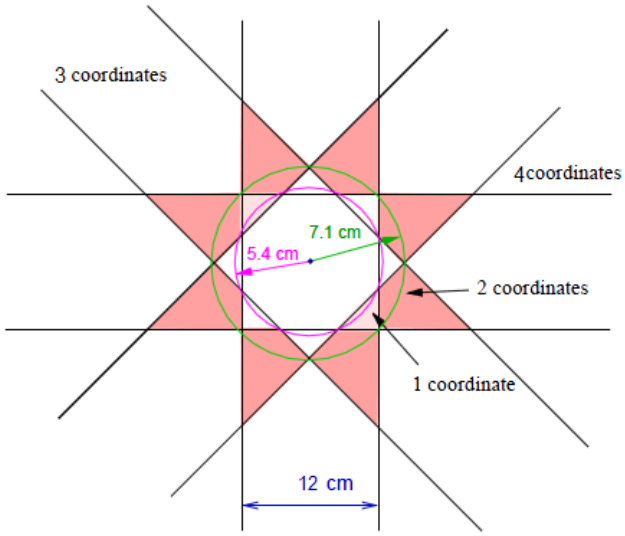


Figura 2.21: Suddivisione del piano nelle 4 zone a seconda del numero di viste che presentano un *hit*.

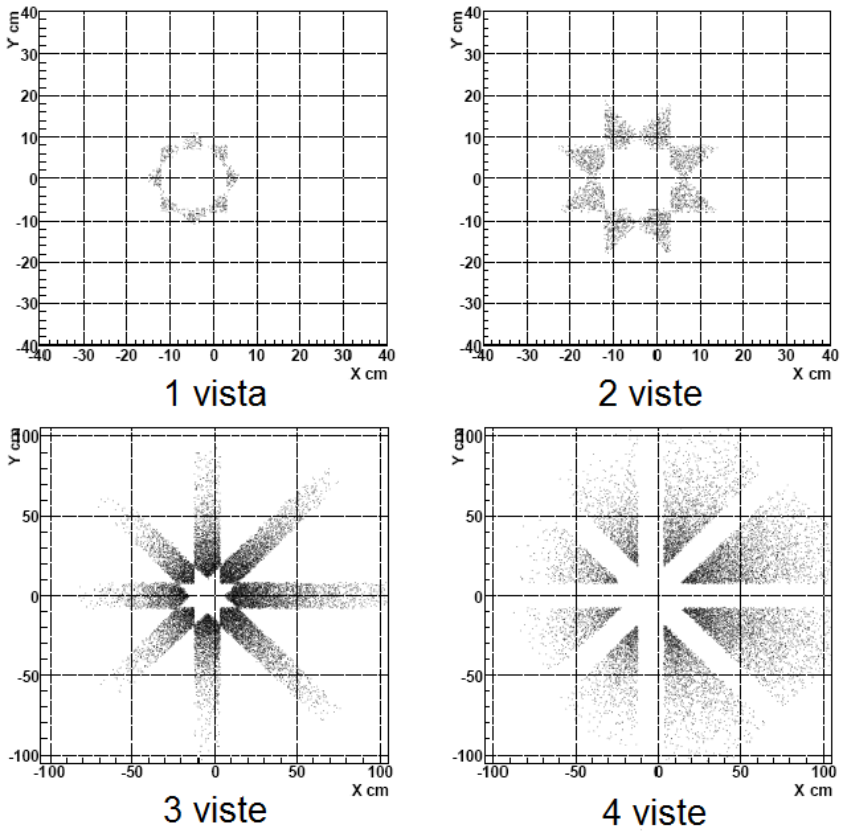


Figura 2.22: Distribuzione del punto d'impatto nella quarta camera del π^+ di un evento di segnale al variare del numero di viste che presentano un *hit*.

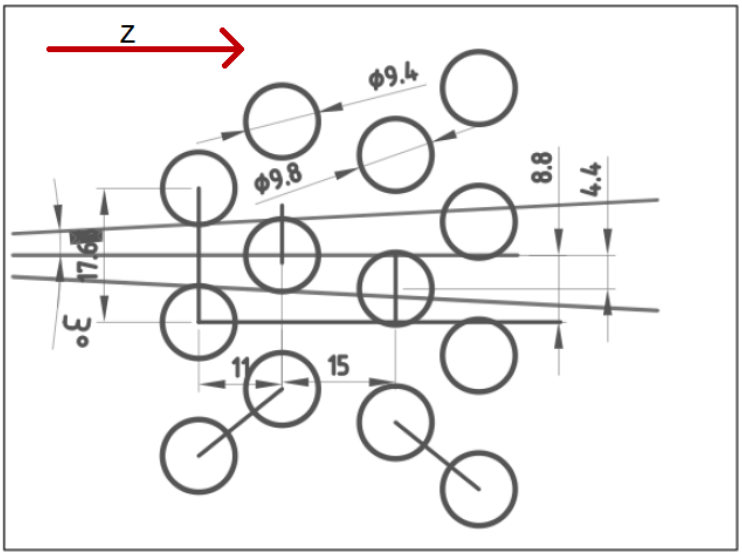


Figura 2.23: Disposizione delle *straw* in ogni vista; la direzione del fascio è da sinistra a destra.

suo valore è sempre lo stesso a meno dell'errore di misura. La differenza tra il tempo del passaggio della particella e l'arrivo delle prime valanghe, cioè l'inizio del segnale, si chiama *leading time*. Il valore del *leading time* (fig. 2.24) dipende dalla minima distanza dal filo a cui è passata la particella e nelle camera a STRAW di NA62 varia tra 0 e 160 ns.

Visto che la finestra temporale delle camere dipende dal massimo valore che possono raggiungere il *leading time* ed il *trailing time*, questi valori risultano importanti nella scelta del gas da utilizzare all'interno delle *straw*, in quanto una finestra temporale più grande comporta un maggior sovrapposizione di *hit* dovuti ad eventi diversi[6]. In figura 2.25 è mostrato il confronto delle prestazioni temporali dei due gas (il primo composto dal 90% di CO_2, 5% di isoC_4H_{10} e 5% di CF_4 mentre il secondo dal 70% di Ar e dal 30% di CO_2) con i grafici del *leading time* in funzione della distanza tra la traiettoria della particella ed il filo all'interno della *straw*. Osservando i due grafici (fig. 2.25) si nota che il gas composto da Argon-Etano è più veloce, in quanto il massimo *leading time* con questo gas risulta essere circa il 40% del massimo *leading time* che si ottiene con l'altra misura. Per questo motivo è stato scelto il primo gas nonostante l'inferiore risoluzione spaziale che ne deriva.

In questo lavoro di tesi abbiamo simulato la risposta temporale delle *straw* utilizzando un'approssimazione lineare (fig. 2.26) per gli andamenti del *leading time* in funzione della distanza tra la traiettoria della particella ed il filo all'interno della *straw*, fissando inoltre un *leading time* massimo leggermente superiore (160 ns) a quello risultante nella simulazione trattata precedentemente (fig. 2.25) per il gas composto dal 70% di Ar e dal 30% di CO_2 ($\sim$ 125 ns).

[6]Una più completa descrizione di questo comportamento e dei suoi effetti è presente nel capitolo 5.

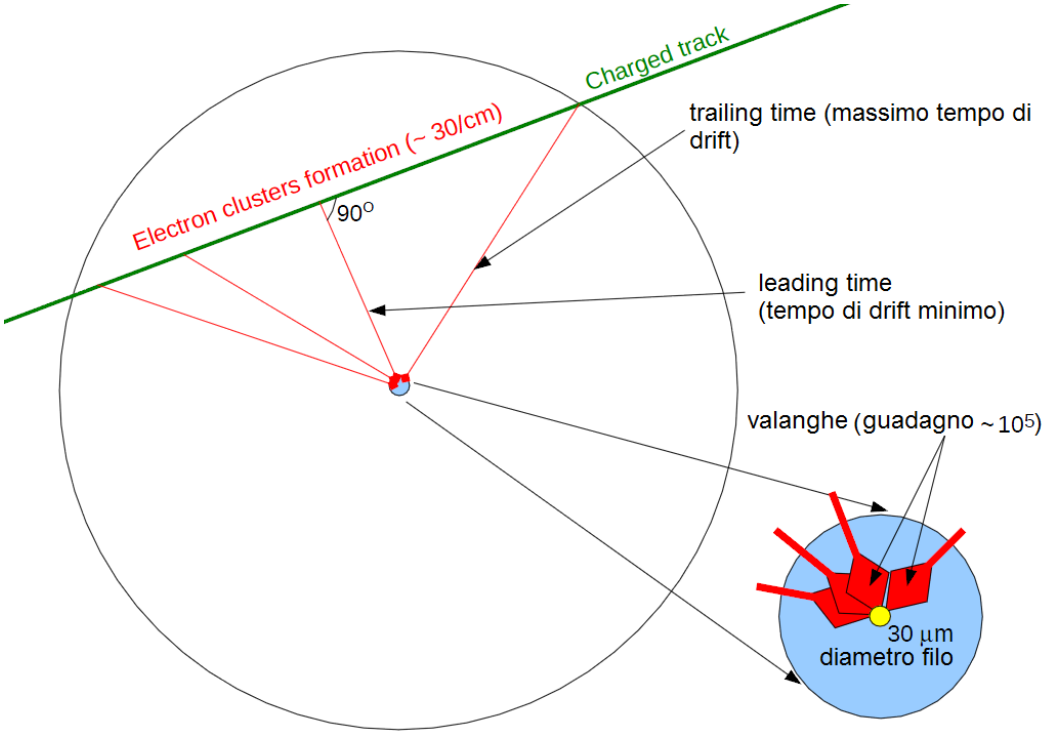
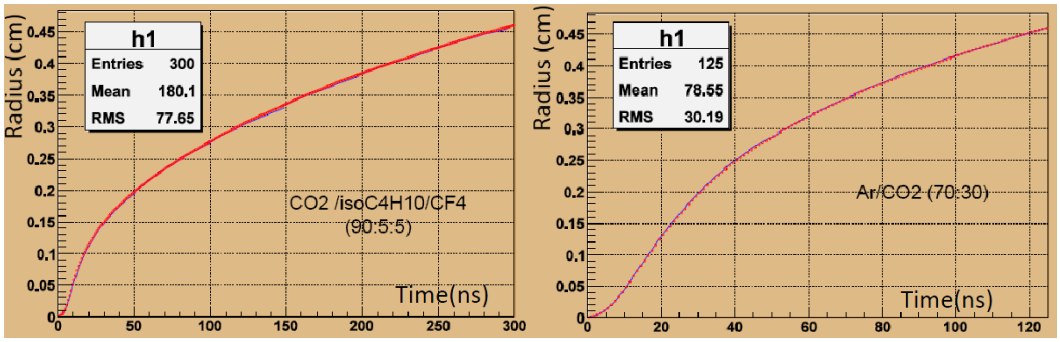


Figura 2.24: Schema semplificato di una *straw* e della formazione del segnale.



(a) (b)

Figura 2.25: Dipendenza temporale dal raggio per i due gas confrontati.

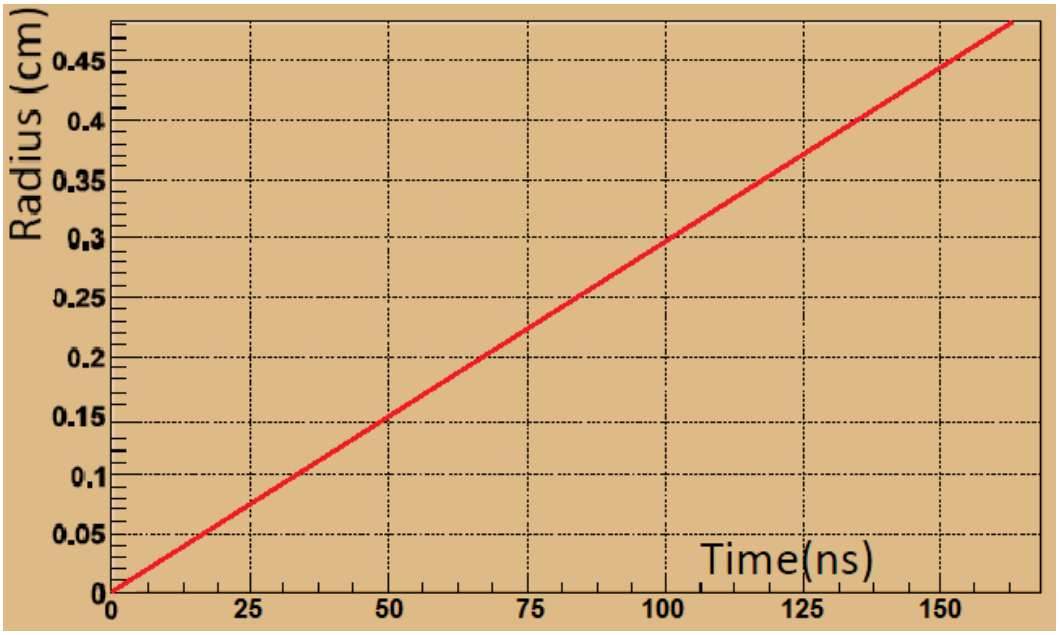


Figura 2.26: Dipendenza del *leading time* della distanza tra la traiettoria della particella ed il filo all'interno della *straw* utilizzata nel capitolo 5; con una distanza dal filo di 0.44 cm si ottiene un leading time di 150 ns.

2.4 Trigger e Sistema di Acquisizione Dati

L'alto *rate* di eventi e la presenza di numerosi rivelatori nell'esperimento NA62 comporta un grande volume di dati in uscita che è impossibile da raccogliere senza filtraggio per un sistema di acquisizione dati; per questo motivo è necessaria la presenza di un *trigger* che riduca la quantità di dati da immagazzinare. Il *trigger* deve identificare velocemente quali sono gli eventi non interessanti e rigettarli.

Il sistema di *trigger* e di acquisizione dati (TDAQ) di NA62 [3] in genere coincide con il sistema di *readout* e di processamento online, permettendo così di avere un maggiore controllo del *trigger*, che usa gli stessi dati disponibili al *readout*, e una maggiore flessibilità dovuta alla riduzione di una parte dei dati per mezzo del software. Per il TDAQ si è cercato, dove si è potuto, di riutilizzare le sistemi già esistenti e di trovare soluzioni adatte alla maggior parte dei rivelatori.

La struttura generale del TDAQ di un esperimento di alte energie moderno è sviluppata su più livelli: i dati provenienti dai rivelatori vengono immagazzinati su dispositivi di archiviazione dedicati soltanto se rispondono ai requisiti posti dai vari livelli controllati in modo sequenziale. In particolare il sistema di *trigger* di NA62 è composto da 3 livelli che devono ridurre il rate di eventi da immagazzinare da 10 MHz a qualche kHz. Il primo livello del *trigger* (livello 0) è completamente hardware e deve prendere una decisione sull'evento in tempi inferiori a 100 ms. Il livello 1 invece sarà software e dovrà prendere decisioni basandosi sulle quantità ricostruite dai singoli rivelatori anche se è possibile utilizzate delle operazioni logiche tra i risultati ottenuti con i vari rivelatori. Anche il livello 2 sarà software però prenderà una decisione sull'intera ricostruzione dell'evento utilizzando i dati forniti da tutti i rivelatori.

Il trigger di L0 e' basato sulle informazioni ottenute dai rivelatori a risposta più rapida, e consiste essenzialmente nell'identificazione positiva di una traccia da parte dell'odoscopio (CHOD), senza segnali nel piano 3 del MUV né rilasci di energia nel LKr incompatibili con

quelli attesi per un π^+, in stretta coincidenza temporale (pochi ns). Le camere a STRAW, a causa della loro lentezza di risposta, non possono essere utilizzate a questo livello di trigger, e possono giocare un ruolo importante ai livelli software successivi.

CAPITOLO 3

LE GPU COME TRIGGER

Indice

Negli ultimi anni è cresciuto notevolmente l'utilizzo di schede grafiche commerciali "*Graphic Processing Unit*" (GPU) per il calcolo scientifico. Nonostante le GPU siano state progettate per la grafica tridimensionale e vengano prodotte in grande numero principalmente per il mercato dei videogiochi, la loro struttura parallela di elaborazione dei dati e la loro potenza di calcolo può essere utilizzata in numerose applicazioni in campo scientifico (ad esempio in fisica medica, astrofisica, meccanica quantistica, chimica molecolare, ecc [37]). Per questi motivi negli ultimi anni si è evoluto un settore della ricerca informatica, chiamato *General-Purpose computing on Graphics Processing Units* (GPGPU), che ha lo scopo di utilizzare le GPU per calcoli in cui è necessaria una grande potenza di elaborazione. Ad esempio sono state utilizzate GPU per calcoli di dinamica molecolare come il calcolo della traiettoria di un sistema di particelle puntiformi in risposta ad una funzione potenziale [5] o calcoli di fluidodinamica [11].

Rispetto alle CPU (Central processing unit) dei comuni elaboratori, le GPU hanno una struttura differente (fig. 3.1) con molta più superficie dedicata alle unità di calcolo piuttosto che alle strutture di controllo del flusso e a quelle adibite al *caching*, rendendo il dispositivo più specializzato al calcolo altamente parallelizzato. Le ultime GPU arrivano a superare i 2 TeraFLOPS(FLoating Point Operations Per Second) di potenza di calcolo ed i 100 GB/s di larghezza di banda di accesso alla memoria principale delle schede. Da un confronto tra le GPU e le CPU (fig. 3.2) si nota che le prestazioni delle GPU negli ultimi anni superano di un fattore 10 quelle delle CPU e l'andamento è ancora in forte crescita.

3.1 Struttura delle GPU

Nonostante esistano molti modelli di GPU con differenti caratteristiche, in generale tutte le unità recenti seguono la stessa architettura che viene descritta in questo capitolo riferendosi in particolare alle schede AMD della famiglia Evergreen. Le GPU, essendo schede grafiche,

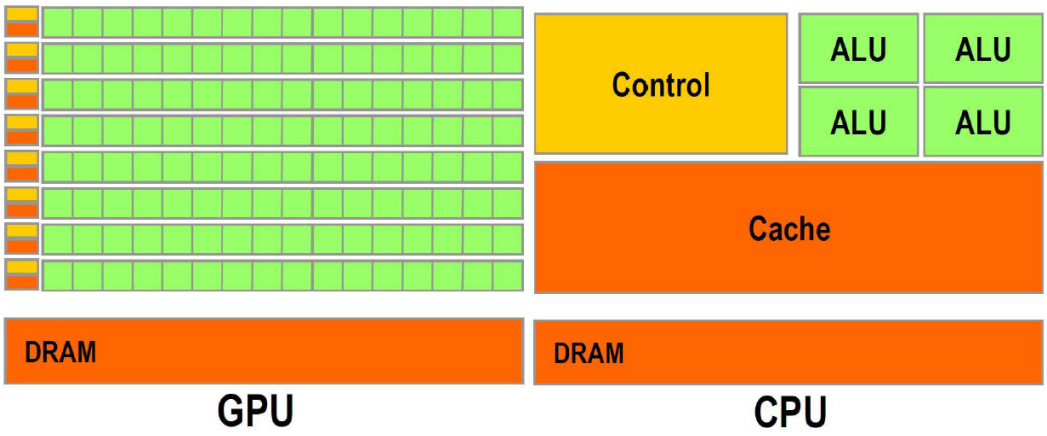
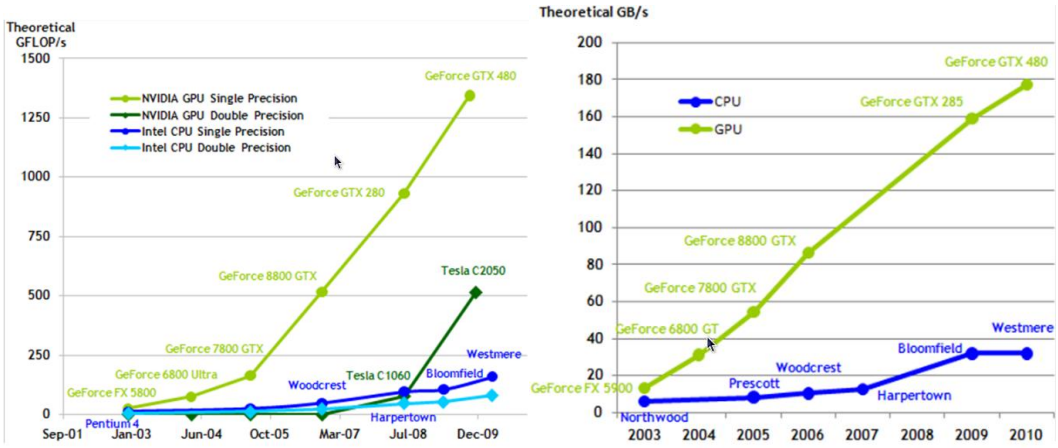


Figura 3.1: Confronto tra l'architettura di una GPU e di una CPU in termini di area di silicio.



(a) (b)

Figura 3.2: Confronto tra la potenza di calcolo (a) e la larghezza di banda per l'accesso alla memoria (b) delle GPU NVIDIA e delle CPU Intel nel decennio 2000-2010.

tipicamente devono essere inserite in un PC, sulla cui CPU, dotata di software e *driver* appositi, compila il programma per la GPU (*kernel*), lo trasferisce su essa insieme ai dati, ne lancia l'esecuzione e infine recupera i risultati con una seconda copia di dati.

Le GPU sono composte da più *Compute Unit* (fig. 3.3) ovvero strutture che raggruppano al loro interno molti processori chiamati *stream core* e che si occupano dell'esecuzione delle molteplici istanze del *kernel* [4]. É possibile far eseguire alla GPU una sola istanza del *kernel* (nella nostra applicazione questo può corrispondere ad esempio a processare un evento per volta, tutti i *Compute Unit* processano lo stesso evento) oppure più istanze (in modo da processare più eventi contemporaneamente). Utilizzando più istanze del *kernel* è possibile far processare contemporaneamente alla GPU un numero di eventi pari al numero di *Compute Unit* della scheda (se per semplicità ammettiamo che il *kernel* sia scritto per processare un solo evento).

Gli *stream core* sono l'unità fondamentale di calcolo responsabile delle operazioni su numeri interi e *float* a singola e doppia precisione e delle operazioni trascendenti (come seno, coseno e logaritmo). Queste operazioni sono eseguite dai *processing element* contenuti all'interno

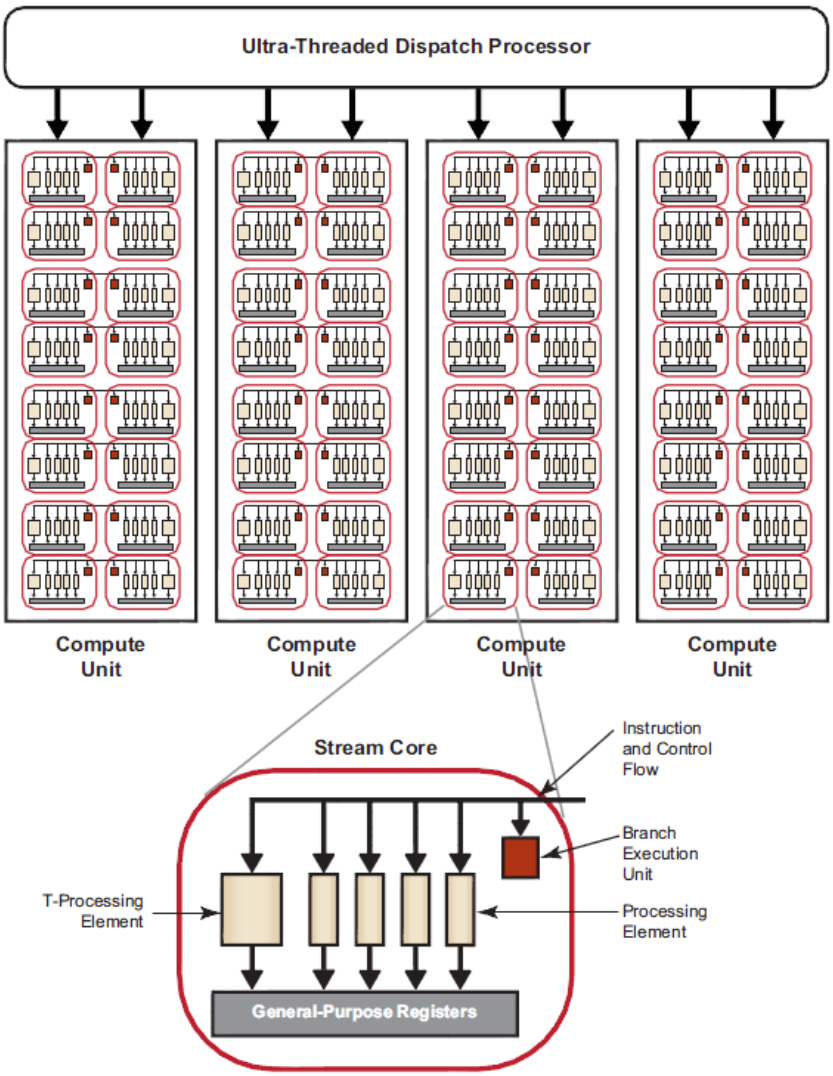


Figura 3.3: Diagramma semplificato della struttura di una GPU AMD Evergreen.

dello *stream core* (fig. 3.3). I *processing element* sono disposti all'interno di uno stream core come un processore per Very Long Instruction Word (VLIW) ovvero un istruzione in cui sono raggruppate fino a cinque operazioni scalari che saranno eseguite ognuna da uno dei *processing element*, cinque per ogni *stream core* nell'architettura Evergreen della AMD. In questa architettura uno dei cinque *processing element*, il *T-processing element*, può anche eseguire operazioni trascendenti mentre le operazioni in doppia precisione sono processate attraverso la connessione di due o quattro *processing element*. Gli *stream core* contengono anche un'unita predisposta alla gestione delle istruzioni ramificate come le istruzioni *if and else* o i cicli *for*. Tutti gli *stream core* di una *Compute Unit* eseguono la stessa sequenza di istruzioni contemporaneamente.

La scheda usata in questo lavoro di tesi è una ATI[1] Radeon HD 5970, una doppia GPU (corrispondente a due unità HD5870) contenente 40 *Compute Unit*, 640 *stream core* e 3200 *processing element* che permettono di raggiungere un picco di potenza di calcolo di 4640

[1]La ATI Technologies è stata assorbita nel 2006 dalla Advanced Micro Devices (AMD), il marchio ATI è stato dismesso nel 2010.

Gflops, ed arriva a sostenere una larghezza di banda per la memoria principale di 256 GB/s. Nella figura 3.4 sono mostrate tutte le specifiche della ATI Radeon HD 5970.

Product Name (ATI Radeon™ HD)	5970	**Peak GPU Bandwidths**	
Engine Speed (MHz)	725	Register Read (GB/s)	22272
Compute Resources		LDS Read (GB/s)	3712
Compute Units	40	Constant Cache Read (GB/s)	7424
Stream Cores	640	L1 Read (GB/s)	1856
Processing Elements	3200	L2 Read (GB/s)	742
Peak Gflops	4640	Global Memory (GB/s)	256
Cache and Register Sizes		**Global Limits**	
# of Vector Registers/CU	16384	Max Wavefronts / GPU	992
Size of Vector Registers/CU	256 kB	Max Wavefronts / CU (avg)	24.8
LDS Size/ CU	32k	Max Work-Items / GPU	63488
LDS Banks / CU	32	**Memory**	
Constant Cache / GPU	96k	Memory Channels	2 x 8
Max Constants / CU	8k	Memory Bus Width (bits)	2 x 256
L1 Cache Size / CU	8k	Memory Type and Speed (MHz)	GDDR5 1000
L2 Cache Size / GPU	2 x 512k	Frame Buffer	2 GB

Figura 3.4: Schema con le specifiche della GPU AMD Radeon HD 5970. La *local memory* è chiamata, in questo schema, LDS. Le specifiche più importanti della scheda sono la sua frequenza (*engine speed*), le risorse di calcolo (*compute resource*) e i picchi di banda delle varie memorie (*Peak GPU Bandwidths*).

Tuttavia si deve considerare solo la metà di questi valori perché fino ad ora il programma AMD Stream SDK, utilizzato per eseguire i *kernel* nella GPU, non permette l'uso di entrambe le schede delle GPU doppie.

3.2 Architettura di OpenCL

Il *kernel* da eseguire sulle GPU AMD e il programma *host* devono essere scritti con un implementazione del linguaggio OpenCL per la tecnologia AMD. OpenCL è un ambiente per la programmazione parallela che include un linguaggio, API (Application Programming Interface), librerie ed un *runtime system*, che comprende un compilatore e un *linker*, per consentire lo sviluppo del *software*. Questo linguaggio è implementato anche nelle GPU NVIDIA, per le quali esiste anche un linguaggio più specifico chiamato CUDA creato dalla NVIDIA stessa. Un programma in OpenCL consiste in due parti: uno o più *kernel*, eseguiti sui *device* OpenCL (CPU o GPU), e un programma *host*, eseguito sulla CPU, che definisce il "contesto" per i *kernel* e organizza la loro esecuzione.

Ogni istanza del *kernel* è chiamata *work-item* ed è identificata da un indice globale. Ogni *work-item* esegue lo stesso codice, però all'interno di questo ci possono essere delle istruzioni che devono essere eseguite solo da alcuni *work-item*[2] specifici, se il programmatore lo desidera.

I *work-item* sono raggruppati in *work-group* a cui è assegnato un altro indice che contraddistingue anche i *work-item* che contiene. L'utilizzo di *work-group* e *work-item* permette di ottenere una parallelizzazione su più livelli, ad esempio nella nostra applicazione è possibile

[2]La selezione dei *work-item* che eseguono un istruzione o un gruppo di istruzioni può avvenire attraverso l'utilizzo dell'istruzione *if* con una condizione sull'indice globale che identifica i *work-item*.

eseguire lo stesso programma su più eventi facendo processare ogni evento da un diverso *work-group* mentre i *work-item* si occupano della parallelizzazione interna del programma.

Lo spazio N-dimesionale[3] degli indici in cui sono raggruppati i *work-group* si chiama *NDRange* (fig. 3.5) che contiene quindi l'insieme di tutte le istanze del *kernel* che vengono eseguite sulla GPU.

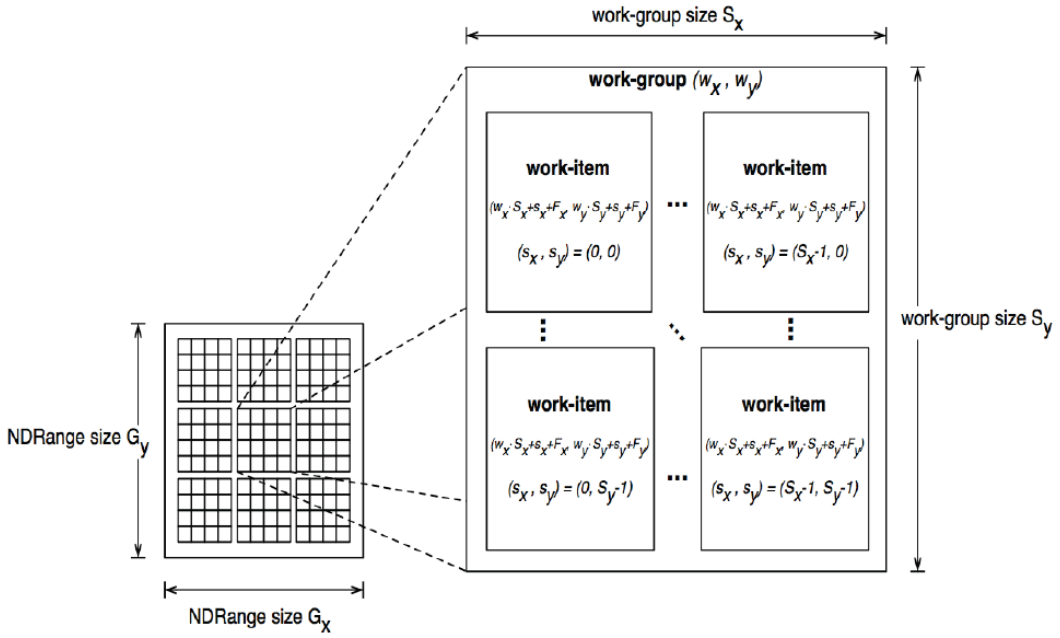


Figura 3.5: Esempio di NDRange bidimensionale in cui sono mostrati gli indici dei *work-group* e dei *work-item*.

Ogni *work-group* viene eseguito dai *processing element* di un unico *compute unit*, i singoli *work-item* sono eseguiti da uno o più *processing element* di uno *stream core* a seconda delle istruzioni. Ad ogni ciclo di *clock*, tutti gli *stream core* di un *compute unit* eseguono la stessa istruzione. Si chiama *wavefront* il blocco di *work-item* che vengono eseguiti contemporaneamente. Nella maggior parte delle GPU AMD un *wavefront* è composto da un massimo di 64 *work-item*, mentre i *work-group* possono contenere fino a 256 *work-item* e possono quindi essere suddivisi in 4 *wavefront*. A causa della dimensione delle *wavefront* si ottengono prestazioni migliori utilizzando *work-group* contenenti un numero di *work-item* multiplo di 64.

I *wavefront* sono utilizzati dalla GPU per nascondere le latenze di accesso alla memoria o dovute all'esecuzione di un operazione che necessita più cicli di *clock*: ogni volta che un *wavefront* risulta in attesa, invece di aspettare, ne viene eseguito un altro in modo che gli *stream core* della GPU non restino mai inattivi. Nell'esempio mostrato in figura 3.6 viene mostrata l'alternanza di quattro *work-item* appartenenti a diversi *wavefront* che vengono eseguiti dallo stesso *stream core*. Prima dell'inizio dell'esecuzione i quattro *work-item* (T0, T1, T2 e T3) sono tutti in attesa; al tempo 0 viene eseguito il primo *work-item* (T0) fino al ciclo di *clock* numero 20 in cui, a causa di un operazione o di un accesso di memoria, si verifica uno stallo. La GPU quindi scambia i *wavefront* ed inizia l'esecuzione di T1 che dopo 20 cicli va in stallo[4] e allo stesso modo vengono eseguiti T2 e T3. Durante l'esecuzione di T3, T0 smette di essere in stallo così che, quando T3 va in stallo, ricomincia la sua esecuzione.

[3] Il programmatore può scegliere se disporre gli indici, e quindi i *work-group*, su una, due o tre dimensioni a seconda dell'organizzazione del programma.

[4] Il numero di cicli dopo cui i *work-item* vanno in stallo è sempre lo stesso perché essi eseguono le stesse operazioni.

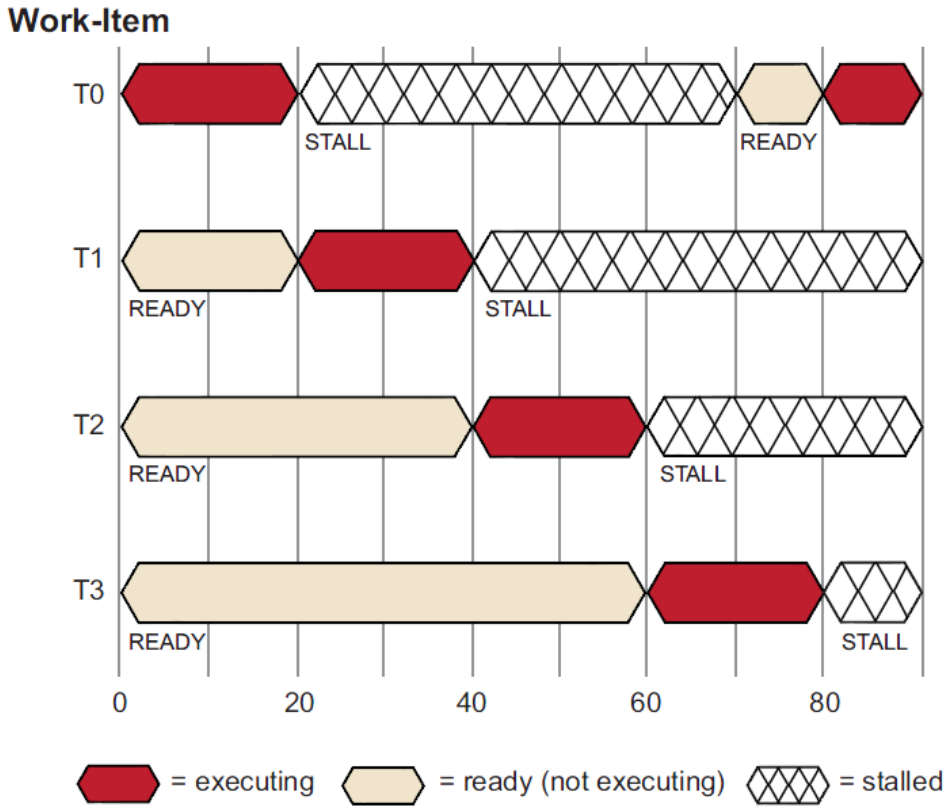


Figura 3.6: Esempio di un esecuzione di work-item su un stream core.

Nel caso in cui tutti i *work-item* siano in stallo allora la GPU resta in attesa finché non ce ne è uno pronto per l'esecuzione(fig. 3.7).

3.3 Controllo del Flusso

Istruzioni condizionali come *if and else* e cicli *for* comportano delle ramificazioni nelle esecuzione di un programma, in quanto possono essere seguiti diversi percorsi con istruzioni differenti da parte di diverse istanze del kernel. Ad esempio se si incontra un istruzione *if and else* possono essere eseguite le istruzioni all'interno del *if* o quelle all'interno dell'*else* portando ad una ramificazione in due percorsi differenti. Se la condizione Booleana è identica per tutti i *work-item* viene eseguito solo il percorso indicato dalla condizione, se invece c'è anche un solo *work-item* per cui la condizione Booleana è diversa allora devono essere eseguiti entrambi i percorsi. Il controllo del flusso, in questo caso, è fatto combinando i percorsi delle varie ramificazioni facendoli eseguire da un *wavefront*. Nel caso in cui un *work-item* contiene una diramazione con due percorsi, come un *if and else*, il *wavefront* esegue il primo percorso, quindi il contenuto della sezione legata alla clausola *if*, poi esegue il secondo, ovvero il contenuto della clausola *else*. Il tempo totale del'esecuzione della diramazione è quindi la somma dei tempi di esecuzione di tutti i percorsi, comportando un raddoppio del tempo di esecuzione.

La riduzione delle prestazioni può essere notevole se le diramazioni sono tante, come può accadere con i cicli *for*. Ad esempio se all'interno di un *wavefront* tutti i *work-item* eseguono una sola iterazione di un ciclo *for*, tranne uno che ne esegue 100, allora il tempo di esecuzione del *wavefront* è determinato dal *work-item* che esegue 100 iterazioni mentre gli

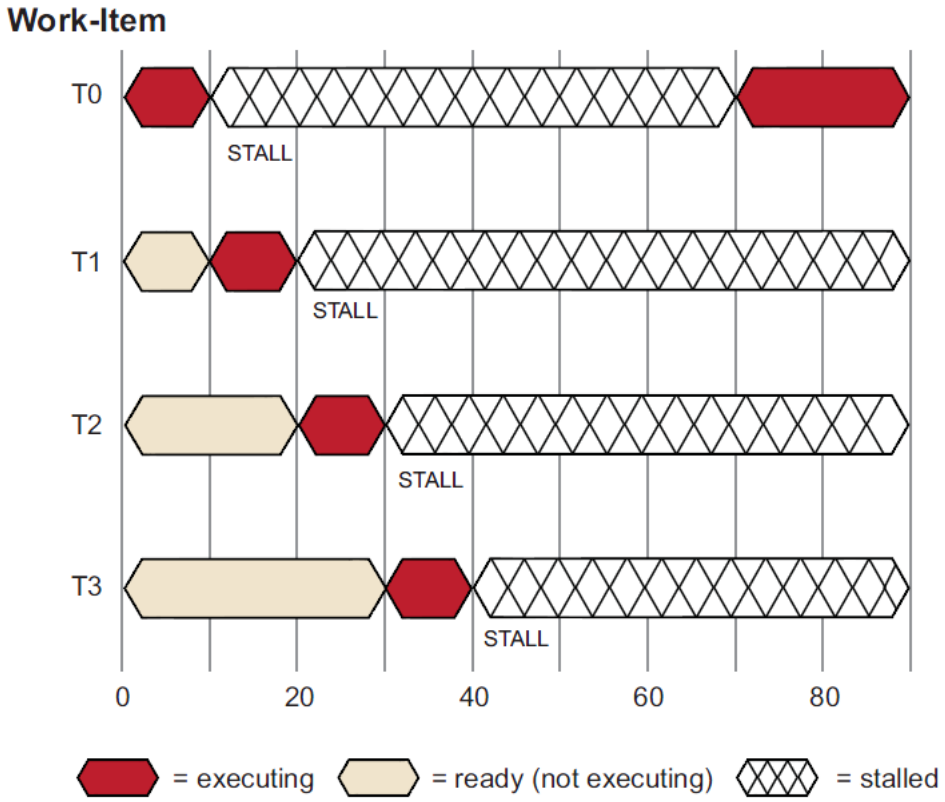


Figura 3.7: Esempio di un esecuzione di *work-item* su un *stream core*.

altri lo attendono. Questo accade perché il *wavefront* occupa il *compute unit* per tutto il tempo in cui c'è almeno un *work-item* in esecuzione, così che il tempo totale di esecuzione di un *wavefront* è determinato dal *work-item* con il tempo di esecuzione maggiore.

3.4 Architettura delle Memorie e Loro Accesso

In generale in una GPU i *work-item* hanno accesso a quattro distinte regioni di memoria (fig. 3.8):

- ***Global Memory***: è la memoria principale della scheda accessibile in lettura e in scrittura sia da tutti i *work-item* di tutti i *work-group* che dal *host*. É di grandi dimensioni ed è la memoria con cui il mondo esterno comunica con la GPU.
- ***Constant Memory***: è una regione della *Global Memory* che resta costante durante l'esecuzione del *kernel*. Solo l'*host* può allocarci e inizializzarci oggetti di memoria.
- ***Local Memory***: è una regione specifica per ogni *work-group* ed è accessibile solo dai *work-item* appartenenti a questo *work-group*. Se nel dispositivo non vi è una regione dedicata a questa memoria, può essere mappata in una sezione della *Global Memory*. É generalmente di piccole dimensioni ma ad accesso molto veloce.
- ***Private Memory***: è una regione dedicata a un singolo *work-item*. Le variabili definite nella *Private Memory* di un *work-item* non sono visibili dagli altri.

Nelle schede AMD l'accesso alla *global memory* avviene attraverso 8 canali distinti che sono connessi a più banchi di memoria, se due o più accessi di memoria devo passare dallo stesso

canale o devono accedere allo stesso banco di memoria l'hardware serializza gli accessi con un aumento del tempo necessario ad accedere alla memoria, si ha così un *channel conflict* o un *bank conflict*. Similmente la *local memory* è composta da 32 banchi di memoria che hanno ciascuno il proprio canale di accesso in modo da poter sopportare 32 accessi contemporanei alla memoria.

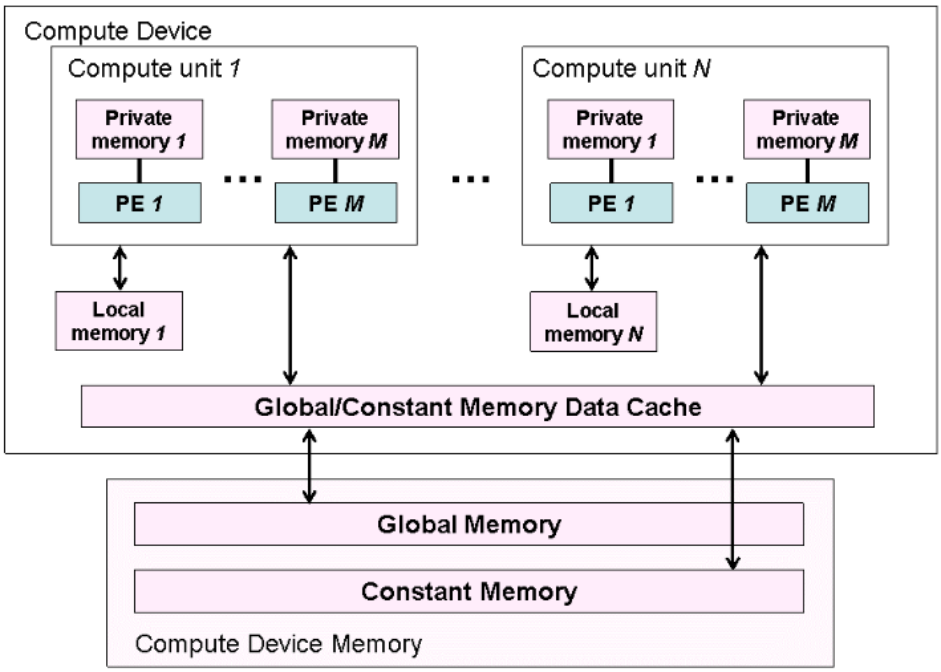


Figura 3.8: Schema dei collegamenti tra le memorie di una GPU.

Come mostrato in figura 3.8 queste memorie non sono realmente collegate ad oggetti creati dal software come i *work-item* o i *work-group*, in quanto questi non sono oggetti fisici. In realtà la *Private Memory* è collegata ai *processing element*, mentre le altre memorie sono collegate ai *Compute Unit*. Il collegamento delle memorie ai *work-item* e ai *work-group* è dovuto al fatto che ogni *work-item* viene processato da uno o più *processing element* e quindi può accedere solo alle memorie collegate ai *processing element* che lo eseguono; allo stesso modo ogni *work-group* ha accesso solo alle memorie collegate al *Compute Unit* che lo esegue.

Nelle GPU recenti le comunicazioni e i trasferimenti di dati tra l'*host* e la GPU avviene mediante bus PCIe (PCI express) comunemente presente negli attuali PC. Il bus PCIe è basato su un sistema di trasferimento seriale su una serie di canali che possono essere accorpati per aumentare la larghezza di banda disponibile. Le schede AMD utilizzano PCIe 2.0 x16 con 16 linee permettendo una velocità di accesso teorica di 8 GB/s; la velocità di accesso effettiva dipende dai chipset e dalla CPU del PC utilizzato.

Gli oggetti di memoria provenienti dall'*host* possono essere trasferiti solo sulla *Global Memory* o sulla *Constant Memory*; successivamente all'interno del kernel possono essere spostati all'interno della *Local Memory* o della *Private Memory* (fig. 3.9). Per tutte le variabili condivise da un *work-group* risulta conveniente utilizzare la *Local Memory* perché tipicamente di un ordine di grandezza più veloce rispetto all'accesso alla *Global Memory* che a sua volta è ~ 20 volte più veloce dell'accesso alla memoria dell'*host* mediante PCIe. Tuttavia gli *stream core* non possono accedere direttamente alla *Local Memory* ma il suo accesso deve avvenire attraverso unità *hardware* dedicate, quindi quando un *work-item* deve accedere alla *Local Memory* viene trasferito all'unità che può permettergli l'accesso, disattivandosi nell'attesa. Mentre un *work-item* è in attesa, come spiegato precedentemente,

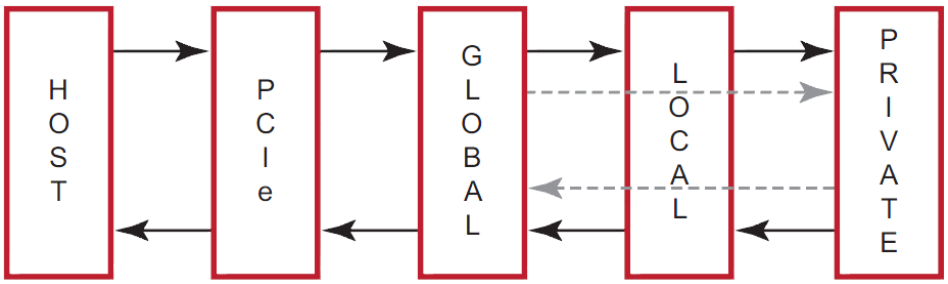


Figura 3.9: Schema del flusso di dati tra l'*host* e la GPU.

se possibile avviene uno scambio di *wavefront* in modo da permettere ad un altro *work-item* di essere eseguito al suo posto.

Il PC utilizzato per le misure e su cui è montata la GPU è un Supermicro SuperServer 7046GT-TRF con due processori Intel Xeon E5620 a 2.40GHz, 12 GB di memoria RAM e un chipset Intel composto da due chip 5520 (IOH-36D) e un South Bridge ICH10R. Nella figura 3.10 sono mostrate le caratteristiche delle memorie della GPU utilizzata in questa tesi, la ATI Radeon HD 5970.

OpenCL Memory Type	Hardware Resource	Size/CU	Size/GPU	Peak Read Bandwidth/ Stream Core
Private	GPRs	256k	5120k	48 bytes/cycle
Local	LDS	32k	640k	8 bytes/cycle
Constant	Direct-addressed constant		48k	16 bytes/cycle
	Same-indexed constant			4 bytes/cycle
	Varying-indexed constant			~0.6 bytes/cycle
Images	L1 Cache	8k	160k	4 bytes/cycle
	L2 Cache		512k	~1.6 bytes/cycle
Global	Global Memory		1G	~0.6 bytes/cycle

Figura 3.10: Schema con le dimensioni e il picco di banda delle memorie della AMD Radeon HD 5970. La memoria *images* è utilizzata per archiviare oggetti con una struttura a due o tre dimensioni.

Utilizzando alcuni strumenti software all'interno di AMD stream SDK abbiamo misurato la banda effettiva di trasferimento dati, che è risultata essere ∼3.06 GB/s nei trasferimenti tra la CPU e la GPU e ∼5.23 GB/s tra la GPU e la CPU, quindi significativamente inferiore alla velocità teorica massima.

3.5 Utilizzo della GPU come Trigger in Tempo Reale

Negli scorsi decenni i sistemi di *trigger* e di acquisizione dati degli esperimenti di fisica delle alte energie utilizzavano processori costruiti appositamente con la più recente tecnologia. Questa situazione è cambiata in seguito alla diffusione maggiore sul mercato di massa di dispositivi elettronici, che ha condotto allo sviluppo di dispositivi digitali di calcolo sempre più performanti e dai costi in costante riduzione. Queste motivazioni, insieme ai costi insostenibili per lo sviluppo di nuovi dispositivi *custom* con le ultime tecnologie, hanno portato la comunità scientifica verso un maggiore uso di soluzioni commerciali per la gran parte della catena di acquisizione dati.

Allo stesso modo la crescita esponenziale della potenza di calcolo delle CPU ha comportato negli anni un progressivo spostamento del loro utilizzo nei livelli di gestione dati. Fino

a dieci anni fa, infatti, i processori commerciale erano utilizzati solo nell'ultimo livelli di processamento *online* dei dati e per il loro immagazzinamento, mentre oggigiorno tutti i trigger di alto livello sono implementati su *farm* di computer, e solo i trigger di basso livello utilizzano elettronica appositamente costruita.

Queste scelte sono dovute ai costi minori, alla maggiore comodità riguardo all'istallazione e alla manutenzione di dispositivi elettronici appositamente realizzati per lo scopo e alla possibilità di aggiornarli all'uscita di un nuovo modello più potente.

Tuttora però non è ancora stato possibile implementare l'intero sistema di acquisizione dati e di *trigger*, per grandi esperimenti a *rate* elevato come quelli di fisica delle alte energie, su dispositivi e processori commerciali in quanto, a causa del grande numero di canali e dell'alto *rate* di eventi, le dimensioni richieste per le *farm* di computer sarebbero impraticabili.

Le GPU potrebbero essere utilizzate per colmare la mancanza di dispositivi commerciali su cui implementare trigger di primo livello e per ridurre le dimensione delle *farm* di computer utilizzate per i trigger di livello superiore.

Il problema principale di tale approccio è che le GPU non sono progettate per risposte a bassa latenza, visto che il loro scopo principale è di occuparsi di fotogrammi video a frequenze inferiori a 100 Hz; i sistemi di *trigger* di primo livello degli esperimenti di alta energia sono invece sottoposti a *rate* di eventi superiori ai MHz e con una latenza massima ben definita dalla dimensione delle memorie che immagazzinano i dati in attesa di una risposta dal *trigger*.

Tuttavia la potenza e la velocità delle GPU sono cresciute in modo tale che esse risultano approssimativamente compatibili con le richieste dei *trigger* di primo livello, rendendo possibile ipoteticamente l'utilizzo [23]. Inoltre essendo dispositivi senza sistema operativo il loro tempo di risposta è completamente predicibile dipendendo solo dall'algoritmo e dai dati, sebbene sia presente un elemento di incertezza relativo al controllo da parte della CPU e ai trasferimenti di codice e di dati dalla CPU alla GPU e viceversa. Per eliminare completamente questa imprevedibilità nella risposta temporale, dovuta al controllo da parte della CPU, si potrebbe utilizzare una scheda PCIe costruita appositamente per emulare i comandi della CPU, utilizzare una CPU priva di sistema operativo oppure utilizzare un sistema operativo *real-time*. Queste possibilità sono però di difficile implementazione in quanto non risultano facilmente disponibili informazioni sui controlli di basso livello delle GPU oltre al fatto che i *driver* di queste GPU sono compatibili solo con i sistemi operativi di largo consumo e che per la loro complessa architettura risulta difficile sfruttarle adeguatamente senza l'interfaccia proprietaria. In ogni caso queste modalità non standard di utilizzo della GPU sono contrarie all'idea di utilizzare una tecnologia economica già pronta all'uso quindi ci siamo concentrati sull'utilizzo di una GPU in un normale PC.

3.6 Automi cellulari

Parliamo ora brevemente di un algoritmo software che verrà utilizzato in seguito perché risulta adatto ad essere implementato su una GPU. Gli automi cellulari compaiono nell'ambiente scientifico alla fine degli anni '40 con lo scopo di studiare alcuni fenomeni biologici e da allora sono stati utilizzati per vari scopi scientifici, dalla teoria della computazione alla fisica. La loro popolarità è dovuta all'esempio conosciuto con il nome *Game of Life* proposto da Joho Horton Conway [26].

Un automa cellulare è costituito da una griglia bidimensionale di celle, di dimensione arbitraria. Ognuna di queste celle può assumere un numero finito di stati che possono essere colori, numeri, forme o semplicemente "vivo" o "morto". L'evoluzione di ogni cella è definita da una regola locale collegata al "vicinato" della cella: lo stato della cella in un determinato istante dipende dagli stati delle celle che sono definite vicine nell'istante immediatamente precedente. Quindi, a seconda di quali celle sono definite "vicine" a quella considerata, si ha una differente evoluzione dell'automa cellulare.

Consideriamo come esempio il *Game of Life*, che simula una popolazione di cellule viventi che si sviluppa nel tempo. Il tempo si evolve in passi discreti e lo stato di ogni cella dipende dallo stato delle celle del "vicinato" secondo le seguenti due regole:

1. una cella resta "viva" solo se ha 2 o 3 celle "vive" nel suo vicinato, se ne ha di meno "muore" per "solitudine" altrimenti "muore" di "fame" per "sovrappopolazione";

2. una cella morta torna in vita se ha esattamente 3 celle vive nel vicinato.

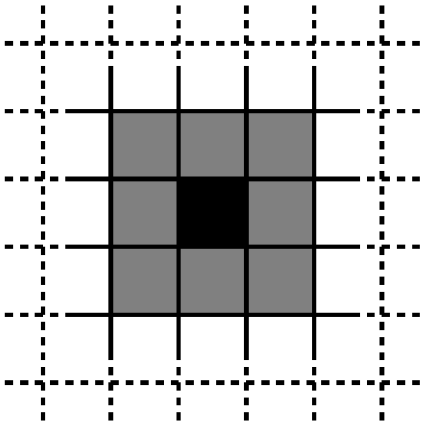


Figura 3.11: Vista della griglia di *Game of Life* in cui è mostrata una cella in nero insieme alle 8 celle del suo vicinato in grigio.

Nella figura 3.12 è mostrata l'evoluzione di cinque stati iniziali per tre passi temporali seguendo le regole appena elencate.

Gli automi cellulari sono stati utilizzati nel campo della fisica delle alte energie per la ricostruzione delle tracce negli spettrometri magnetici[28]. L'idea consiste nel considerare il segmento di traccia tra due *hit* (*tracklet*) come una cella di un automa cellulare. In questo modo la griglia di celle consiste nell'insieme di tutti i segmenti che uniscono due *hit* e il "vicinato" di una cella solitamente è formato dai segmenti di traccia il cui *hit* sinistro coincide con l'*hit* destro del segmento della cella presa in considerazione[5] [34]. Inoltre nelle condizioni sul vicinato è possibile inserire delle altre limitazioni come la pendenza dei segmenti, l'angolo formato tra un segmento e quelli "vicini", un calcolo della variazione del χ^2 che descrive la quantità statistica dell'appartenenza degli *hit* ad una traccia e altre condizioni scelte in base alla fisica degli eventi e alle caratteristiche del tracciatore. Le regole di evoluzione delle celle dipendono dal tracciatore ma in generale riguardano il numero dei vicini in modo che, in seguito ai passi temporali, sopravvivano solo le tracce che attraversano tutto il tracciatore e che non contengono biforcazioni. Ad esempio si può scegliere di assegnare un contatore numerico che parte da 0 ad ogni cella (fig. 3.13) e ad ogni passo temporale aumentare di uno il valore se è presente almeno un altra cella nel suo vicinato con un valore maggiore o uguale al suo. Alla fine dell'evoluzione il segmento con il valore maggiore è quello che da vita alla traccia più lunga (fig. 3.14).

[5] La scelta di un determinato *hit* del segmento serve a dare una direzione nella valutazione della traccia formata dalle celle alla fine dell'evoluzione.

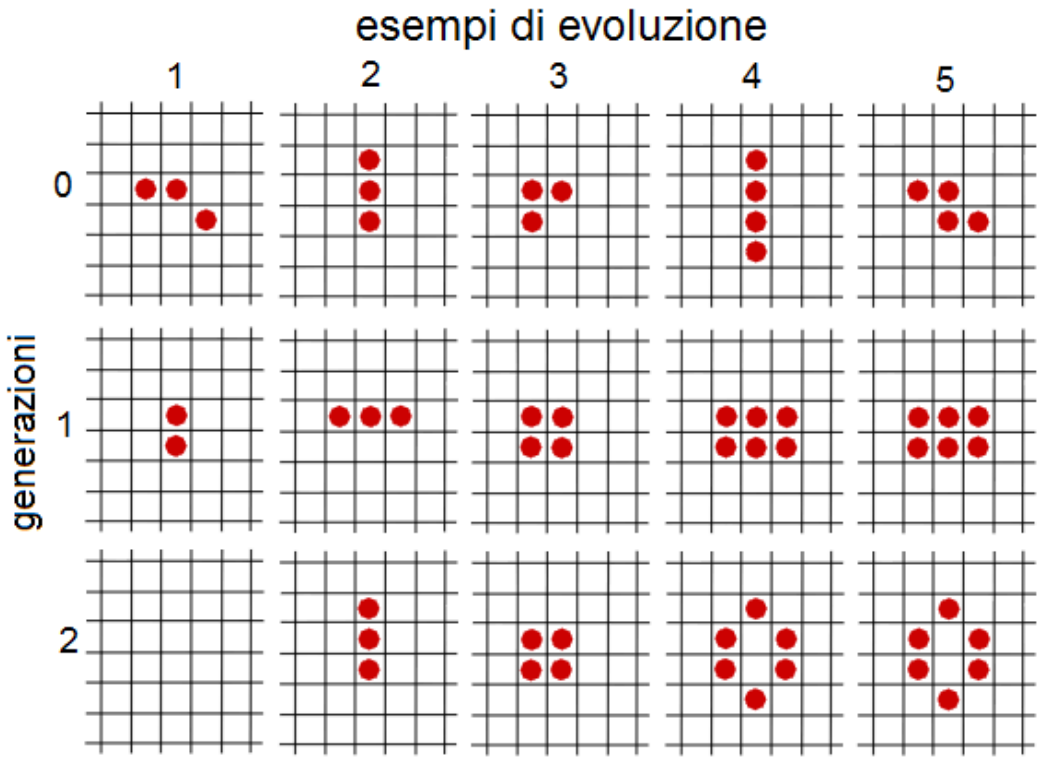


Figura 3.12: Esempi di evoluzione del *Game of Life*.

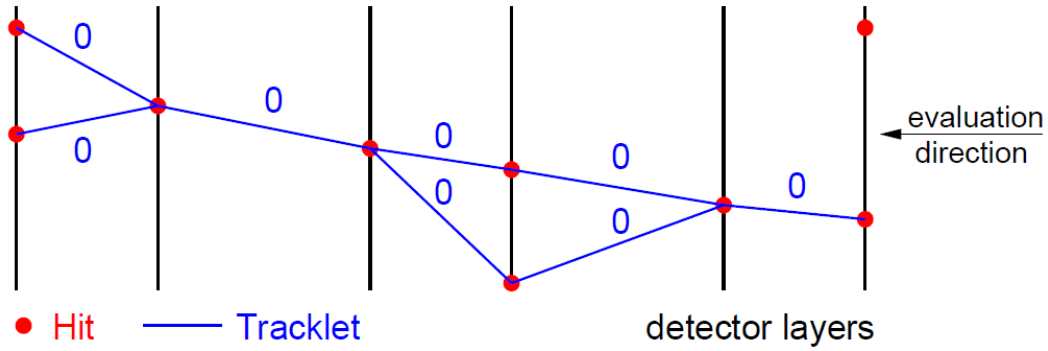


Figura 3.13: Esempio di stato iniziale di una griglia di celle formata da segmenti di traccia.

Il metodo degli automi cellulari per ricostruire le tracce risulta facilmente parallelizzabile in quanto è possibile assegnare ad ogni cella un *work-item* che si occupa della sua evoluzione ed è quindi adatto all'implementazione di un algoritmo di ricostruzione delle tracce su una GPU.

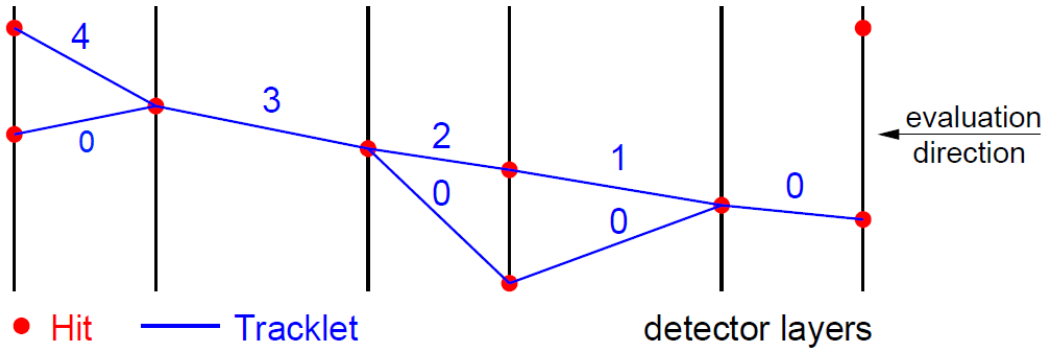


Figura 3.14: Momento finale dell'evoluzione della griglia di celle mostrata in figura 3.13 seguendo l'ordine crescente dei valori dei contatori si ottiene la ricostruzione della traccia.

CAPITOLO 4

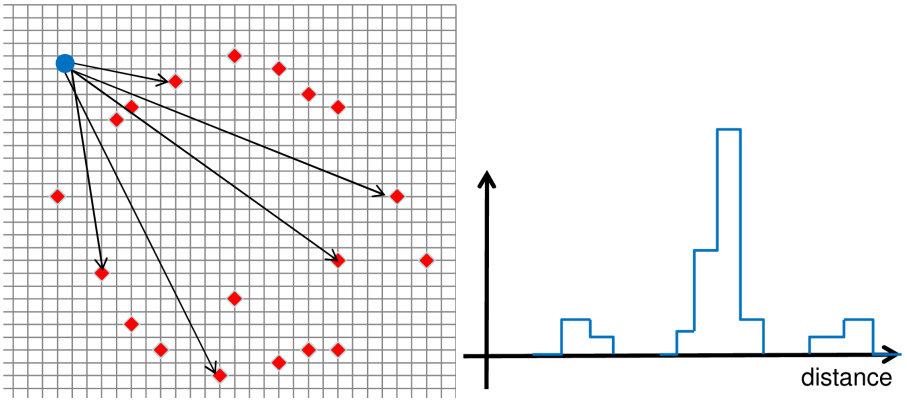
OTTIMIZZAZIONE DI UN ALGORITMO DI TRIGGER L0 PER IL RICH

Indice

Come prima applicazione, nell'esperimento NA62, si è studiato l'uso delle GPU nel trigger di primo livello (L0) per la ricerca ed individuazione di singoli cerchi Čerenkov nel RICH. Poiché al livello L0 la latenza massima del trigger è di 1 ms e ci aspettiamo un rate dell'ordine dei 10 MHz, la decisione del trigger deve essere pronta entro 1 ms ed ogni singolo evento deve essere elaborato mediamente in 100 ns.

Il rivelatore RICH comprende 2 regioni, ciascuna contenente circa 1000 fotomoltiplicatori di 16 mm di diametro disposti a nido d'ape. Ogni traccia genera un numero di hit che si avvicina a 20 in media per un π^+ con un impulso di 25 GeV/c. La conoscenza del centro e del raggio del cerchio, ricostruito mediante gli hit, può essere utile al fine di avere condizioni più selettive al livello L0. Sono stati ideati vari algoritmi per questo scopo, implementati su GPU NVIDIA utilizzando l'architettura CUDA.

Lo scopo di questa parte della tesi è stato di considerare uno di questi algoritmi ed ottimizzarlo per la GPU ATI RADEON 5970. L'algoritmo in questione si chiama DOMH (Device-Optimized Multi-Histograms), brevemente descritto nel seguito. viene considerata una griglia di possibili centri del cerchio (4.1a) e per ognuno si riempe un istogramma (4.1b) con le distanze tra il PMT e gli hit. L'algoritmo ricava da ogni istogramma la distanza che si ripete più frequentemente, quindi confronta le distanze scelte di ogni possibile centro in modo da selezionare come centro del cerchio il punto della griglia con la distanza con più ripetizioni. Ogni *work-group* processa un evento diverso (quindi più eventi sono elaborati



(a) *Distanze tra un PMT e gli hit.* (b) *Esempio di istogramma di un candidato centro.*

Figura 4.1: Algoritmo DOMH

contemporaneamente, uno per ogni *Compute Unit*[1]) così da utilizzare nel miglior modo la parallelizzazione offerta dalla GPU e ridurre il numero di conflitti nella local memory. Il massiccio utilizzo di memoria locale per la creazione degli istogrammi unito al gran numero di calcoli ed operazioni che il *work-group* deve compiere (ci sono circa 1000 PMT) sono le principali limitazioni alla velocità di questo algoritmo. Sono stati considerati algoritmi più veloci però, di questi, solo uno, oltre a DOMH, può supportare un'implementazione che permetta l'individuazione di più cerchi generati nel medesimo evento.

Il programma, contenente l'algoritmo DOMH ed eseguito dalla GPU, è chiamato *stream kernel* o semplicemente *kernel*. Il *kernel* viene compilato e lanciato all'interno di un programma host[2], eseguito dalla CPU, che ha il compito di trasmettere tutti i dati e di ricevere i risultati. I buffer trasferiti dalla memoria del PC alla *global memory* della GPU sono tre:

- un vettore bidimensionale contenente le coordinate degli *hit* di ogni evento in successione;
- un vettore contenente il numero di *hit* per ogni evento;
- un vettore bidimensionale contenente le coordinate dei possibili centri;

Alla fine dell'esecuzione del *kernel* viene trasferito un buffer dalla *global memory* contenente un vettore tridimensionale costituito dall'indice del punto della griglia che ha ottenuto più valori e quindi identificante il centro del cerchio, il raggio del cerchio e la frequenza del raggio nell'istogramma. Nel *kernel* ogni *work-group* copia nella *local memory* i dati di un singolo evento dai vettori presenti nella *global memory*. Successivamente i singoli *work-item* dei *work-group* eseguiti calcolano le distanze, riempono gli istogrammi e confrontano i candidati centri.

Tutte le volte che si eseguono operazioni di scrittura o di lettura sulla *local memory* è meglio utilizzare delle funzioni di sincronizzazione[3] per essere sicuri che quelle operazioni siano finite prima che qualche *work-item* ne esegua altre sugli stessi indirizzi di memoria. La barriera inoltre garantisce un corretto ordinamento delle operazioni inerenti la *local memory*, accodandole una dopo l'altra.

Nel *kernel* si adoperano *work-group* da 256 *work-item* nonostante se ne utilizzino solo 193 per i calcoli e i confronti, questo perché è conveniente al livello di prestazioni usare

[1] La GPU che utilizziamo ha 20 *Compute Unit*.
[2] In appendice si trovano le trascrizioni complete del programma host e del *kernel*.
[3] barrier(CLK_LOCAL_MEM_FENCE).

work-group contenenti un numero di *work-item* multiplo del *wavefront*[4]. I 193 *work-item* sono suddivisi in gruppi, tutti separati da una barriera, in modo da distribuire i vari compiti su *work-item* diversi:

- 32 *work-item* copiano gli *hit* di un evento dalla *global memory* alla *local memory* ed inizializzano il vettore "histos" che contiene le distanze migliori per ogni istogramma;
- 128 *work-item* calcolano le distanze tra i candidati centri del cerchio e i centri dei PMT colpiti, riempono gli istogrammi e trovano la distanza con più ripetizioni di ogni punto della griglia;
- i restanti 33 *work-item* eseguono i confronti tra tutti i punti della griglia per trovare il centro del cerchio.

4.1 analisi delle prestazioni dell'algoritmo non ottimizzato

Sono partito da una traduzione dell'algoritmo DOMH, originariamente scritto in CUDA [33] per schede NVIDIA, in OpenCL [36]. Per eseguire l'analisi delle prestazioni dell'algoritmo sulla scheda bisogna misurare i tempi di esecuzione del *kernel*, per questo scopo bisogna abilitare il profiling di openCL inserendo l'opzione CL_QUEUE_PROFILING_ENABLE nella funzione *clCreateCommandQueue* nel programma host. Con l'abilitazione del profiling le librerie runtime di openCL registrano automaticamente le informazioni di *timestamp* di ogni operazione di memoria od esecuzione del *kernel*. Il tempo di esecuzione del *kernel* comprende anche il suo tempo di lancio, cioè la somma del tempo speso dal *programma utente*[5] a quello che occorre alle librerie per avviare il *kernel*. Il tempo di lancio del *kernel* per le GPU è notevolmente più lungo rispetto alle CPU, infatti per le GPU si attesta intorno ai 200 μs[6] mentre per la CPU risulta intorno ai 25 μs [4]. Inoltre l'abilitazione del profiling aggiunge approssimativamente da 10 μs a 40 μs al tempo di lancio del *kernel* e quindi anche alla misura del tempo di esecuzione.

Utilizzando una simulazione di Monte Carlo sono stati prodotti due tipi di dati per l'analisi temporale e di risoluzione dell'algoritmo:

- per l'analisi temporale sono stati usati eventi con singoli cerchi, tutti centrati nello stesso punto (0,0), con raggio variabile, ma numero di hit costante[7]; per i confronti tra gli algoritmi ottimizzati sono stati usati eventi con 20 hit per cerchio, in quanto più vicino al numero medio di *hit* che ci si aspetta;
- per l'analisi della risoluzione[8] dell'algoritmo sono stati usati invece eventi con singoli cerchi con posizione del centro, raggio e numero di hit variabile, in modo da studiare la dipendenza della risoluzione dai vari parametri su tutto lo *spot*;

Per verificare le prestazioni dell'algoritmo in presenza di rumore sono stati aggiunti *hit* spuri nei dati.

Nell'analisi delle prestazioni di un algoritmo è importante considerare l'andamento del tempo di esecuzione del *kernel*, per singolo evento, al variare del numero di eventi che vengono fatti processare dalla GPU. Come compromesso tra velocità di esecuzione e risoluzione dell'algoritmo sono stati usati istogrammi da 16 *bin*, da riempire con le distanze tra i centri e gli *hit*. Di seguito è mostrata la tabella con i dati (4.1), il grafico (4.2) ed un suo zoom

[4] In questa scheda una *wavefront* è formata da 64 *work-item*.
[5] Il periodo che trascorre tra l'incodamento dei comandi ed il loro invio alla GPU.
[6] Questo tempo dipende dalle dimensioni del *kernel* e da quante operazioni deve compiere.
[7] Sono stati generati dati con questo numero di *hit* 7,10,12,15,17,20,23,25.
[8] Nelle misure di risoluzione sono stati fatti processare contemporaneamente 10000 eventi per avere una statistica significativa.

(4.3), per ogni punto è stata fatta la media di 30 misure e per calcolare gli errori sui tempi di esecuzione, alcune volte invisibili nei grafici su questa scala, abbiamo usato lo stimatore $s = \sqrt{\frac{\sum_i^n (x_i - \overline{x})^2}{n-1}}$ con $n = 30$.

numero eventi	tempo esecuzione(ns)
20	7042 ± 20
40	5242 ± 167
100	5192 ± 38
200	4878 ± 5
300	7687 ± 139
400	6890 ± 115
500	6535 ± 85
600	6200 ± 70
700	6038 ± 69
1000	5626 ± 47
3000	5075 ± 18
5000	4964 ± 10
7000	4912 ± 7
10000	4879 ± 6
15000	4830 ± 92

Tabella 4.1: Tempi di esecuzione del kernel al variare del numero di eventi processati

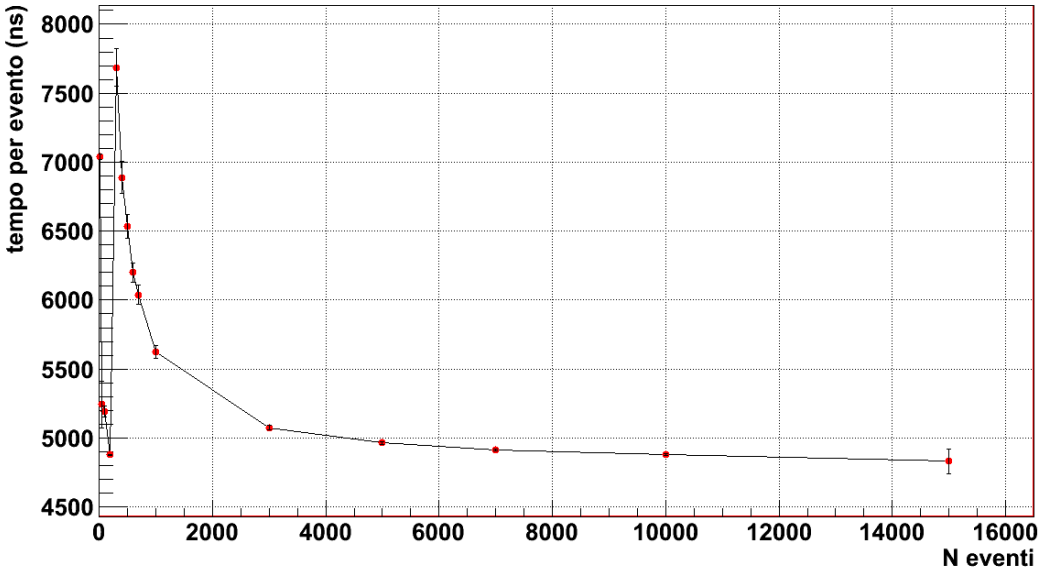


Figura 4.2: Grafico del tempo di esecuzione del *kernel*, per singolo evento, al variare del numero di eventi processati

Come si può notare dai grafici (4.2 e 4.3) l'andamento non è monotono come ci si potrebbe aspettare, ma abbiamo un primo minimo a 200 seguito da un picco e da un andamento esponenziale. Questo andamento, in particolare la zona precedente al picco, è dovuto probabilmente all'utilizzo di una *cache* interna utilizzata nella lettura di dati contenuti nella *local memory*. Quando i dati, che si leggono dalla *local memory*, superano la capienza della *cache*, tale memoria deve essere azzerata ed il *kernel* perde molto tempo [23]. Questo si vede nel grafico come uno scalino seguito da un picco ed da un'ulteriore curva esponenziale.

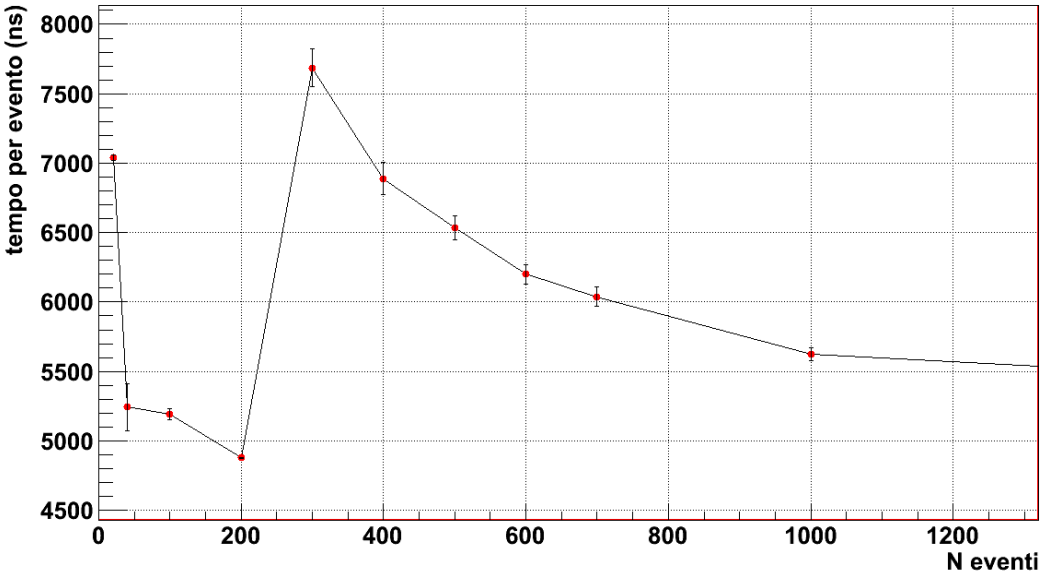


Figura 4.3: Zoom del grafico del tempo di esecuzione del *kernel*, per singolo evento, al variare del numero di eventi processati

Nella regione dello scalino, tra il picco e il massimo, il tempo di esecuzione del *kernel* soffre di un grande errore perché oscilla tra il valore del minimo e quello del picco. L'errore è dovuto quindi alla presenza di due popolazioni che possiedono un errore dello stesso ordine di grandezza delle altre misure (figura 4.4). Visto che i dati processati sono sempre gli stessi ad ogni iterazione, il non attestarsi intorno ad un unico tempo di esecuzione medio potrebbe essere dovuto a processi concorrenziali al *kernel*, come il *refresh* dello schermo, che, se capitano durante l'esecuzione, fanno passare lo scalino ai punti vicino alla transizione. Quindi queste fluttuazioni potrebbero essere eliminate con la disabilitazione dell'*output* video della GPU, ma fino ad ora non è stato possibile in quanto non consentito dal software della ATI.

Le ottimizzazioni dell'algoritmo verranno confrontate in seguito utilizzando principalmente il tempo di esecuzione per 1000 eventi.

4.2 Prime Ottimizzazioni

All'inizio ho eseguito solo delle ottimizzazioni sulle operazioni matematiche e di memoria che esegue il *kernel*. Alcune ottimizzazioni non hanno comportato miglioramenti sostanziali come la sostituzione di moltiplicazioni o divisioni per potenze di due con operatori di shift.

Il ridotto utilizzo della *global memory* da parte del *kernel*[9] non permette grandi margini di miglioramento nei tempi di esecuzione dall'ottimizzazione dell'accesso alla *global memory* per operazioni di scrittura o lettura. L'implementazione di un accesso sequenziale agli indirizzi di memoria da parte dei *work-item* in un *work-group* ha comportato un miglioramento solo di circa 25 ns per evento. Una lettura (o scrittura) sequenziale nella *global memory* comporta che l'intero *wavefront* acceda ad un solo canale (sezione 3.4), questo modo di accedere alla memoria potrebbe sembrare sconveniente se non si considera che tutti i *work-group* della GPU accedono alla *global memory* contemporaneamente, utilizzando diversi canali [4].

Neanche l'implementazione di un accesso sequenziale alla *local memory* ha comportato un miglioramento delle prestazioni del *kernel* nonostante la rimozione dei conflitti di accesso tra

[9] I dati occupano circa 17 Mb su 1 Gb della *global memory*.

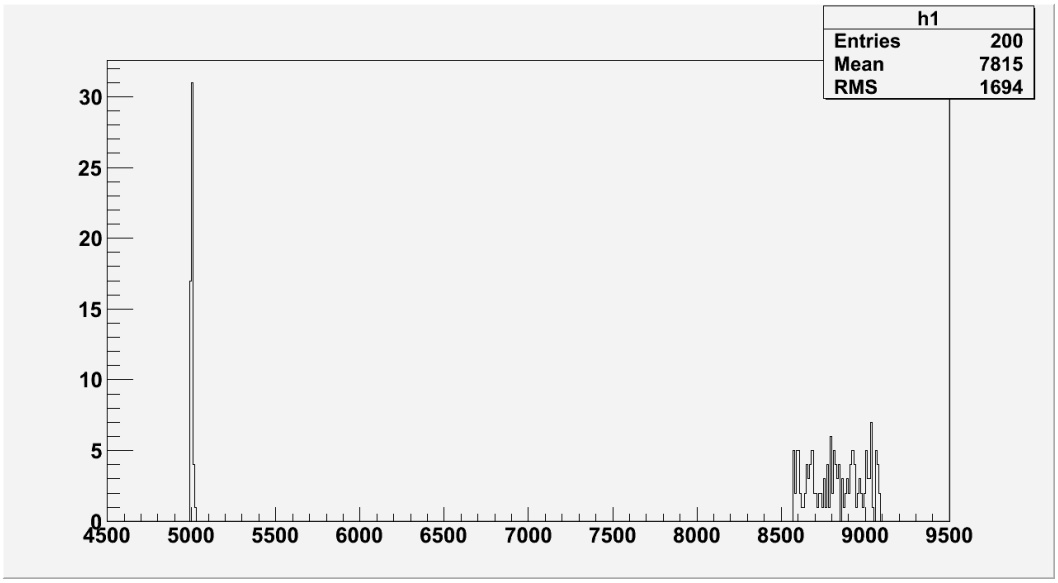


Figura 4.4: Istogramma dei tempi di esecuzione se si processano 220 eventi, nella regione prossima alla discontinuità nel grafico 4.3

i banchi della local memory sia molto importante ai fini prestazionali, in quanto teoricamente può portare ad un miglioramento del tempo di accesso alla memoria fino ad un fattore 32. Gli accessi allo stesso banco di memoria vengono serializzati, con la conseguente perdita del parallelismo tra i *work-item*: questo porta ad uno stallo della *compute unit* fintanto che tutti i conflitti non siano risolti [4]. Una successiva verifica con l'ATI Stream Profiler[10] ha mostrato che in entrambi i casi i conflitti nella *local memory* sono inferiori all'1%, giustificando così il mancato miglioramento delle prestazioni.

Importante miglioramento nelle prestazioni, circa 1.4 μs per evento, si è raggiunto con l'utilizzo della "funzioni native" di openCL in particolare native_divide() e native_sqrt(). Queste funzioni sono più veloci rispetto alle corrispondenti funzioni *classiche*[11], nonostante siano meno accurate. La minore accuratezza, comunque, non ha comportato cambiamenti nella risoluzione dell'algoritmo.

Dopo queste modifiche il tempo di esecuzione per 1000 eventi analizzati dal *kernel* è 4035 ± 33 ns per evento.

4.3 Ottimizzazione Del numero di *work-item*

Come già accennato, le barriere sono funzioni di sincronizzazione, e tutti i *work-item* devono aver raggiunto la barriera prima che qualsiasi di essi possa proseguire oltre. Ciò comporta, con la suddivisione del lavoro dell'algoritmo in gruppi composti da *work-item* diversi, un ridotto utilizzo della parallelizzazione e delle risorse di esecuzione perché tutti i *work-item* restano fermi per la maggior parte del tempo (gli ultimi 63 non vengono mai utilizzati). Per esempio, mentre i primi 32 *work-item* copiano gli hit ed inizializzano il vettore "histos", gli altri *work-item* non lavorano; allo stesso modo dopo la prima barriera i successivi 128 eseguono delle operazioni ed i primi 32 insieme ai rimanenti restano inutilizzati. Per

[10] L'ATI Stream Profiler è un tool che permette di evidenziare i conflitti nelle memorie ed esaminare le prestazioni del *kernel*.

[11] native_sqrt() è più veloce di sqrt() di un fattore 1.8, mentre native_divide() è più veloce di "/" di un fattore 4.4.

migliorare la performance del *kernel* dovrebbero essere sempre in esecuzione un numero di *work-item* multiplo del *wavefront*.

Su questa base ho esteso, dove possibile[12] i *work-item* che eseguono le operazioni contemporaneamente. In particolare portando a 256 *work-item* (da 32) l'inizializzazione del vettore "histos" si riduce di circa 100 ns il tempo di esecuzione del *kernel* per evento; allo stesso modo estendendo a 256 (da 128) il numero di *work-item* che calcolano le distanze tra i centri e gli *hit* si ha un ulteriore miglioramento di circa 600 ns per evento.

Inoltre ho alterato la parte del *kernel* che eseguiva i confronti tra gli istogrammi aggiungendo uno step, in cui 256 *work-item* ne confrontano 4 PMT ciascuno, prima di quello già esistente, esteso da 32 a 64 *work-item*, ed uno, da 8 *work-item*, dopo. L'ultimo confronto, in cui un *work-item* confronta i rimanenti 8 per trovare il migliore centro del cerchio, non può essere modificato.

Dopo queste modifiche, il tempo di esecuzione per 1000 eventi analizzati dal *kernel* è 3389 ± 10 ns per evento.

4.4 Ottimizzazione Delle istruzioni

Una verifica con l'ATI Stream Profiler, sotto la voce "ALUPacking", ha mostrato una scarsa efficienza del compilatore nell'impacchettare negli ALU[13] scalari o vettoriali fino a 5 istruzioni VLIW[14]. Una valore basso di questo parametro indica l'impossibilità di un utilizzo pieno del processore.

Il primo approccio al problema, che non ha portato risultati, si è basato sull'aggiunta di un ciclo *for* intorno al corpo del *kernel*, in modo che venisse eseguito più volte comportando il processamento di più eventi per ogni *work-group*.

La seconda possibilità è consistita nel verificare che il *kernel* contenesse abbastanza parallelismo per riempire tutti i *Processing Element*. Per esporre più parallelismo al compilatore ho "srotolato"[15] tutti i loop di un fattore 4 ed ho raggruppato, dove possibile, le operazioni di lettura e scrittura in moto che potessero essere eseguite in parallelo. Questa operazione ha comportato un miglioramento di circa 90 ns per evento, arrivando ad un tempo di esecuzione per evento di 3320 ± 10 ns.

4.5 Ottimizzazione Dei Wavefront

L'ATI Stream Profiler ha mostrato anche un basso valore alla voce "ALUBusy", che indica la frazione percentuale, sul tempo di esecuzione del *kernel*, in cui la GPU processa istruzioni ALU. Un basso valore di questo parametro suggerisce che la *Compute Unit* non ha a disposizione abbastanza *wavefront* per mantenere sempre in uso gli *Stream Core*. La possibilità di avere più *wavefront* attive contemporaneamente permette di "nascondere" sia la latenza delle operazioni aritmetiche[16] sia la latenza di accesso alla *global memory*[17], perché, mentre un *wavefront* aspetta di accedere alla memoria, la *Compute Unit* può processare un'altro *wavefront* finché anche questa non deve accedere alla memoria. Ogni *Compute Unit* esegue un quarto di *wavefront* ogni ciclo, così in 4 cicli è processata l'intera *wavefront*. Quindi, per esempio, bastano 2 *wavefront* per nascondere la latenza dovuta alle operazioni

[12] Ogni evento presenta al massimo 32 hit quindi è impossibile usare più di 32 *work-item* per la copiatura degli hit dalla global alla local memory.
[13] Arithmetic Logic Unit, è l'unità responsabile delle operazioni logiche o aritmetiche.
[14] Very Long Instruction Word, il compilatore assembla le istruzioni in blocchi che contengono fino a 5 operazioni, una per ognuno dei 5 *Processing Element* che compongono uno *Stream Core*.
[15] Ho dovuto usare un fattore 4 invece di 5 per mantenere il fattore in potenze di due.
[16] La maggior parte delle operazioni aritmetiche hanno una latenza di 8 cicli, quindi il *workitem* deve aspettare 8 cicli prima di poter utilizzare la variabile appena calcolata.
[17] La latenza della global memory varia da 300 a 600 cicli.

aritmetiche. Una volta raggiunto il numero di *wavefront* necessario a nascondere la latenza, l'aggiunta di ulteriori *wavefront* non offre ulteriori benefici.

Ci sono due principali motivi per i quali le *Compute Unit* possono non avere a disposizione un numero sufficiente di *wavefront* per nascondere le latenze: un eccessivo uso dei registri o della *local memory* da parte del *kernel*.

Ogni *Compute Unit* ha a disposizione 16384 registri condivisi tra tutti i *wavefront* attivi nella *Compute Unit*. Un *kernel* che utilizza una grande quantità di registri può provocare una riduzione del massimo numero di *wavefront* attivi, in quanto la *Compute Unit* non dispone del necessario numero di registri per tenere attivi tutti gli *wavefront*. Utilizzando sempre l'ATI Stream Profiler abbiamo trovato che il *kernel* supporta fino a 11 *wavefront* attivi, che sono sufficienti a nascondere le latenze.

Come già accennato, oltre ai registri, anche l'utilizzo della *local memory* può ridurre il numero di *wavefront* attivi in quanto la *local memory* è condivisa tra tutti i *work-group* attivi che possono contenere fino ad un massimo di 4 *wavefront*[18]. Il *kernel* utilizzava per l'algoritmo circa 22 KB sui 32 KB posseduti dalla *local memory*, comportando la possibilità di tenere attivo un solo *work-group* e quindi solo 4 *wavefront*, insufficienti a mantenere sempre in uso le *Compute Unit* durante le latenze dovute agli accessi di memoria. Quindi per aumentare il numero di *wavefront* attivi era necessario ridurre l'utilizzo della *local memory*: per fare ciò ho ridotto alla metà la dimensione del vettore "histos" inserendo le distanze più frequenti di due centri nella stesso *bin* del vettore. In questo modo ho ridotto a 16 bit (invece di 32 bit) lo spazio utilizzato per ogni distanza. Allo stesso modo ho ridotto ad un quarto la dimensione degli istogrammi di ogni centro riducendo a 8 bit lo spazio destinato alla frequenza di ogni distanza. La riduzione a 16 e 8 bit è possibile grazie al fatto che i valori che riempiranno entrambi i vettori non arrivano comunque ad 8 bit[19]. Con questo intervento si è ridotto l'occupazione della *local memory* a 7.7 KB permettendo di mantenere attivi fino a 4 *work-group* e quindi 16 *wavefront*. Il numero dei *wavefront* attivi, quindi, resta limitato ad 11 dal numero di registri utilizzati dal *kernel*; l'utilizzo dei registri, però, risulta difficile da ridurre senza modificare strutturalmente l'algoritmo.

In seguito a questa ottimizzazione il tempo di esecuzione per 1000 eventi analizzati dal *kernel* è 2836 ± 10 ns per evento: complessivamente si è raggiunto quindi un miglioramento di 500 ns rispetto all'algoritmo come era stato scritto inizialmente.

[18] La dimensione del *work-group* è limitata ad un massimo di 256 *work-item* mentre il *wavefront* è sempre composto da 64 *work-item*.

[19] Il numero massimo di hit è 32, da 0 a 31 (6 bit), e il raggio massimo è inferiore a 64 cm (7 bit).

4.6 Uso del Quadrato delle Distanze

Come ultima ottimizzazione ho tolto dall'algoritmo la radice quadrata nel calcolo delle distanze per poi inserirla, solo per il risultato finale, nel programma *host*. La rimozione della radice quadrata non solo ha permesso un miglioramento del tempo di esecuzione di circa 70 ns per evento ma ha anche comportato un cambiamento della risoluzione dell'algoritmo in quanto riduce la risoluzione sul raggio, e sulla posizione del centro, per raggi piccoli[20] mentre migliora per raggi grandi. Visto che non ci aspettiamo, o comunque non sono interessanti, eventi con raggi piccoli, l'utilizzo della distanza al quadrato comporta un miglioramento generale della risoluzione dell'algoritmo.

In tabella 4.2 e in figura 4.5 è riportato il confronto tra la risoluzione, in cm, dell'algoritmo con la distanza e con il quadrato della distanza al variare del numero di bin degli istogrammi. Per il confronto sono stati usati i dati con le posizioni dei centri variabili come indicato nella sezione 4.1.

numero bin	risoluzione con distanza (cm)	risoluzione con quadrato distanza (cm)
4	8.75	6.43
8	7.93	5.12
12	5.60	4.45
16	7.48	3.96
20	3.78	3.71
24	4.24	3.51
28	4.89	3.46
32	5.45	3.42

Tabella 4.2: confronto tra risoluzione con la radice quadrata o senza al variare del numero di *bin* negli istogrammi.

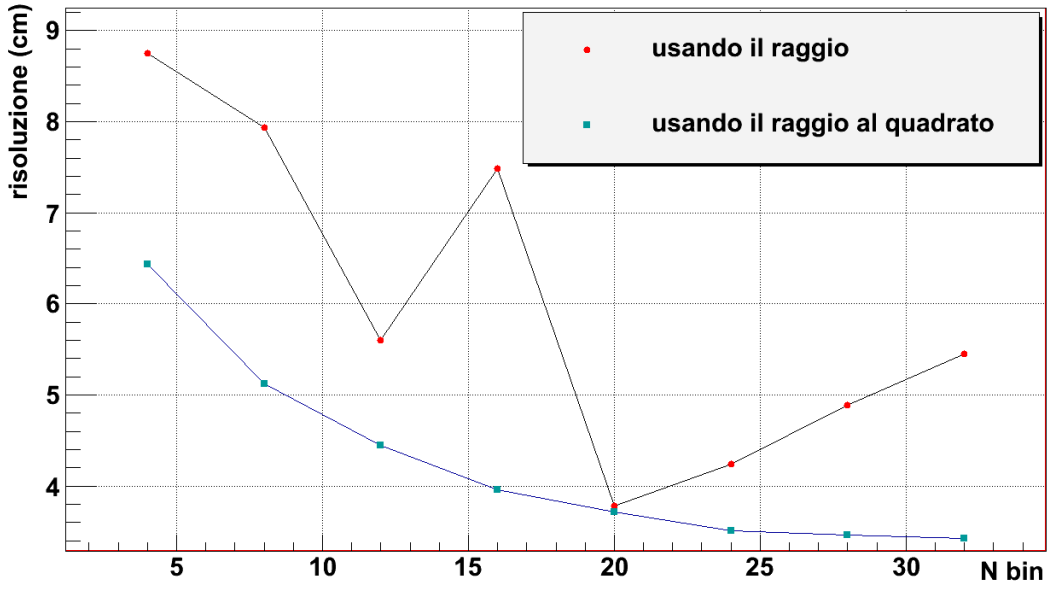


Figura 4.5: confronto tra risoluzione con la radice quadrata o senza al variare del numero di *bin* negli istogrammi.

[20] All'aumentare della distanza i bin risultano più densi e meno distanziati l'uno dall'altro in quanto sono equidistanti se si considera il quadrato del raggio.

4.7 analisi delle prestazioni dell'algoritmo ottimizzato

In questa sezione analizzeremo le prestazioni del *kernel* al variare di alcuni parametri relativi ai dati e all'algoritmo. Come prima cosa analizzeremo la dipendenza del tempo di esecuzione dell'algoritmo dal numero di *bin* degli istogrammi che vengono riempiti con le distanze tra i centri e gli *hit*. Di seguito si trovano la tabella 4.3 ed il grafico 4.6 delle misure.

numero bin	risoluzione (cm)	tempo di esecuzione (ns)
4	6.43	2694 ± 10
8	5.12	2725 ± 10
12	4.45	2751 ± 9
16	3.96	2773 ± 9
20	3.71	2804 ± 10
24	3.51	2833 ± 12
28	3.46	2868 ± 10
32	3.42	3103 ± 23

Tabella 4.3: Tempo di esecuzione del *kernel* al variare del numero di *bin* negli istogrammi.

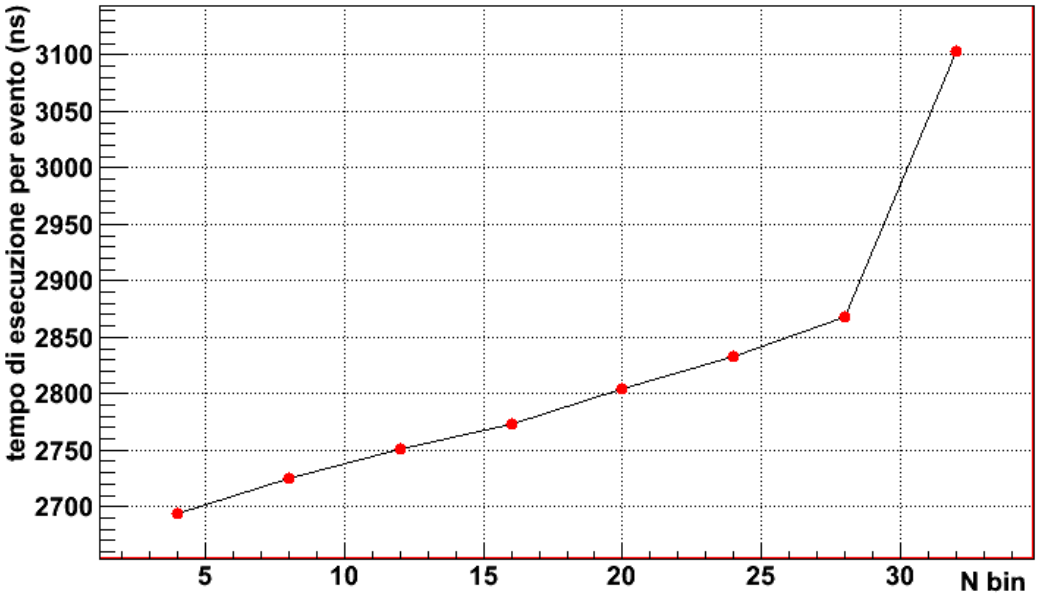


Figura 4.6: Tempo di esecuzione del *kernel* al variare del numero di *bin* negli istogrammi.

Il tempo di esecuzione del *kernel* con 32 *bin* non segue l'andamento dei casi precedenti perché gli istogrammi occupano un quantitativo di *local memory* tale da permettere l'esecuzione contemporanea solamente di 2 *work-group* invece di 4 in una *Compute Unit*, come spiegato nella sezione 4.5.

Per scegliere quanti *bin* sia conveniente usare risulta utile il grafico (4.7) che mette in relazione il tempo di esecuzione del *kernel* e la risoluzione che si ottiene con quel numero di *bin*; il numero di *bin* nel grafico è riportato accanto ad ogni punto. Dal grafico si nota che scegliere di utilizzare istogrammi da 16 *bin* sembra un buon compromesso tra velocità e risoluzione del *kernel*.

Dato che il numero di *hit* che compongono i cerchi varia a seconda del tipo di particella che attraversa il RICH e del suo impulso (fig. 2.15), è interessante vedere la dipendenza del tempo di esecuzione del *kernel* dal numero di *hit* (in tabella 4.4 e nel grafico 4.8). Il

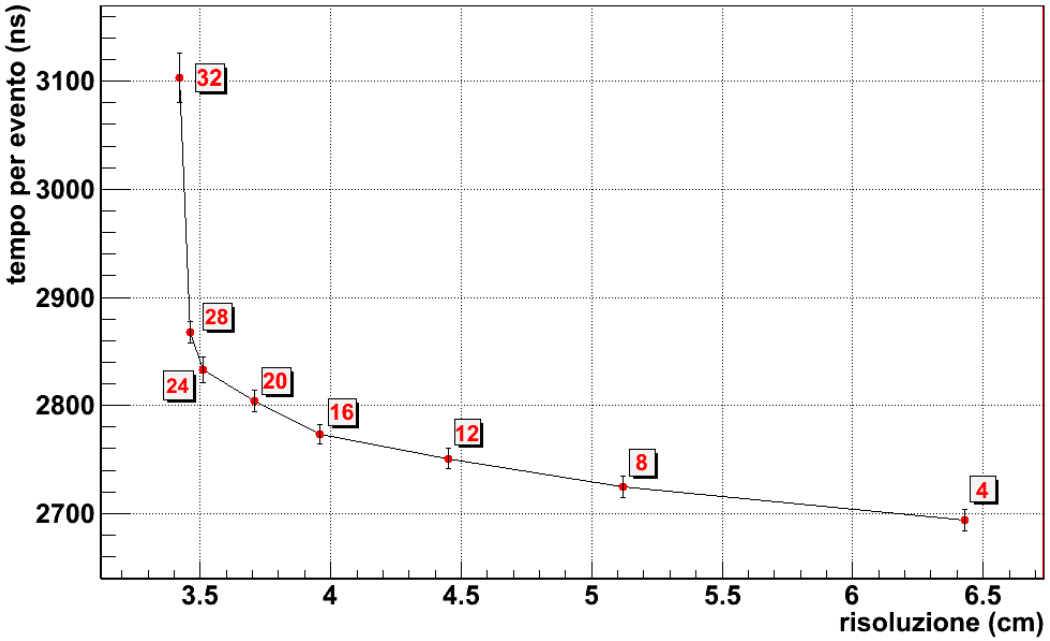


Figura 4.7: Tempo di esecuzione del *kernel* in funzione della risoluzione spaziale. I numeri nelle caselle indicano il numero di *bin* usati negli istogrammi delle distanze.

numero hit	tempo di esecuzione (ns)
7	817 ± 1
10	1627 ± 434
12	2116 ± 7
15	2366 ± 7
17	2529 ± 9
20	2773 ± 9
23	3023 ± 12
25	3190 ± 11

Tabella 4.4: Tempo di esecuzione del *kernel* al variare del numero di *hit*.

cambiamento di pendenza che si nota nel grafico (4.8) è dovuto allo spostamento della discontinuità, tra il minimo e il picco (vedi sezione 4.1), oltre i 1000 eventi, permettendo di avere tempi di esecuzione inferiori rispetto a quello che ci potremo aspettare solamente dalla riduzione del numero di calcoli dovuta al minor numero di *hit*. Si ha uno spostamento della discontinuità perché con meno *hit* ci sono meno dati da leggere dalla *local memory* quando si eseguono i calcoli delle distanze.

Vediamo ora l'andamento temporale del *kernel* variando il numero di eventi processati. Confrontando la tabella 4.5 e i grafici 4.9 e 4.10 con i rispettivi per l'algoritmo non ottimizzato (vedi 4.1 e 4.2) si nota che, nonostante il grande miglioramento nel tempo di esecuzione (compreso tra 2.2 μs e 4.4 μs a seconda del numero di eventi) hanno lo stesso andamento temporale in funzione del numero di eventi. Dal grafico si nota che il punto di lavoro migliore sarebbe intorno a 500 eventi, dove si ha il minimo per il tempo di esecuzione.

Infine è importante eseguire una verifica della stabilità temporale della GPU su periodi lunghi di esecuzione in quanto se l'algoritmo non fosse stabile durante un *run* il processamento di alcuni eventi potrebbe durare più della latenza e l'evento andrebbe perso. Per questo motivo è stato fatto eseguire alla GPU un ciclo di 5000 esecuzioni del *kernel*, per una durata

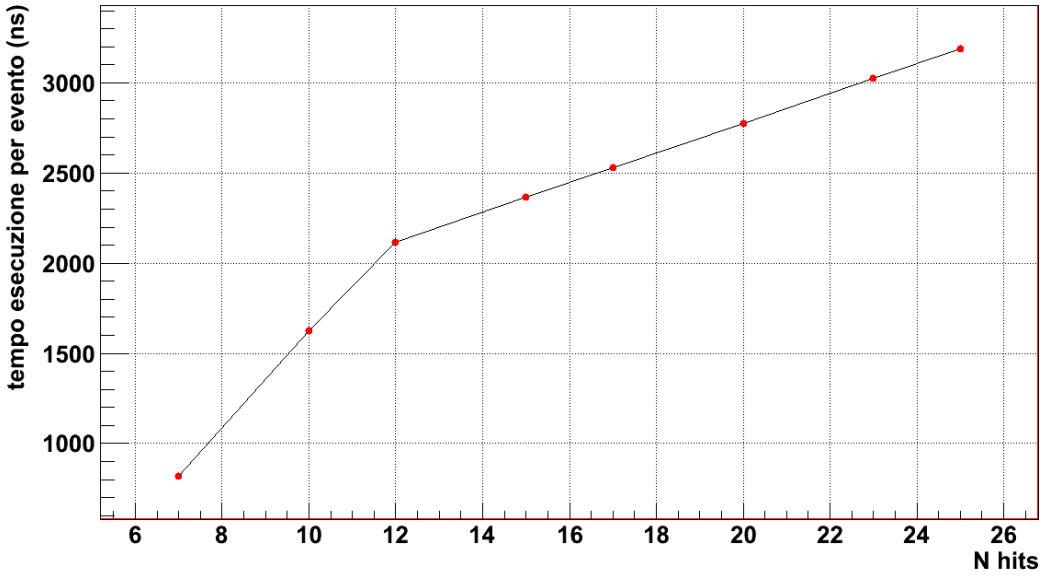


Figura 4.8: Tempo di esecuzione del *kernel* al variare del numero di *hit*.

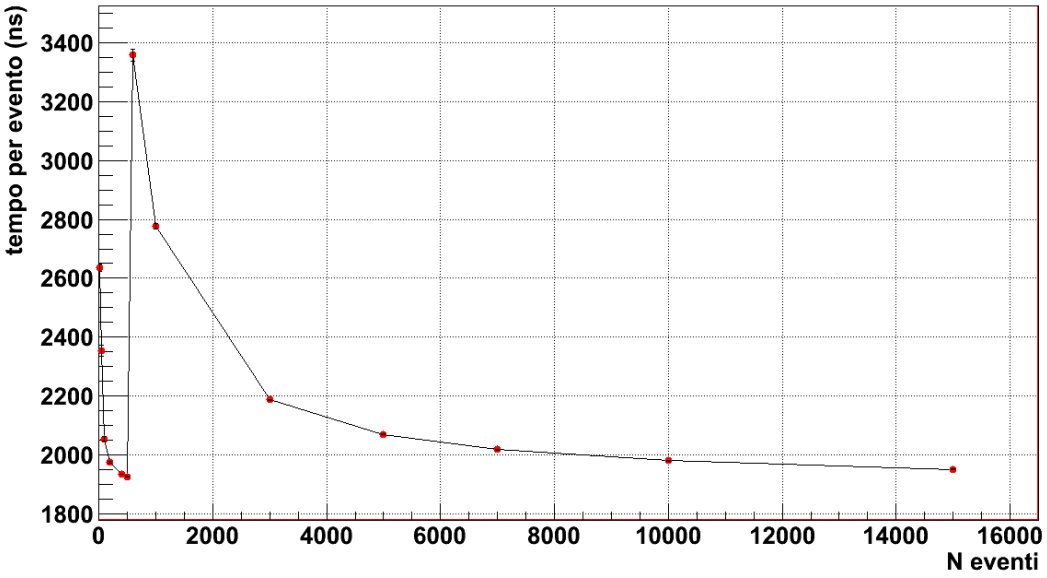


Figura 4.9: Tempo di esecuzione del *kernel*, per singolo evento, al variare del numero di eventi processati.

complessiva di 90 minuti. Come si può notare dal grafico (4.11), il tempo di esecuzione del *kernel* non subisce grandi fluttuazioni dovute all'uso continuato della GPU, infatti i suoi valori sono ben descritti da una Gaussiana di deviazione standard 1.6 ns e media 1934 ns (vedi grafico 4.11), anche se si notano 10 eventi che si discostano dalla media, di non più di 20 ns. Nonostante non siano compatibili con una coda Gaussiana, questi eventi sono completamente ininfluenti sulla stabilità temporale del programma dato che lo spostamento è solo del 1% rispetto alla media.

Questo ciclo di 90 minuti è stato utilizzato anche per verificare l'andamento della temperatura della GPU su lunghi periodi. La ATI Radeon HD 5970 contiene due GPU completamente

numero eventi	tempo di esecuzione per evento (ns)
20	2635 ± 12
40	2353 ± 19
100	2053 ± 8
200	1974 ± 2
400	1933 ± 1
500	1925 ± 2
600	3359 ± 21
1000	2776 ± 10
3000	2186 ± 3
5000	2069 ± 2
7000	2017 ± 1
10000	1979 ± 1
15000	1950 ± 1

Tabella 4.5: Tempo di esecuzione del *kernel* per evento al variare del numero di eventi processati.

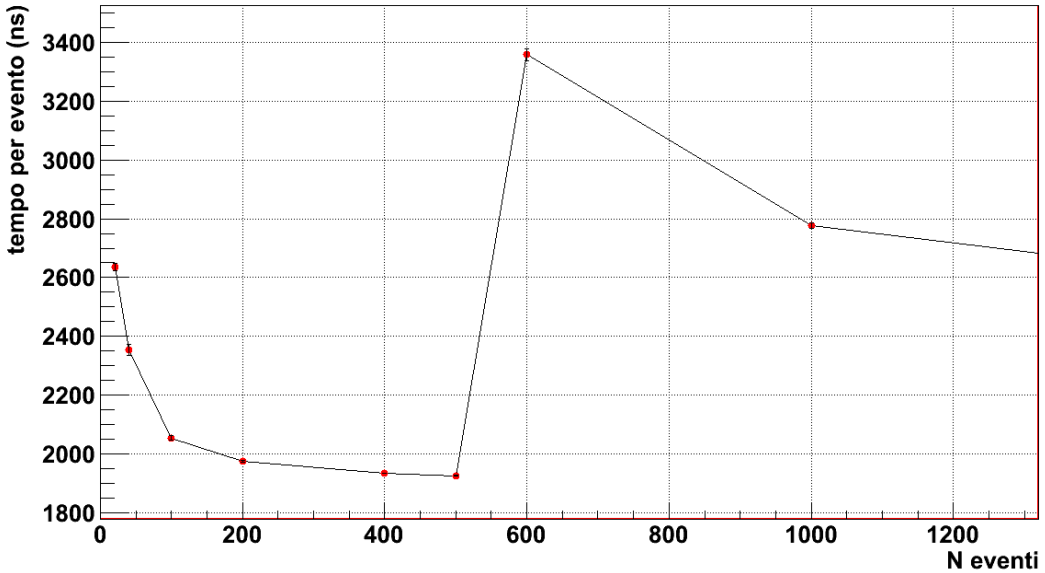


Figura 4.10: Zoom del tempo di esecuzione del *kernel*, per singolo evento, al variare del numero di eventi processati.

uguali di cui, per i calcoli in openCL, possiamo usarne solo una[21]. Di seguito sarà mostrato l'andamento della temperatura delle due GPU in due diverse esecuzioni del *kernel*. La prima esecuzione (vedi figura 4.12) è stata usata anche per la misura della stabilità temporale e consiste nel processamento di 400 eventi per 5000 ripetizioni: si sono usati 400 eventi perché risiedono nella zona che sarà utilizzata, più probabilmente, nell'utilizzo del *kernel* come trigger. Nel secondo caso (vedi figura 4.13) si è voluto studiare l'andamento della temperatura se usiamo la GPU in un ciclo di 1000 ripetizioni, in cui elabora 30000 eventi, per un totale di 21 minuti.

Come si vede dal primo grafico (4.12) la temperatura nei 90 minuti non si discosta molto dalla temperatura iniziale delle schede di 46 °C, mentre nel secondo caso (4.13) la temperatura aumenta rapidamente fino a stabilizzarsi con un plateau attorno a 57 °C,

[21] Il programma ATI Stream SDK supporta al momento solo una delle due GPU.

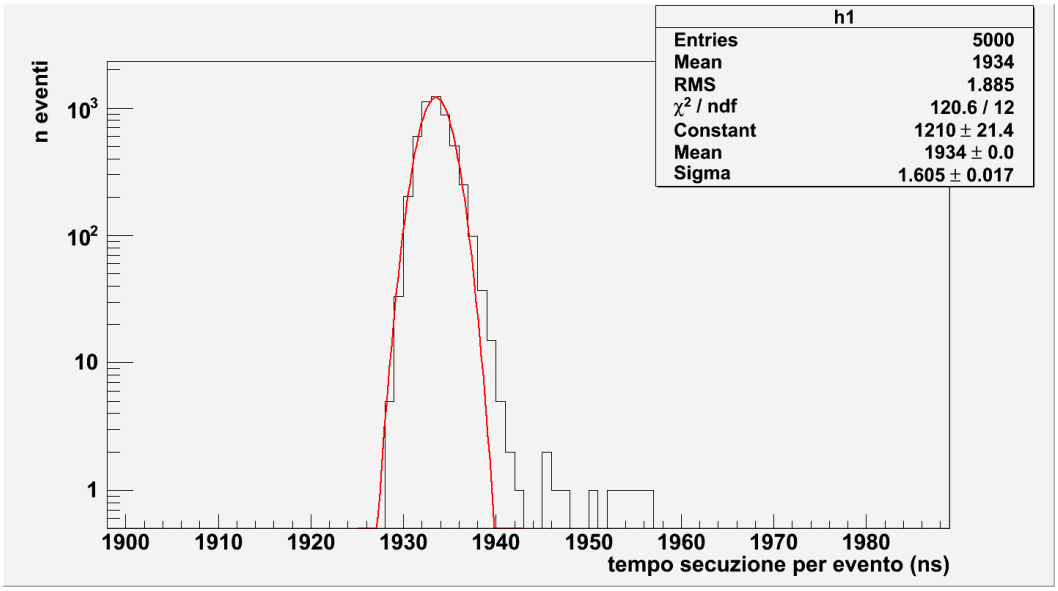


Figura 4.11: fit della stabilità temporale del *kernel*

comunque lontano dalla massima temperatura che la scheda può sopportare, visto le GPU in genere sopportano temperature superiori a 90 °C.

L'ultima verifica consiste nel controllare che l'aumento di temperatura non comporti variazioni nel tempo di esecuzione del *kernel*. Per questa verifica abbiamo usato il ciclo in cui l'esecuzione di 30000 eventi veniva ripetuta 1000 volte. Dal confronto dell'istogramma realizzato con le prime 40 ripetizioni del ciclo (vedi istogramma 4.14a), in cui la temperatura della scheda è inferiore a 49 °C, con quello contenente i tempi di esecuzione di tutte le ripetizioni (vedi istogramma 4.14b), in cui la temperatura raggiunge i 58 °C, si nota che non ci sono variazioni della sua forma. Quindi la temperatura della GPU, almeno fino a 58 °C, non influenza il tempo di esecuzione del *kernel*. Ci siamo preoccupati dell'influenza della temperatura sul tempo di esecuzione dell'algoritmo perché la GPU potrebbe avere un sistema di controllo che in caso di temperature elevate abbassi il *clock*.

4.8 cambiamento file d'ingresso

Nella sua versione definitiva il programma non riceverà in ingresso i file con le coordinate degli *hit* e dei centri, ma un unico flusso di dati che elencherà i PMT colpiti mediante un indice (4.15). Il programma *host*, quindi, dovrà trasmettere al *kernel*, oltre al vettore con il numero di *hit* per evento, solo un vettore monodimensionale con i numeri dei PMT colpiti al posto dei due vettori bidimensionale contenenti le coordinate comportando, così, un guadagno nel tempo di trasferimento dalla memoria dell'*host* alla *global memory*. I PMT saranno numerati in un modo appositamente scelto per questa applicazione in modo da velocizzare l'esecuzione dell'algoritmo e gli *hit* dovranno contenere questi indici. Di seguito sono mostrati i sistemi di equazioni con cui ho trasformato le coordinate dei PMT in un indice n (formula 4.1) e viceversa (formula 4.2)[22]. Posizionando, all'interno del *kernel*, la trasformazione dal indice n del PMT alle sue coordinate al momento del trasferimento dei dati dalla *global memory* alla *local memory*, il tempo di esecuzione del *kernel* resta invariato.

$$\text{indice del PMT} = \begin{cases} a \equiv (n \bmod 65) = floor[(x + 28.8)/0.9] \\ n = 65 \cdot floor[(y + 28.0592)/3.1177] + a \end{cases} \tag{4.1}$$

[22]floor è una funzione che arrotonda il valore all'interno della parentesi all'intero inferiore più vicino.

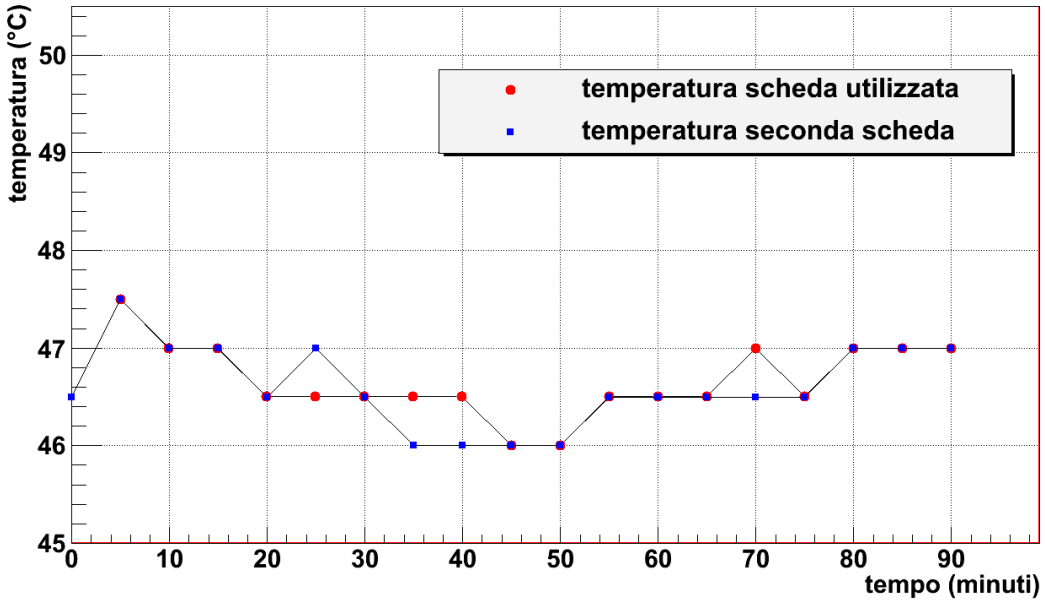


Figura 4.12: Andamento della temperatura del *kernel* mentre processa di 400 eventi per 5000 ripetizioni

$$\text{coordinate del PMT} = \begin{cases} x = -28.8 + 0.9 \cdot (n \bmod 65) \\ y = -28.0592 + 1.55885 \cdot (n \bmod 2) + 3.1177 \cdot floor(n/65) \end{cases} \quad (4.2)$$

Per ottenere delle formule computazionalmente efficienti ho inscritto lo *spot* di PMT in un rettangolo infatti dalla figura (4.15) si nota che non tutti i valori degli indici sono utilizzati. I valori -28.8 e -28.0592 che si trovano nelle formule sono le coordinate x e y in cm del primo indice della griglia rettangolare mentre 0.9 cm e 3.1177 cm sono rispettivamente il passo tra due indici consecutivi e due righe[23] di indici consecutive.

Anche se per semplicità, fino a questo punto, si sono considerati come possibili candidati centri i centri dei PMT, non c'è alcuna ragione per mantenere questa identità, e si è quindi studiata l'ottimizzazione delle scelte della griglia di centri candidati, sia per quanto riguarda il numero di punti sia per il loro passo, in modo da cercare la disposizione migliore che sia un buon compromesso tra velocità di esecuzione del *kernel* e la sua risoluzione spaziale. Di seguito sono mostrati in un grafico (4.16) ed in una tabella (4.6) i risultati ottenuti, come risoluzione e tempo di esecuzione del *kernel* per evento, con le varie griglie di punti che sono state provate. Nella tabella (4.6) sono presenti: l'indice di una numerazione per distinguere i punti nel grafico, il numero totale di punti della griglia, le coordinate del primo punto della griglia, il numero di passi della griglia, il passo esistente tra i punti, la risoluzione spaziale dell'algoritmo[24] ed il tempo di esecuzione per evento (processando 400 eventi) con il suo errore.

Dal grafico (4.16) si nota che le migliori disposizioni di possibili centri sono la numero 3 e la numero 12, entrambe comportano una risoluzione migliore del *kernel* che utilizza i PMT come possibili centri. In tal caso si aveva un risoluzione di 3.96 cm (vedi tabella 4.3) mentre le due griglie di punti indicate comportano rispettivamente una risoluzione di 3.12 cm e 3.27 cm equivalente, circa, al doppio del diametro di un PMT[25]. Tra le due griglie, la

[23] Ogni riga contiene 64 indici.
[24] Per misurare la risoluzione sono stati usati istogrammi con 16 *bin* (vedi sezione 4.7) facendo processare 10000 eventi per avere una statistica significativa.
[25] Il diametro di un PMT è circa $1,6$ cm.

griglia	punti totali	primo punto (x,y) (cm)	N passi (X,Y)	passo (x,y) (cm)	risoluzione (cm)	tempo esecuzione per evento (ns)
1	1045	(-27,-27)	(55,19)	(1,3)	3.43	2052.53 ± 1,44
2	885	(-29,-28)	(59,15)	(1,4)	3.51	1823.13 ± 1,38
3	870	(-29,-28)	(30,29)	(2,2)	3.12	1823.97 ± 1,66
4	784	(-27,-26)	(56,14)	(1,4)	3.68	1556,07 ± 1,08
5	765	(-25,-21)	(51,15)	(1,3)	4.78	1556,07 ± 1,08
6	756	(-27,-26)	(28,27)	(2,2)	4.46	1558.27 ± 1,46
7	742	(-26,-26)	(53,14)	(1,4)	4.41	1556,07 ± 1,08
8	589	(-30,-27)	(31,19)	(2,3)	3.77	1329.37 ± 1,58
9	516	(-21,-22)	(43,12)	(1,4)	4.18	1064,9 ± 1.12
10	506	(-22,-21)	(23,22)	(2,2)	4.35	1064,9 ± 1.12
11	493	(-28,-24)	(29,17)	(2,3)	4.39	1064,9 ± 1.12
12	361	(-27,-27)	(19,19)	(3,3)	3.27	832.3 ± 0.98

Tabella 4.6: Risoluzione ed il tempo di esecuzione del *kernel* per evento, processando 400 eventi, per le varie griglie di candidati centri.

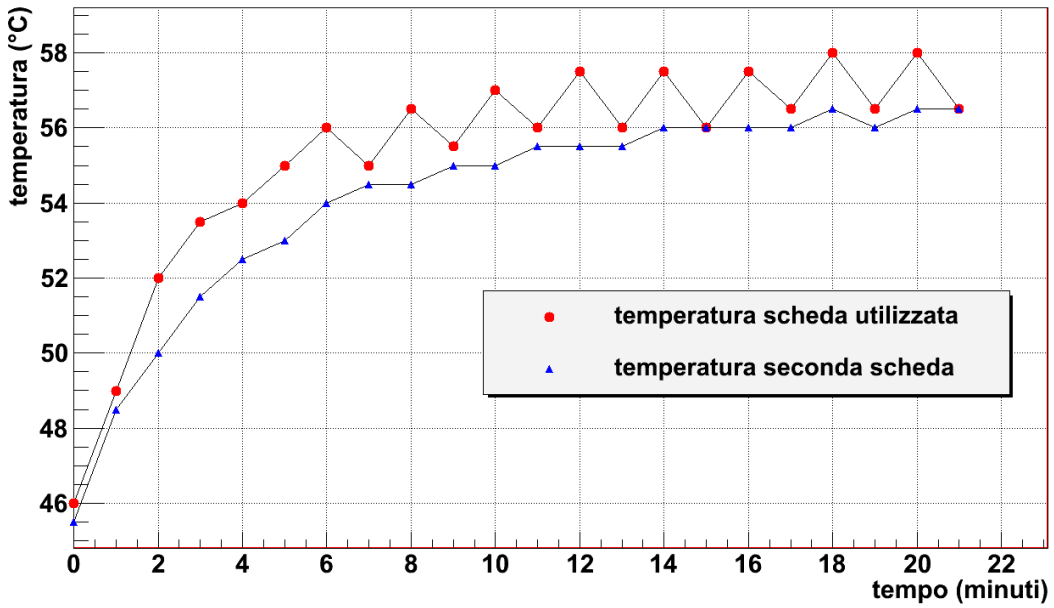
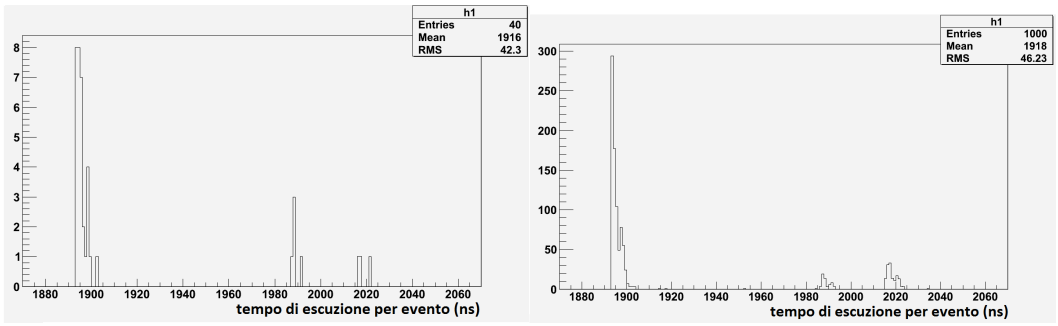


Figura 4.13: Andamento della temperatura del *kernel* mentre processa di 30000 eventi per 1000 ripetizioni



(a) *Prime 40 ripetizioni del kernel.* (b) *Tutte le 1000 ripetizioni del kernel.*

Figura 4.14: Tempo di esecuzione del *kernel* che processa 30000 eventi.

più conveniente sicuramente è la numero 12 perché comporta un miglioramento nel tempo di esecuzione di un fattore > 2 con un peggioramento della risoluzione solo del 4.6%. Di seguito è mostrato con una tabella (4.7) ed un grafico (4.17) l'andamento del tempo di esecuzione per evento del *kernel* al variare del numero di eventi processati contemporaneamente con l'uso di questa griglia.

L'andamento temporale è sempre lo stesso anche se il minimo si è spostato a 1200 eventi. Lo spostamento è dovuto alla riduzione dell'utilizzo della *local memory* dovuto al minor numero di dati trasferiti alla GPU. A 1300 eventi è visibile un punto con grande errore nel punto di transizione tra il minimo e il picco: la sua fluttuazione è dovuta alla presenza di due popolazioni, una del picco ed una del minimo, come già spiegato nella sezione 4.1.

Infine è possibile utilizzare una griglia più rada per ridurre ulteriormente il tempo di esecuzione a discapito della risoluzione spaziale dell'algoritmo infatti ad esempio con una griglia di 169 punti (13X13) si è ottenuto un tempo di esecuzione per 1000 eventi analizzati di 430.4 ± 0.5 ns e una risoluzione spaziale di 4.99 cm.

In conclusione si possono confrontare i risultati delle ottimizzazioni e dei miglioramenti

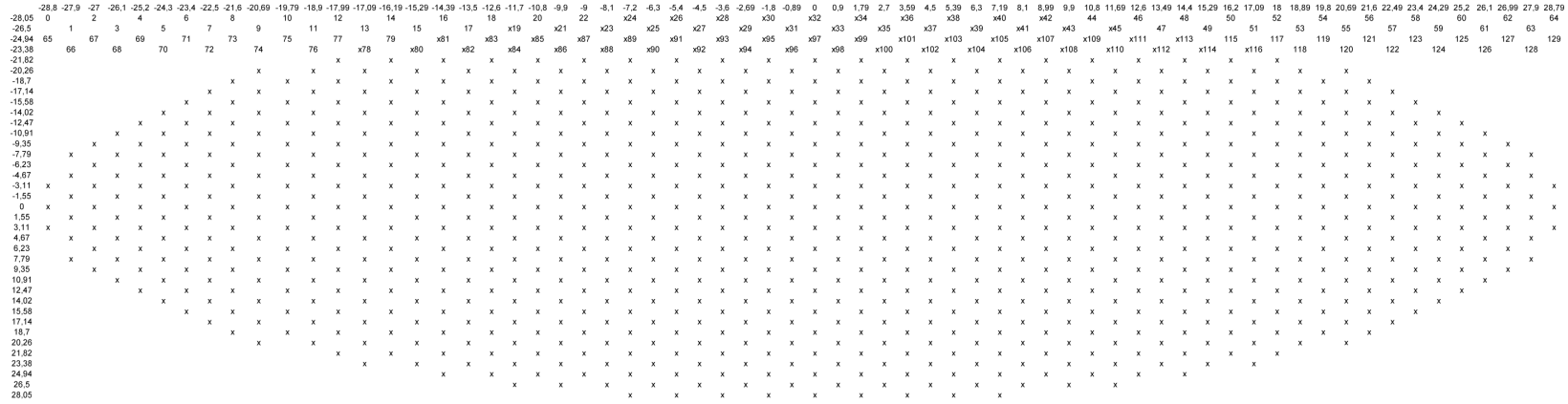


Figura 4.15: rappresentazione grafica di uno *spot* di PMT insieme ad un esempio, nelle prime 4 righe, della loro numerazione per gli *hit*, ai bordi sono presenti le coordinate x,y in cm

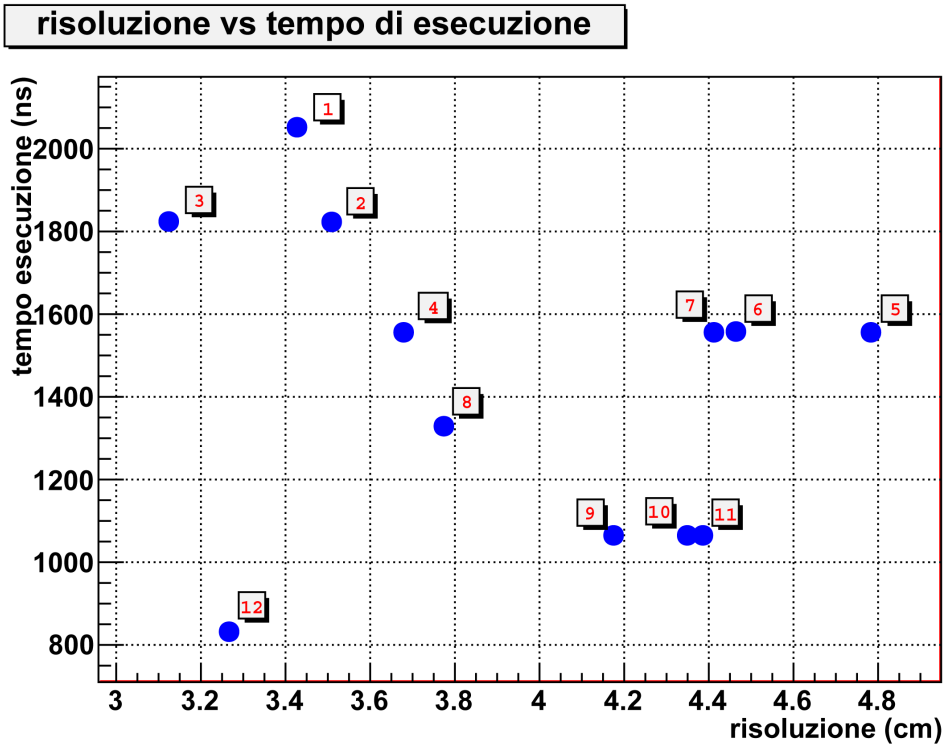


Figura 4.16: Tempo di esecuzione per evento del *kernel* e risoluzione spaziale per diverse scelte di griglie dei punti (vedi testo).

del *kernel* analizzati in questo capitolo nel grafico (4.18).

4.9 Tempi di trasferimento

Per l'utilizzo dell'algoritmo per le GPU come *trigger* di livello 0 bisogna tenere in considerazione oltre al suo tempo di esecuzione anche il tempo di trasferimento dei dati dalla memoria della CPU a quella della GPU insieme al tempo di trasferimento dei risultati dalla GPU alla CPU. Per questo motivo abbiamo misurato questi tempi con lo stesso metodo utilizzato per misurare il tempo di esecuzione (vedi sezione 4.1). Il grafico (fig. 4.17) mostra degli andamenti a scalini con un aumento sostanziale dell'errore di misura nei trasferimenti dei dati dalla CPU alla GPU per un numero di eventi processati superiore a 7000. Questo errore è dovuto ad un comportamento (fig. 4.20) simile a quello riscontrato nella zona dello scalino dei grafici dell'andamento del tempo di esecuzione in funzione del numero di eventi processati (vedi sezione 4.1) ma la sua motivazione dovrebbe essere diversa anche se tutt'ora non è chiara.

numero eventi	tempo di esecuzione per evento (ns)
20	1525.8 ± 10.88
40	1209.4 ± 10.64
100	938.83 ± 5.43
200	868.93 ± 2.38
400	833.23 ± 1.10
600	820.83 ± 0.70
1000	810.57 ± 0.50
1200	807.97 ± 0.41
1300	1170.27 ± 345.77
1420	1427.4 ± 2.85
2000	1245.8 ± 1.77
3000	1094.87 ± 1.36
5000	975.23 ± 0.77
7000	923.8 ± 0.55
10000	885.33 ± 0.48
15000	855.13 ± 0.35

Tabella 4.7: Tempo di esecuzione per evento del *kernel*, che usa la griglia numero 12 di punti, al variare del numero di eventi.

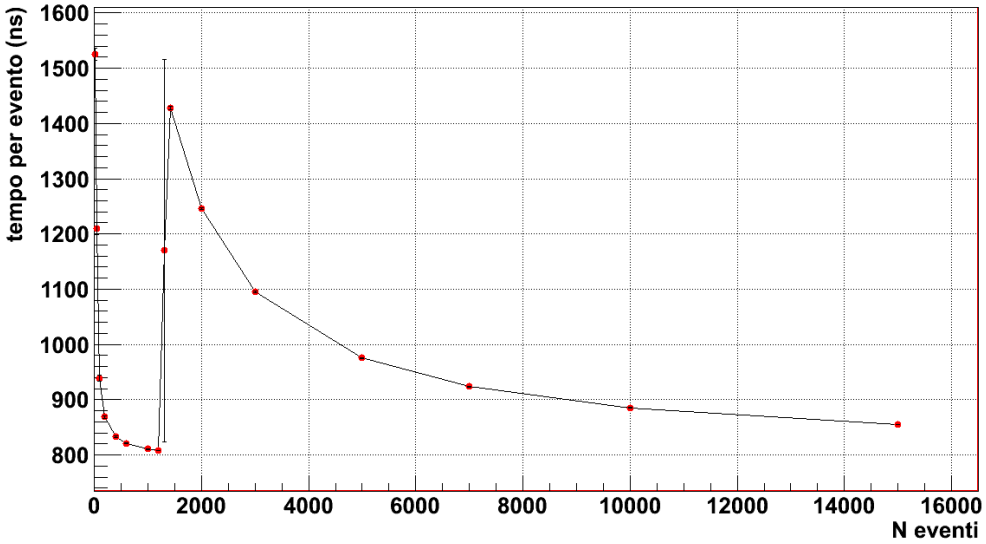


Figura 4.17: Tempo di esecuzione per evento del *kernel*, che usa la griglia numero 12 di punti, al variare del numero di eventi processati. Gli errori sono cosi piccoli da non essere visibili tranne che per il punto che si trova nella discontinuità per il motivo spiegato nella sezione (4.1).

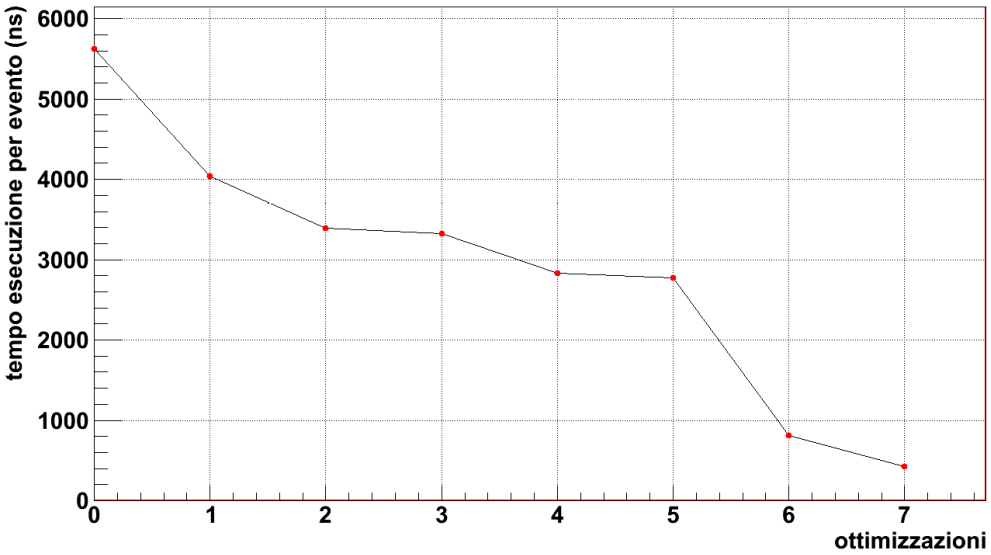


Figura 4.18: Confronto del tempo di esecuzione per evento del *kernel* dopo ogni ottimizzazione.
legenda:
0 = algoritmo non ottimizzato
1 = prime ottimizzazioni
2 = ottimizzazione del numero di *work-item*
3 = ottimizzazione delle istruzioni
4 = ottimizzazione degli *wavefront*
5 = uso del quadrato delle distanze
6 = griglia 19 X 19
7 = griglia 13 X 13

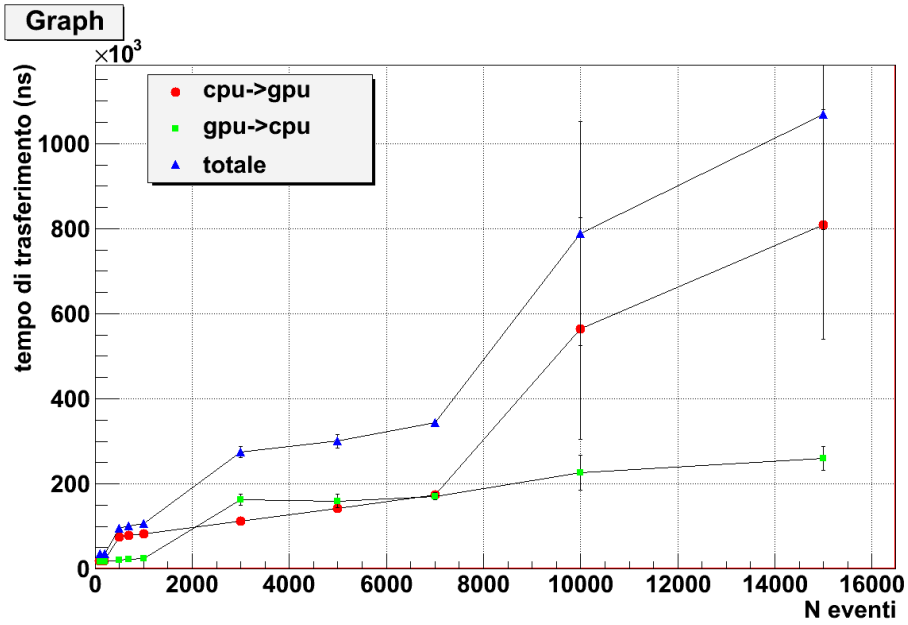


Figura 4.19: Andamento dei tempi di trasferimento dei dati, dei risultati e totali in funzione del numero di eventi processati dall'algoritmo.

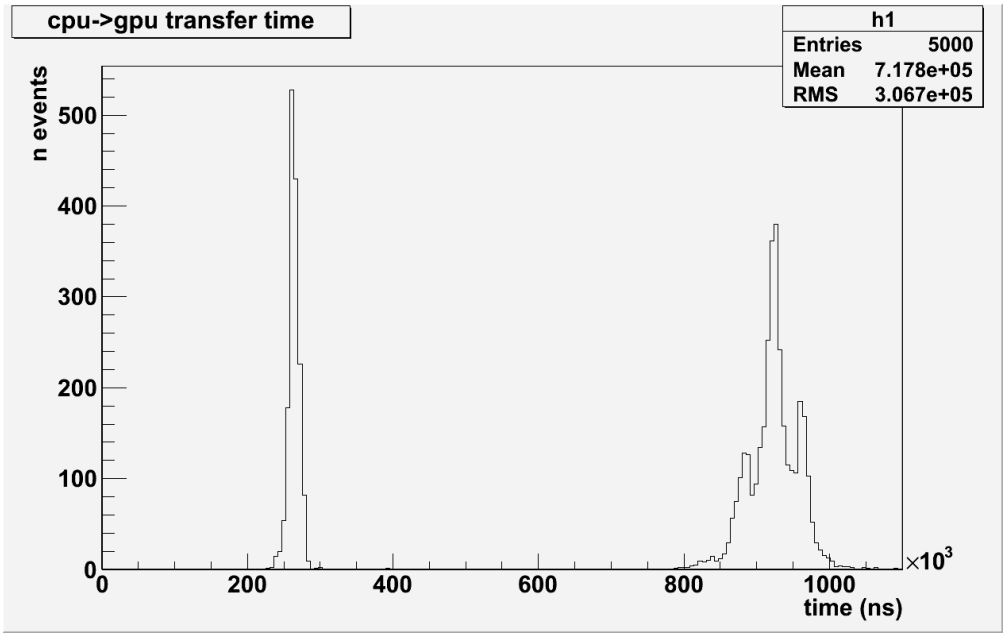


Figura 4.20: Istogramma dei tempi di trasferimento di dati dalla memoria della CPU a quella della GPU nella regione che presenta misure con una grande incertezza 4.19.

CAPITOLO 5

STUDIO DI UN ALGORITMO DI TRIGGER L1 PER LE CAMERE A *STRAW*

Indice

In seguito all'apprendimento dei metodi e delle tecniche di ottimizzazione dei programmi per le GPU, ci siamo occupati di sviluppare, e successivamente implementare su una GPU, un algoritmo di trigger di livello 1 per le camere a *straw*. Lo scopo di questo algoritmo di trigger consiste in una semplice ricostruzione delle tracce e del vertice di decadimento per riconoscere e rigettare gli eventi $K^+ \to \pi^+\pi^+\pi^-$ senza perdere troppi eventi di segnale $K^+ \to \pi^+\nu\bar{\nu}$. Indicativamente, visto lo scopo dell'esperimento, saremo più preoccupati di mantenere elevata l'efficienza per il segnale che di ridurre il fondo; il compromesso tra queste due esigenze sarà scelto soltanto in base all'analisi dei primi dati reali dell'esperimento. Ci siamo focalizzati su questo particolare decadimento perché è quello che subisce meno tagli dal trigger di livello 0. Nonostante abbia un *branching ratio* di 5.58%, al livello 1 compone il 45.6% del totale degli eventi. Inoltre le camere a STRAW non sono utilizzate al livello 0 a causa della loro scarsa localizzazione temporale, e rappresentano un rivelatore ideale per ottenere informazioni su questo tipo di decadimento a tre tracce.

A causa delle dimensioni dei tubi (*straw*) delle camere (di raggio 4.7 mm) e della velocità di deriva, il *leading time* degli hit varia tra 0 e 160 ns dopo l'istante di passaggio della particella, a seconda della distanza dal filo a cui è passata la traccia. Visto che ci aspettiamo un rate di eventi nel rivelatore di circa 10 MHz, il *trigger* di livello 0 deve prendere una decisione per ogni evento in media ogni 100 ns rendendo problematico l'utilizzo dell'informazione delle camere a fili.

Lo studio dell'utilizzo delle camere nel *trigger* si è sviluppato in due parti: nella prima si è studiato l'algoritmo per la reiezione dei decadimenti $K^+ \to \pi^+\pi^+\pi^-$ considerando

ogni evento singolarmente, mentre nella seconda parte si è considerato l'effetto del *pileup*, ovvero della sovrapposizione di eventi avvenuti a tempi diversi. Questo fenomeno è dovuto alla finestra temporale di lettura delle camere, che essendo di 210 ns, a causa del tempo massimo di deriva, comporta che nel 91.7% degli eventi, nelle singole camere, ci sia una sovrapposizione di fino a 11 hit di *pileup* come mostrato dalla simulazione Monte Carlo in figura 5.1.

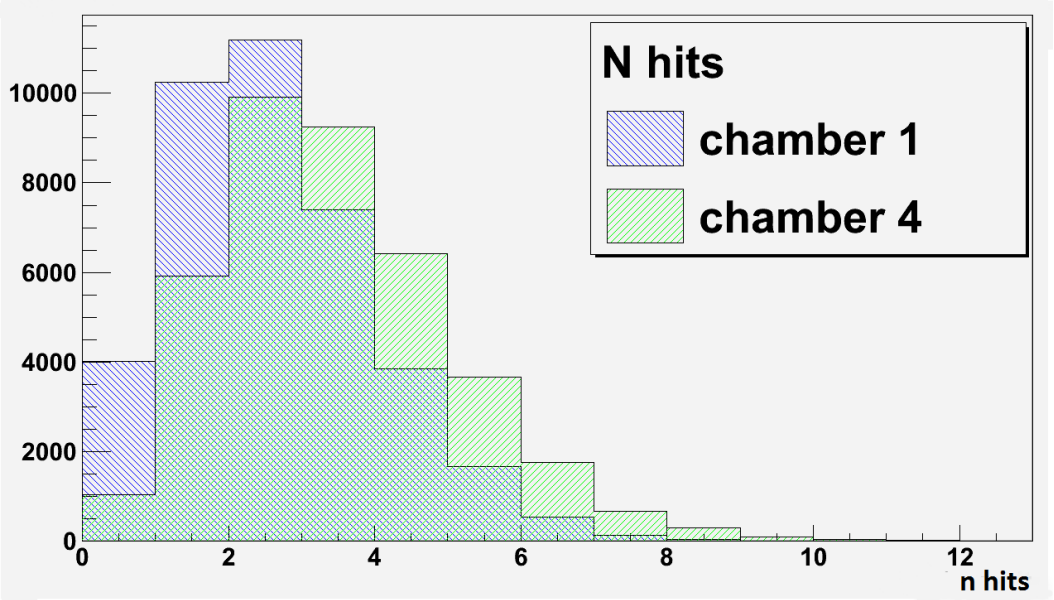


Figura 5.1: Distribuzione del numero di *hits* sovrapposti all'interno della finestra di lettura nella camera 1 e nella camera 4.

5.1 Studio Dell'Algoritmo Senza Pileup

Per lo sviluppo dell'algoritmo di trigger nel caso senza *pileup*, abbiamo generato 1000 eventi di segnale $K^+ \to \pi^+\nu\bar{\nu}$ nella regione fiduciale, compresa tra 105 m e 165 m dal bersaglio di produzione dei K^+, e 1000 eventi di $K^+ \to \pi^+\pi^+\pi^-$ in una regione più lunga, tra 103 m e 183 m dal bersaglio di produzione[1], utilizzando la simulazione Monte Carlo ufficiale della collaborazione *NA62* [31] basata sul *package GEANT4* [2].

Visto che l'algoritmo dovrà essere usato come trigger di livello 1, in cui ci aspettiamo una frequenza in ingresso di 200 KHz, la decisione per ogni evento dovrebbe essere presa in media ogni $5\mu s$. Per questo motivo ci siamo focalizzati su una semplice e veloce ricostruzione delle tracce ed abbiamo utilizzato solo la proiezione y delle camere ottenendo una ricostruzione delle tracce sul piano y-z. Il campo magnetico dello spettrometro di NA62 è diretto lungo l'asse y quindi le tracce sul piano y-z non saranno deviate e risulterà più semplice e veloce ricostruirle. In questo algoritmo, quindi, non viene sfruttata la curvatura delle tracce dovuta al campo magnetico in quanto, per il suo scopo, non è necessario conoscere l'impulso delle particelle ma solo verificare la presenza di tre tracce provenienti da un unico decadimento.

Per la ricostruzione delle tracce abbiamo usato, come coordinate degli *hits*, solo le posizioni dei centri dei fili colpiti, senza utilizzare informazioni temporali per calcolare le coordinate fini degli *hits* all'interno dei tubi. L'informazione data dal *leading time*, ovvero il tempo impiegato dai primi elettroni prodotti dalla traccia carica per arrivare al filo, combinata

[1] 103 m indica il punto di inizio della regione di decadimento mentre a 183 m si trova la posizione della prima camera a *straw*.

con la conoscenza della velocità di deriva, permette di calcolare con precisione la distanza della traccia dal filo, ma questa informazione non viene usata in questo algoritmo. In questo modo si perderà in risoluzione spaziale ma guadagnando molto in semplicità e velocità dell'algoritmo.

Dato che ogni camera è composta da 4 piani per ogni vista, avremo molti *hits* e quindi troppe possibili combinazioni, che aumentano il tempo di esecuzione dell'algoritmo di ricostruzione delle tracce in modo inaccettabile; per questo motivo abbiamo preso gli eventi di segnale (composti da un unica traccia) e abbiamo riempito un istogramma con tutte le distanze lungo y tra due *hits* dei 4 piani della vista y (fig. 5.2). Come si può vedere dall'istogramma, la maggior parte degli *hits* di un unica vista sono distanti al massimo 13.3 mm l'uno dall'altro; abbiamo quindi deciso di raggruppare tutti gli *hits* che si trovano entro questa distanza tra loro ed assegnare come coordinata y del *cluster* il valor medio delle loro coordinate. Utilizzando 13.3 mm per questa procedura non dovremmo raggruppare *hit* provenienti da tracce diverse, tranne in qualche raro caso, in quanto il valore è inferiore alla distanza tra due *straw* dello stesso piano (17.6 mm). Utilizzando solo la vista y, a causa

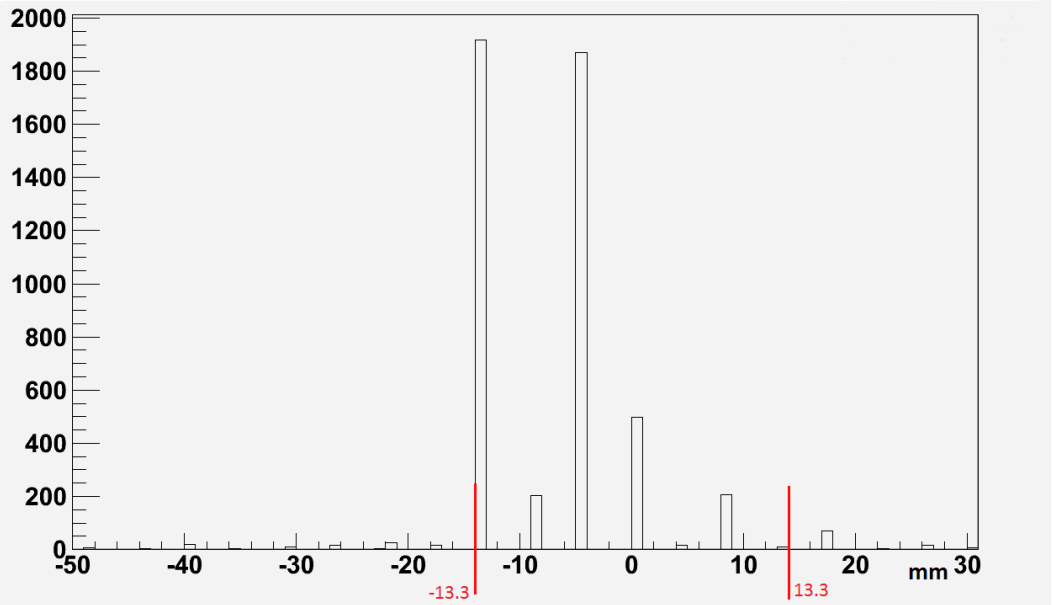
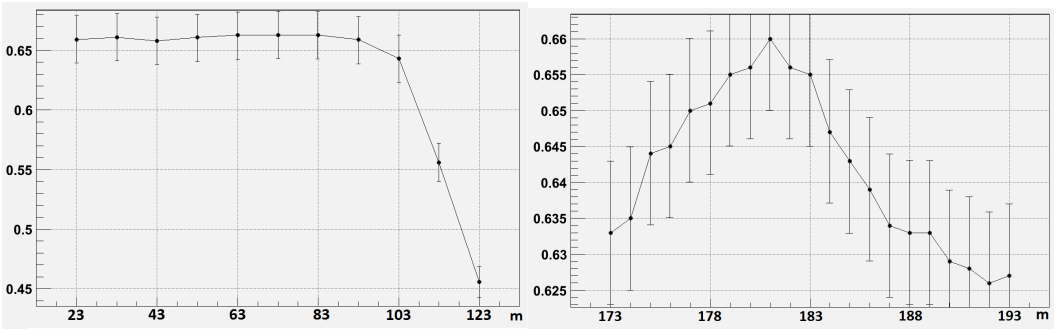


Figura 5.2: Distribuzione delle distanze tra due fili colpiti nella vista y di una camera da una singola traccia.

delle inefficienze e della mancanza dei fili nella regione centrale, perdiamo troppi *hits* ($\sim 19\%$ del totale) quindi abbiamo deciso di utilizzare le combinazioni delle viste u e v per ottenere *hits* addizionali per la vista y anche se per via delle combinazioni, uguali al quadrato del numero di *hit* (n^2), si includono molti *hits* falsi, che potrebbero generare tracce false ed aumentare il tempo di esecuzione dell'algoritmo. Per ridurre al minimo il numero di *hits* falsi prodotti dalle combinazioni delle viste u e v abbiamo rimosso tutte le coppie delle due viste la cui combinazione genera *hits* già presenti nella vista y, in modo da non considerare due volte gli stessi *hits* e ridurre le combinazioni delle viste u e v.

Dopo il completamento della lista di *hits* nella vista y utilizzando gli hit addizionali delle viste u e v, l'algoritmo esegue la ricostruzione delle tracce in tale piano verticale partendo da una retta passante per due *hits* in diverse camere e verificando, per limitare il numero tra tracce false ricostruite, se essa interseca l'asse z (asse nominale del fascio nel piano verticale) in una regione compresa tra 63 e 181 m. I limiti di tale regione sono stati scelti in modo da massimizzare la reiezione del fondo $K^+ \to \pi^+\pi^+\pi^-$ e minimizzare le false tracce: nelle figure

5.3a e 5.3b è possibile osservare la variazione della reiezione degli eventi $K^+ \to \pi^+\pi^+\pi^-$ al variare, rispettivamente, del taglio a monte e del taglio a valle.



(a) *Taglio a monte* (b) *Taglio a valle*

Figura 5.3: Percentuale di reiezione di $K^+ \to \pi^+\pi^+\pi^-$ in funzione dei tagli sulla regione di intersezione della retta con l'asse z.

Infine per considerare valida una traccia, verifichiamo che ci sia almeno un *hit* in un'altra camera che sia distante dalla retta meno di 3.4 mm: in figura 5.4 si vede che questo valore massimizza la reiezione del fondo.

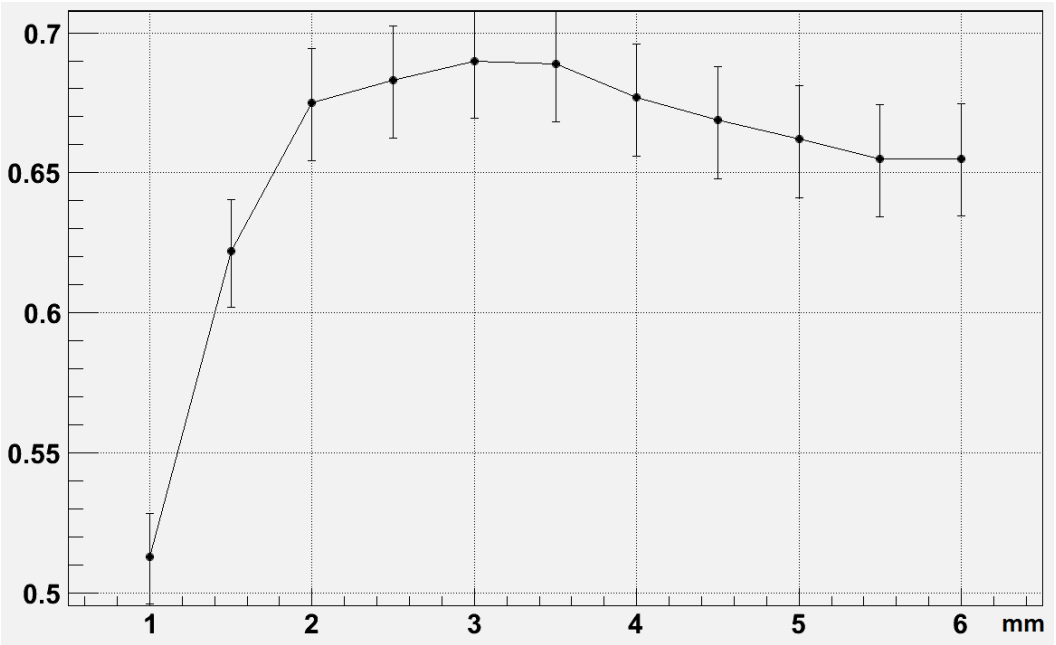


Figura 5.4: Percentuale di reiezione di $K^+ \to \pi^+\pi^+\pi^-$ in funzione del taglio sulla distanza del terzo *hit* dalla retta passante per i primi due.

Nella ricostruzione delle tracce ogni *hit* viene usato solo una volta: quando è assegnato ad una traccia valida viene eliminato dalla lista degli *hits* da considerare.

Una volta determinate le tracce valide vengono cercati i vertici di decadimento degli eventi $K^+ \to \pi^+\pi^+\pi^-$, per verificare la presenza di due tracce che si intersechino nella regione del fascio, ovvero entro una distanza lungo y dall'asse z pari a ~ 25 mm: questo valore corrisponde alla semi-larghezza massima del fascio prima delle camere. Il taglio in questa

variabile viene effettuato ad un valore ottenuto sommando ai 25 mm 2 volte la risoluzione dell'algoritmo in y (σ_y) per un totale di 34 mm (fig. 5.5).

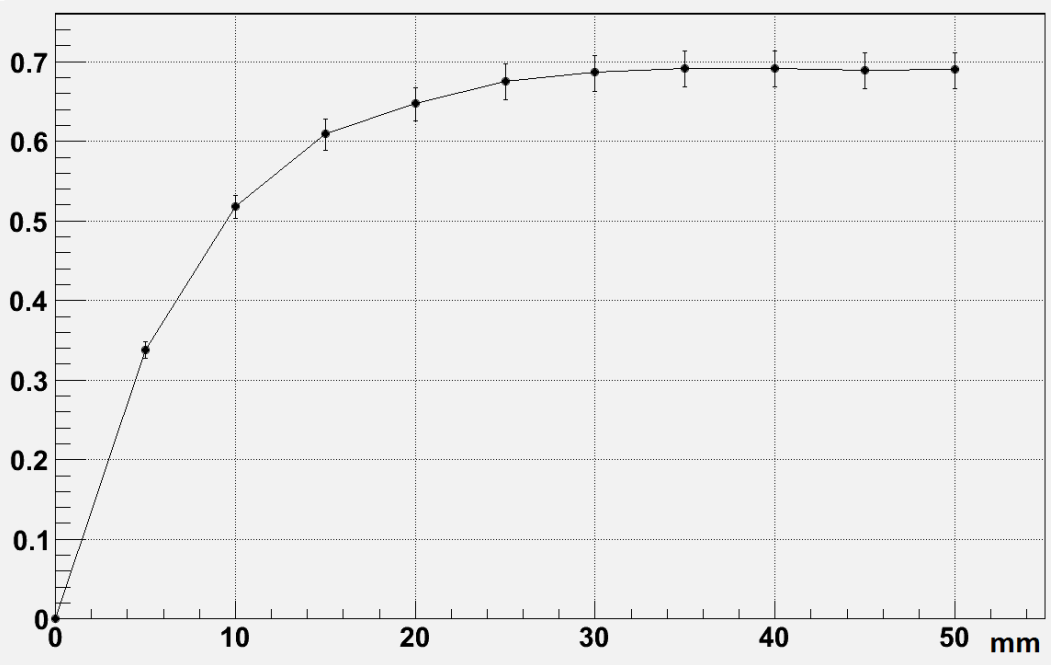


Figura 5.5: Percentuale di reiezione di $K^+ \to \pi^+\pi^+\pi^-$ variando il taglio sulla massima distanza del punto di intersezione delle 2 tracce dall'asse z.

Infine se è presente una terza traccia, affinché l'evento sia considerato un $K^+ \to \pi^+\pi^+\pi^-$ e quindi rigettato, l'intersezione delle prime due tracce proiettate nel piano yz deve essere distante lungo y meno di 21 mm da questa traccia. Non abbiamo scelto questo valore in modo da massimizzare la reiezione del fondo, visto che tale massimo si raggiunge con il valore che considera tutti gli eventi con la terza traccia come un evento $K^+ \to \pi^+\pi^+\pi^-$ (anche se in verità non provengono dallo stesso vertice) ma abbiamo utilizzato un valore pari a $3\sigma_y$ della risoluzione del punto di intersezione e della terza traccia. Non sempre l'algoritmo riesce a ricostruire tutte e tre le tracce di un evento $K^+ \to \pi^+\pi^+\pi^-$: nella maggioranza dei casi ne ricostruisce due e alcune volte solo una (fig. 5.7); questo è dovuto solo in minima parte ad errori di ricostruzione dell'algoritmo stesso, nella maggior parte dei casi è dovuto alla creazione dei *cluster*, all'accettanza delle camere (fig. 5.6) e, in misura ridotta, all'efficienza dei fili (fig. 5.6 e fig. 5.7). Abbiamo riscontrato che in alcuni casi vengono raggruppati in *cluster hits* provenienti da diverse tracce o *hits* di una traccia con *hits* di un raggio delta.

Gli eventi con un vertice che rispetta questi parametri vengono considerati eventi di $K^+ \to \pi^+\pi^+\pi^-$ e pertanto vengono rigettati.

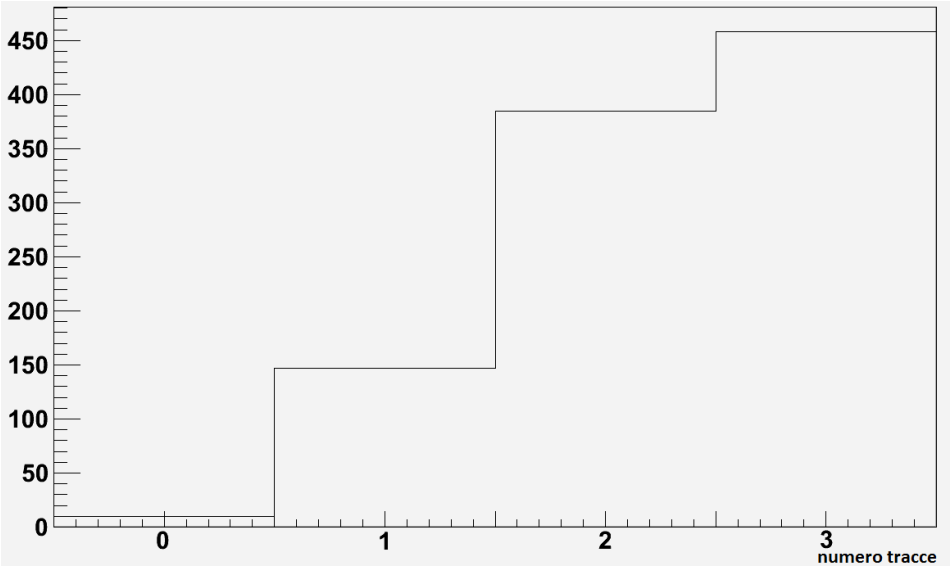


Figura 5.6: Distribuzione del numero tracce di un evento $K^+ \to \pi^+\pi^+\pi^-$ all'interno dell'accettanza del rivelatore.

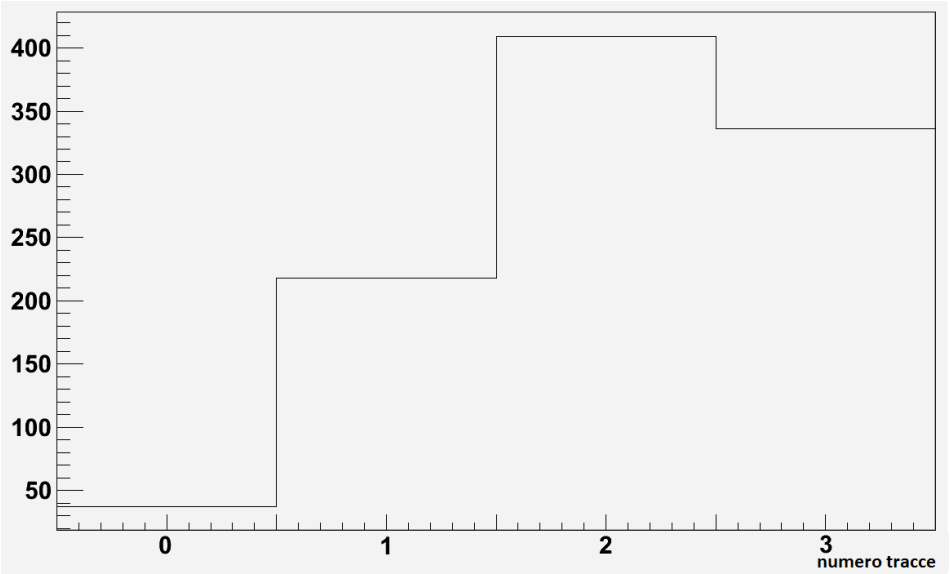


Figura 5.7: Distribuzione del numero di tracce di un evento $K^+ \to \pi^+\pi^+\pi^-$ identificate dall'algoritmo, quindi considerando oltre all'accettanza del rivelatore, l'efficienza dei fili e gli errori dovuti alla formazione dei *cluster*.

5.2 Analisi Dell'Algoritmo

Durante la definizione dell'algoritmo abbiamo proceduto ad una sua analisi per verificare la correttezza dei risultati. In primo luogo abbiamo misurato la risoluzione in z e y con cui vengono trovati i vertici di decadimento sia per i casi a 2 tracce che per quelli a 3 tracce (fig. 5.8). Per il caso con il vertice a 2 tracce abbiamo riscontrato le seguenti risoluzioni

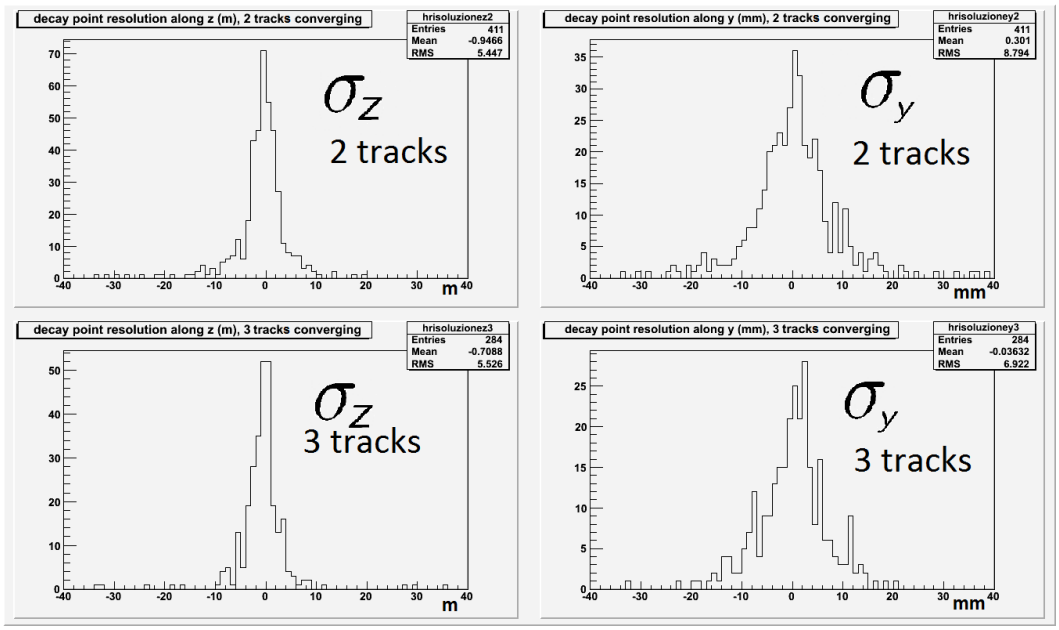
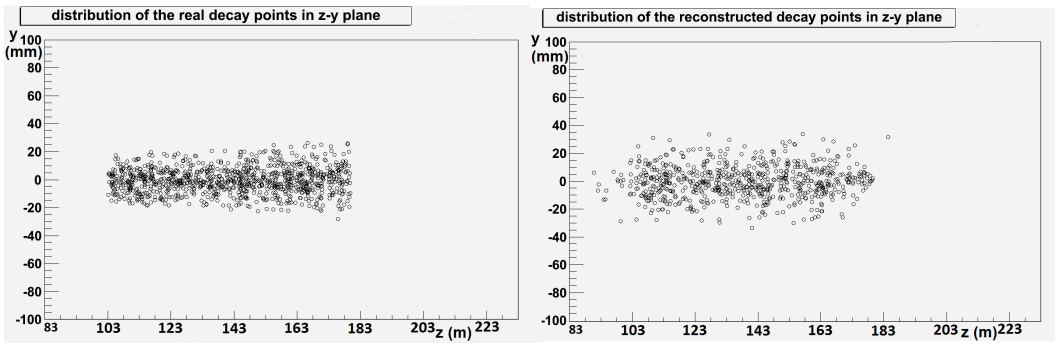


Figura 5.8: Risoluzione dell'algoritmo nelle coordinate z e y del vertice per eventi con 2 tracce e con 3 tracce.

rispettivamente in z e y: $\sigma_z = 2.05 \pm 0.13$ m e $\sigma_y = 4.97 \pm 0.29$ mm. Per il caso con il vertice a 3 tracce invece: $\sigma_z = 2.17 \pm 0.14$ m e $\sigma_y = 4.04 \pm 0.34$ mm. La grande disparità nelle risoluzioni in z e y è dovuta all'angolo delle tracce nel piano y-z, inferiore a 8 mrad rispetto all'asse z. Le risoluzioni non sono molto buone se confrontate con quelle ottenibili con un analisi completa delle informazioni delle camere, ma dobbiamo considerare la semplicità della ricostruzione che stiamo usando e ricordare che il suo scopo non è la precisione di ricostruzione delle tracce ma la sua velocità di esecuzione.



(a) (b)

Figura 5.9: Distribuzione dei vertici di decadimento generati con la simulazione Monte Carlo (a) e ricostruiti con l'algoritmo (b) nel piano y-z per eventi $K^+ \to \pi^+\pi^+\pi^-$.

Coerentemente con la risoluzione dell'algoritmo possiamo confrontare i punti di decadimento degli eventi $K^+ \to \pi^+\pi^+\pi^-$ ottenuti dalla simulazione Monte Carlo (fig. 5.9a) con i medesimi calcolati dall'algoritmo (fig. 5.9b).

Successivamente abbiamo confrontato il numero di tracce identificate dall'algoritmo[2] con il numero di tracce realmente ricostruite, sia per gli eventi di fondo (fig. 5.10) che per il segnale $\pi^+\nu\bar{\nu}$ (fig. 5.11). Come si può notare nelle figure, nella maggioranza dei casi

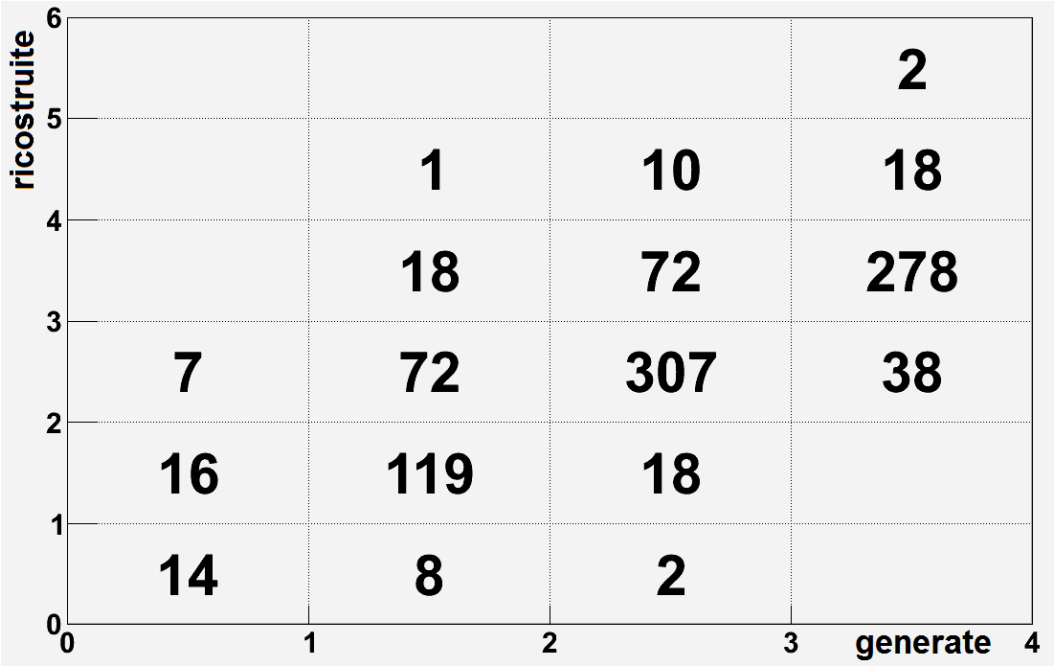


Figura 5.10: Confronto tra il numero di tracce ricostruite dall'algoritmo rispetto al numero di tracce generate (considerando oltre all'accettanza del rivelatore, l'efficienza dei fili e la creazione dei *cluster*) per eventi $K^+ \to \pi^+\pi^+\pi^-$

l'algoritmo riconosce il numero corretto di tracce con un efficienza del 95.1% per gli eventi di segnale ed un efficienza ridotta al 71.8% per gli eventi $K^+ \to \pi^+\pi^+\pi^-$. Nel caso del fondo l'efficienza si riduce perché essendoci più *hits* aumenta la probabilità che l'algoritmo crei più tracce finte collegando *cluster* non appartenenti alla stessa traccia, *cluster* formati da *hits* spuri dovuti a di raggi delta o da false combinazioni di *hits* delle viste u e v. Inoltre l'algoritmo può ricostruire più o meno tracce di quelle che dovrebbe a causa della diffusione multipla o della formazione di *cluster* con *hits* provenienti da una traccia e *hits* dovuti a raggi delta[3] o del taglio sull'intersezione delle tracce con l'asse z. Tale taglio è utile a ridurre il numero di tracce finte, ma che può anche far perdere qualche traccia vera a causa anche della limitata risoluzione in z.

Infine abbiamo misurato la capacità di reiezione dell'algoritmo per gli eventi di fondo $K^+ \to \pi^+\pi^+\pi^-$ (fig. 5.12), considerando identificati come tali gli eventi in cui tutte le tracce ricostruite convergono in un unico vertice[4] in modo da evitare perdite di eventi di segnale dovute ad una loro sovrapposizione accidentale con un decadimento $K^+ \to \pi^+\pi^+\pi^-$. Nell'istogramma 5.12 sono indicati con "*0 tracce*" gli eventi in cui l'algoritmo non ricostruisce nessuna traccia nonostante ci siano il numero di *hits* sufficienti alla sua ricostruzione; i motivi di queste perdite sono quelli elencati in precedenza. Non abbiamo considerato nel

[2]Nel misurare il numero di tracce identificate ho considerato l'accettanza del rivelatore, l'efficienza dei fili e la perdita di *hits* nella creazione dei *cluster*.

[3]L'inclusione di *hits* spuri comporta uno spostamento della coordinata del *cluster*.

[4]Nell'istogramma 5.12 sono gli eventi colorati di blu e viola.

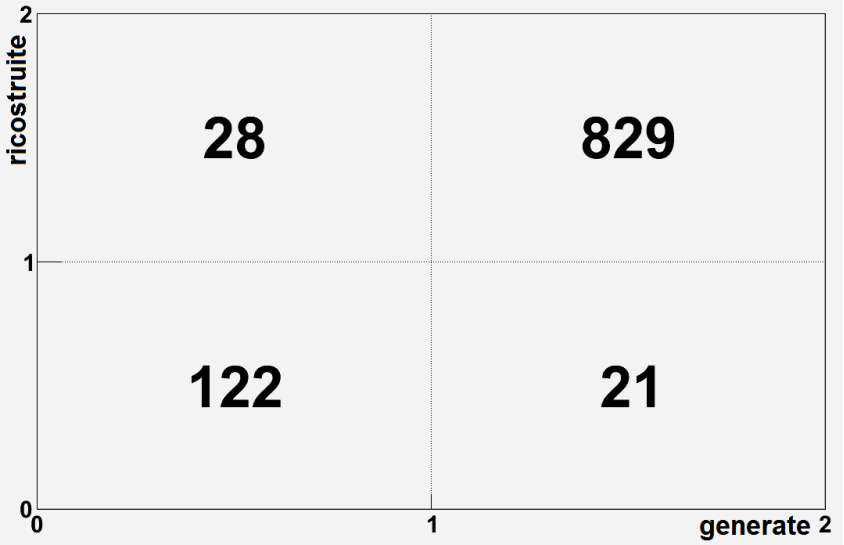


Figura 5.11: Confronto tra il numero di tracce ricostruite rispetto al numero di tracce generate dall'algoritmo (considerando oltre all'accettanza del rivelatore, l'efficienza dei fili e la creazione dei *cluster*) per eventi di segnale $K^+ \to \pi^+\nu\bar{\nu}$.

computo gli eventi in cui tutte le tracce siano fuori dall'accettanza delle camere a fili o siano presenti un numero di *hits* insufficienti alla ricostruzione di una traccia[5] in quanto non possono produrre un fondo agli eventi di segnale. In questo modo abbiamo ottenuto una reiezione del $64.6 \pm 3.2\%$ degli eventi $K^+ \to \pi^+\pi^+\pi^-$ (fig. 5.12) senza perdere nessun evento di segnale nel campione analizzato in quanto per eventi $K^+ \to \pi^+\nu\bar{\nu}$ l'algoritmo non riconosce mai due o tre tracce che convergono in un vertice (fig. 5.13).

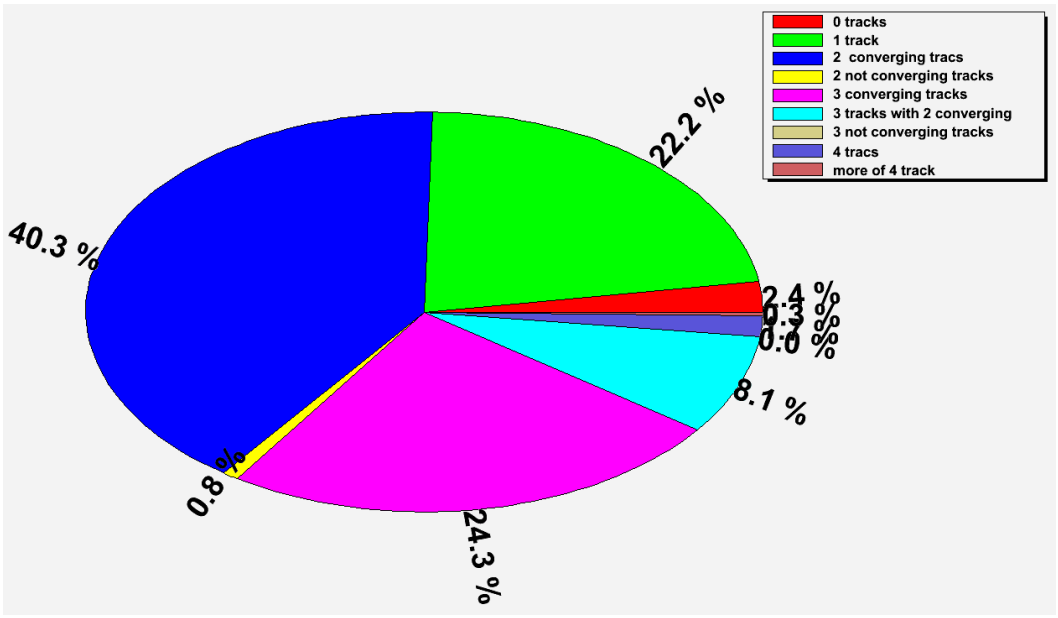


Figura 5.12: Distribuzione dei risultati dell'algoritmo per gli eventi di fondo $K^+ \to \pi^+\pi^+\pi^-$.

[5]Ricordiamo che l'algoritmo ricostruisce una traccia solo se sono presenti almeno 3 *hits* in camere differenti.

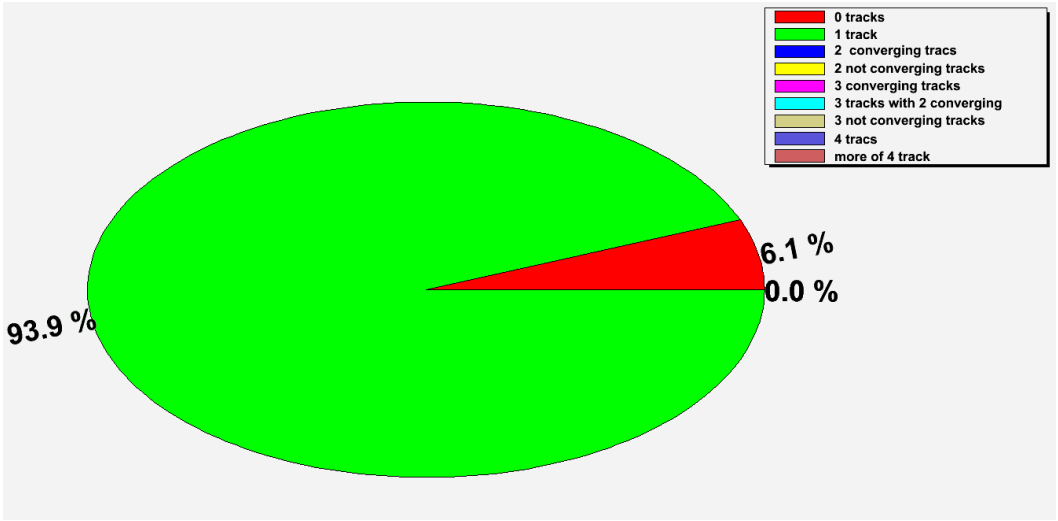


Figura 5.13: Distribuzione dei risultati dell'algoritmo per gli eventi di segnale $K^+ \to \pi^+\nu\bar{\nu}$.

5.3 Algoritmo con il Pileup

A causa del rate elevato in NA62, il *pileup* è importante e se non ne teniamo conto rischiamo di perdere una grande percentuale di eventi di segnale. Come già accennato, con una finestra temporale di lettura di 210 ns per le camere a STRAW, nel 91.7% degli eventi sono presenti fino a 11 *hits* di *pileup* nelle singole camere (fig. 5.1) così che, senza utilizzare tagli temporali per riconoscere il *pileup* e ridurre la finestra temporale, non è possibile utilizzare un *trigger* per rigettare il fondo $K^+ \to \pi^+\pi^+\pi^-$ in modo sicuro, cioè senza perdere una cospicua parte di segnale, pari al $12.7 \pm 1.2\%$ (blu e viola in fig. 5.14).

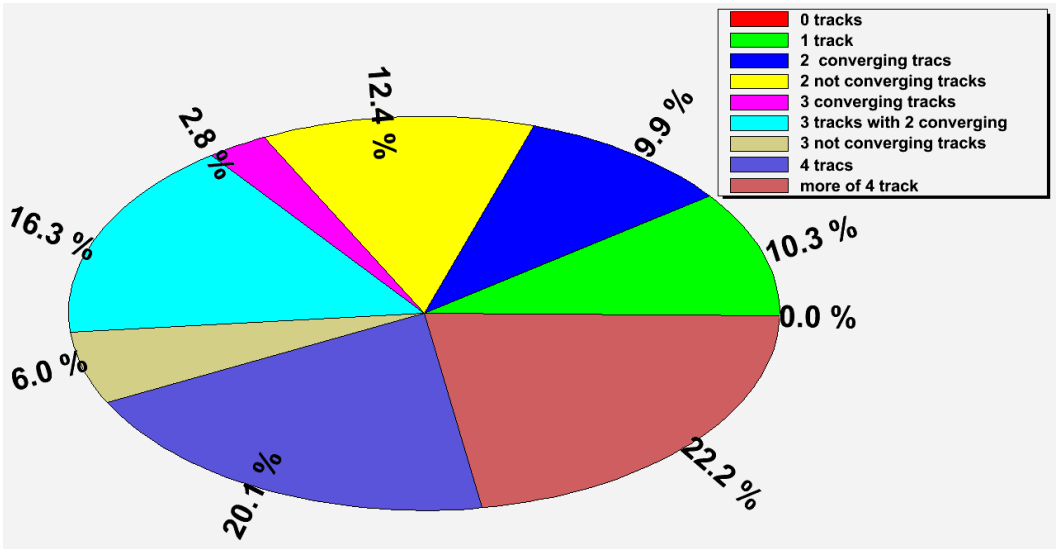


Figura 5.14: Distribuzione dei risultati dell'algoritmo per gli eventi di segnale $K^+ \to \pi^+\nu\bar{\nu}$ con *pileup*, senza tagli temporali.

La simulazione Monte Carlo ufficiale della collaborazione non consente ancora di generare eventi con il *pileup*, così ho studiato i suoi effetti utilizzando Flyo, ovvero una simulazione Monte Carlo semplificata ma molto più veloce sviluppata a Pisa [25]. Il tempo necessario

per generare 1000 eventi con il Monte Carlo ufficiale è circa 2 ore mentre con Flyo 1000000 di eventi si generano in circa 40 minuti.

Per studiare gli effetti del *pileup* sulla reiezione del fondo e sulle perdite di segnale ho generato 25000 eventi $K^+ \to \pi^+\pi^+\pi^-$ nella regione compresa tra il bersaglio di produzione e 215 m dallo stesso[6] e 6000 eventi[7] $K^+ \to \pi^+\nu\bar{\nu}$ nella regione fiduciale, compresa tra 105 m e 165 m dal bersaglio di produzione. In entrambi i casi si considera il *pileup* proveniente da tutti i principali decadimenti del K^+ (tab. 5.1) e generato nella regione compresa tra il bersaglio di produzione e 215 m dallo stesso, in quanto una qualsiasi particella generata prima delle camere può sovrapporsi all'evento di *trigger*, considerando anche che circa il 63% dei decadimenti genera muoni che sono particolarmente penetranti.

decadimenti	branching ratio (%)
$K^+ \to \mu^+\nu$	63.54
$K^+ \to \pi^+\pi^0$	20.66
$K^+ \to \pi^+\pi^+\pi^-$	5.59
$K^+ \to \pi^0 e^+\nu$	5.07
$K^+ \to \pi^0\mu^+\nu$	3.353
$K^+ \to \pi^+\pi^0\pi^0$	1.761

Tabella 5.1: *Branching ratio* usati in Flyo per la produzione del pileup

Si considerano regioni di decadimento diverse per il segnale e gli eventi $K^+ \to \pi^+\pi^+\pi^-$ perché le perdite di segnale devono essere misurate soltanto nella regione che verrà considerata nell'analisi dei dati; invece durante la presa dati possono concorrere al fondo nelle camere tutte le particelle che colpiscono il rivelatore, anche parzialmente, e quindi gli eventi di fondo devono essere generati in questa regione per misurare la reiezione.

5.3.1 I Cluster

Per ridurre le perdite di segnale abbiamo effettuato dei tagli temporali, iniziando dalla creazione dei *cluster* che è diventata così più elaborata e precisa.

Abbiamo considerato, per ogni vista e ogni camera, ognuno dei 4 piani singolarmente, raggruppando ogni *hit* nel primo e nel secondo piano con al massimo uno del terzo o del quarto; questo perché gli angoli delle tracce nel piano *y-z* sono così piccoli[8] che per ogni *straw* del primo e del secondo piano ce ne è solo una del terzo o del quarto piano che può combaciare. Se consideriamo l'esempio nella figura (5.15) possiamo notare che una traccia passante per la *straw* 2 (secondo piano) può colpire soltanto le *straw* 3 e 4 (rispettivamente del terzo e quarto piano); allo stesso modo la *straw* 1 (primo piano) può essere colpita dalla traccia che colpisce solamente le *straw* 3 e 5 (terzo e quarto piano). In questo modo ogni *cluster* è composto da due *hits* e risultano improbabili sia la sua creazione con *hits*

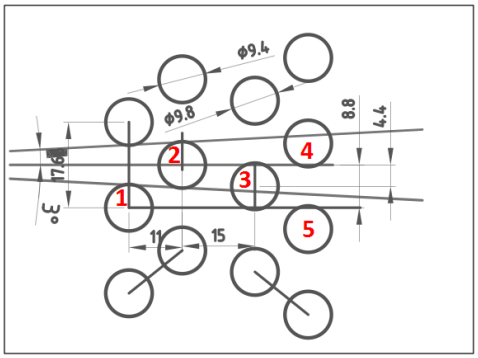


Figura 5.15: Distribuzione delle *straw* di una vista colpite dalla stessa traccia.

[6] Coordinata longitudinale a cui è posizionata l'ultima camera dello spettrometro.
[7] Numero superiore di un fattore 60 al numero di eventi di segnale che la collaborazione mira a raccogliere.
[8] Le tracce di un qualsiasi decadimento formano un angolo inferiore a 20 mrad rispetto all'asse *z* nel piano *y-z*, in particolare le tracce dovute ai decadimenti $K^+ \to \pi^+\pi^+\pi^-$ formano angoli inferiori a 8 mrad rispetto all'asse *z*.

provenienti da diverse tracce, sia lo spostamento della sua coordinata per l'inclusione di raggi delta come avveniva col precedente metodo di formazione dei *cluster* di *hits*.

Usando due *hits* insieme ai loro *leading time* è possibile assegnare come coordinata fine al *cluster* la posizione in cui è passata la traccia, all'interno della *straw* del primo o del secondo piano, con una risoluzione inferiore a 0.2 mm, compatibile con quella misurata al *test beam* sui prototipi di *straw* [35].

Per ridurre la larghezza temporale della distribuzione degli *hit* e riconoscere gli *hit* ed i *cluster* di *pileup* sono stati introdotti, durante la creazione dei *cluster*, alcuni tagli sul *trailing time* e sul *leading time* degli *hits*. Il rivelatore ci fornisce dei tempi quantizzati (*timestamp*) e dobbiamo eseguire la differenza tra i medesimi ed il *timestamp* del *trigger* fornito da un altro rivelatore, che molto probabilmente sarà il RICH. I *timestamp* del *leading time* e del *trailing time* sono soggetti a un errore di misura che si propaga nei tagli insieme all'errore dovuto alla risoluzione del tempo di *trigger* (dovuto al RICH). Il *timestamp* del *trigger* arriverà al computer con la GPU in formato digitale, a seconda del numero di bit che gli vengono riservati può avere una valore di 3, 6, 12 o 25 ns per il bit meno significativo (errore di quantizzazione o $\Delta_{trigger}$). Considerando uniforme la distribuzione dei *timestamp* del *trigger* così troncato, abbiamo usato per la propagazione dell'errore tale $\Delta_{trigger}/\sqrt{12}$.

I primi due tagli temporali riguardano gli *hits* singolarmente, con una verifica sul loro *trailing time* (eq. 5.1) e *leading time* (eq. 5.3):

$$|trailing\ time - 160\ \text{ns}| < 3\ \sigma_{trailing\ time} \tag{5.1}$$

$$\sigma_{trailing\ time} = \sqrt{\sigma^2_{trailing\ timestamp} + (\Delta_{trigger}/\sqrt{12})^2} \tag{5.2}$$

$$0\ \text{ns} - 2\ \sigma_{leading\ time} < leading\ time < 160\ \text{ns} + 2\ \sigma_{leading\ time} \tag{5.3}$$

$$\sigma_{leading\ time} = \sqrt{\sigma^2_{leading\ timestamp} + (\Delta_{trigger}/\sqrt{12})^2} \tag{5.4}$$

Nelle disequazioni 5.1 e 5.3 abbiamo usato 160 ns perché è il valore del *leading time*, non considerando gli errori, nel caso in cui la traccia passi sul bordo della *straw* (fig. 2.24); per lo stesso motivo 160 ns sono anche al valor medio del *trailing time*.

Questi tagli si basano sul fatto che i *trailing time* e i *leading time* degli *hits* di *pileup* sono spostati temporalmente di una quantità pari alla differenza tra il tempo in cui è passata la traccia di *pileup* ed il tempo dell'evento di *trigger*.

Come ulteriore taglio più efficace, abbiamo utilizzato una proprietà dei *cluster*: la somma delle distanze tra le posizioni dei due *hits* all'interno delle *straw* del *cluster* ed il centro delle stesse è pari alla distanza lungo l'asse ortogonale alla traccia tra i centri delle due *straw*, cioè 4.4 mm (fig. 5.16). Questa proprietà è valida solo per tracce perpendicolari all'asse y, quindi l'angolo delle tracce, anche se molto piccolo, aggiunge in pratica un valore, che sarebbe calcolabile dall'angolo della traccia che però è ignoto, alla misura di questa quantità su cui effettuare un taglio. Considerando che l'angolo massimo di una traccia in un evento $K^+ \to \pi^+\pi^+\pi^-$ è circa 8 mrad, e che la distanza massima in z tra due piani (cioè tra il primo e il quarto) è di 37 mm allora l'errore massimo introdotto è inferiore a 0.3 mm, mentre nel caso del segnale $K^+ \to \pi^+\nu\bar{\nu}$ l'angolo massimo delle tracce è circa 10 mrad comportando un errore massimo inferiore a 0.5 mm.

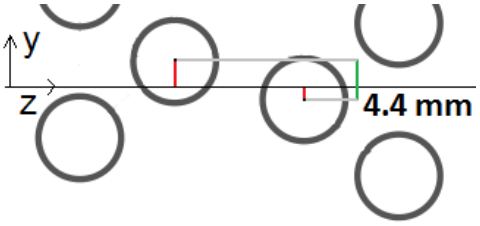


Figura 5.16: Somma delle distanze degli *hits* di una traccia dai centri delle *straw*.

Nella simulazione Monte Carlo abbiamo assunto una approssimazione lineare della relazione che collega la distanza dell'*hit* dal centro della *straw* ed il *leading time*: così è possibile

sommare direttamente i *leading time* dei due *hits* al posto delle distanze dal centro delle *straw*. Tale relazione non sarà esattamente lineare ma una volta nota la relazione spazio-tempo sarà possibile utilizzarla per migliorare l'algoritmo. Per un evento in tempo con il *trigger* proveniente dal RICH la somma dei *leading time* deve essere quindi uguale al *leading time* di una traccia che passa a 4.4 mm dal centro della *straw* (150 ns) a meno dei vari errori introdotti dalla misura dei *leading timestamp*, dall'angolo delle tracce e dalla risoluzione del *trigger* (eq. 5.5).

$$|leading\ time\ (1) + leading\ time\ (2) - 150\ \text{ns}| < 3\ \sigma_{somma\ leading\ time} \tag{5.5}$$

$$\sigma_{somma\ leading\ time} = \sqrt{2 * \sigma^2_{leading\ timestamp} + 2 * (\Delta_{trigger}/\sqrt{12})^2 + (\sigma_{angolo})^2} \tag{5.6}$$

Per i *cluster* costruiti con *hits* di *pileup* la somma dei *leading time* è spostata temporalmente di una quantità pari al doppio della differenza tra il tempo dell'evento e il tempo del *trigger*. Approssimando le distribuzioni con delle Gaussiane abbiamo ricavato le risoluzioni temporali da utilizzare nella propagazione dell'errore sulla somma dei *leading time* degli *hits* dei *cluster*. Per l'errore dovuto all'angolo della traccia ci siamo basati sugli eventi $K^+ \to \pi^+\pi^+\pi^-$ che vogliamo rigettare ricavando $\sigma_{angolo} = 3.5$ ns dalla simulazione Monte Carlo, mentre per gli errori di misura del *leading time*, del *trailing time* e la risoluzione del trigger abbiamo testato più ipotesi in quanto i valori non sono ancora stati finalizzati.

Per ultimo abbiamo aggiunto un taglio sul numero di *cluster* ricostruiti: se vengono ricostruiti più di 3 *cluster* in una vista questi non vengono considerati nella ricostruzione delle tracce perché probabilmente saranno presenti tra di essi *cluster* di *pileup*; inoltre con più *cluster* sarebbe più lenta l'esecuzione dell'algoritmo e ci sarebbe una maggiore la probabilità di creazione di false tracce.

Abbiamo verificato come varia la larghezza della distribuzione temporale degli *hit* in seguito ai tagli utilizzando diversi valori per gli errori appena discussi. Errori utilizzati:

$$\sigma_{leading\ timestamp} \longrightarrow 2,\ 4,\ 6,\ 8\ \text{ns}$$

$$\sigma_{trailing\ timestamp} \longrightarrow 5,\ 10,\ 15,\ 20\ \text{ns}$$

$$\Delta_{trigger} \longrightarrow 1,\ 3,\ 6,\ 12,\ 25\ \text{ns}$$

Variando l'errore sul *trailing timestamp* non abbiamo riscontrato sostanziali variazioni nella larghezza della distribuzione, mentre variando la risoluzione del *trigger* si osserva un allargamento significativo per il valore massimo $\Delta_{trigger} = 25$ ns (fig. 5.17).

I cambiamenti più significati sono portati dalla variazione dell'errore di misura del *leading timestamp* (fig. 5.18) dovuti alla maggiore importanza del taglio sulla somma dei *leading time* per riconoscere i *cluster* di *pileup*.

Abbiamo quindi misurato la percentuale di eventi che non presentano *cluster* di *pileup* in seguito ai tagli che abbiamo applicato, al variare dei valori degli errori di misura del *leading timestap* (fig. 5.19), del *trailing timestamp* (fig. 5.20) e della risoluzione temporale del *trigger* (fig. 5.21). Coerentemente con gli istogrammi delle distribuzioni temporali i grafici mostrano una maggiore importanza al livello della creazione dei *cluster* dell'errore sul *leading timestamp*, la cui variazione comporta una differenza pari a circa il 12% sull'individuazione di *cluster* di *pileup*, mentre la variazione dell'errore sul *trailing timestamp* comporta una differenza solo del 3% concentrata quasi interamente nel passaggio tra $\sigma = 10$ ns e $\sigma = 5$ ns. Analogamente riguardo alla risoluzione temporale del *trigger* si ha una variazione significativa ($\sim 3\%$) solo nel passaggio tra 12 ns e 25 ns, quindi a questo livello tutti i valori di risoluzione tra 1 ns e 12 ns sembrano equivalenti.

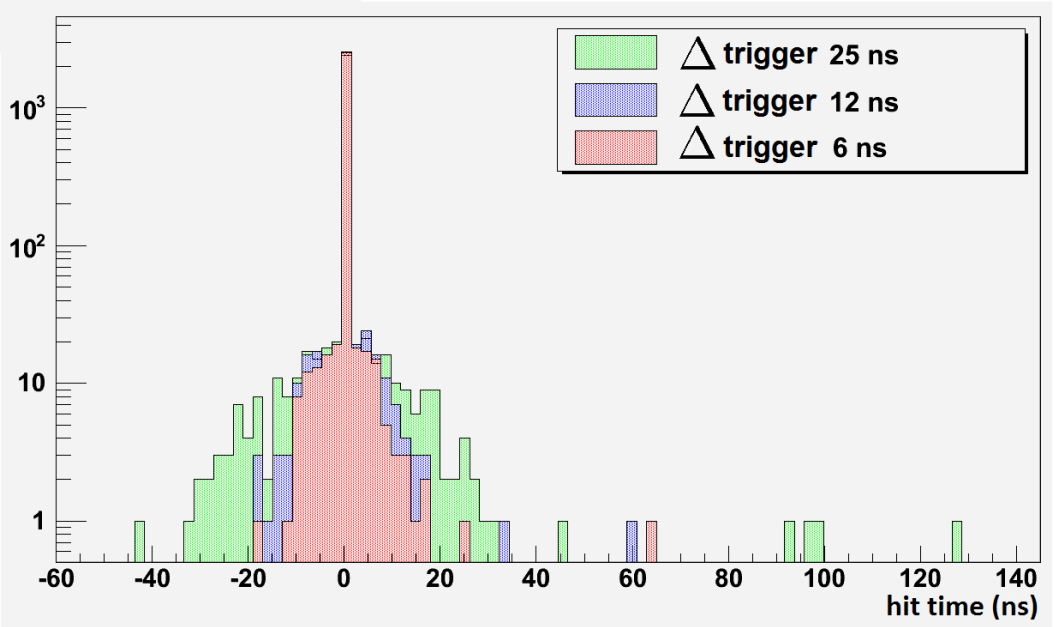


Figura 5.17: Distribuzione del tempo degli *hits* al variare della risoluzione temporale del trigger: il picco a 0 ns sono gli *hits* in tempo con l'evento di *trigger*.

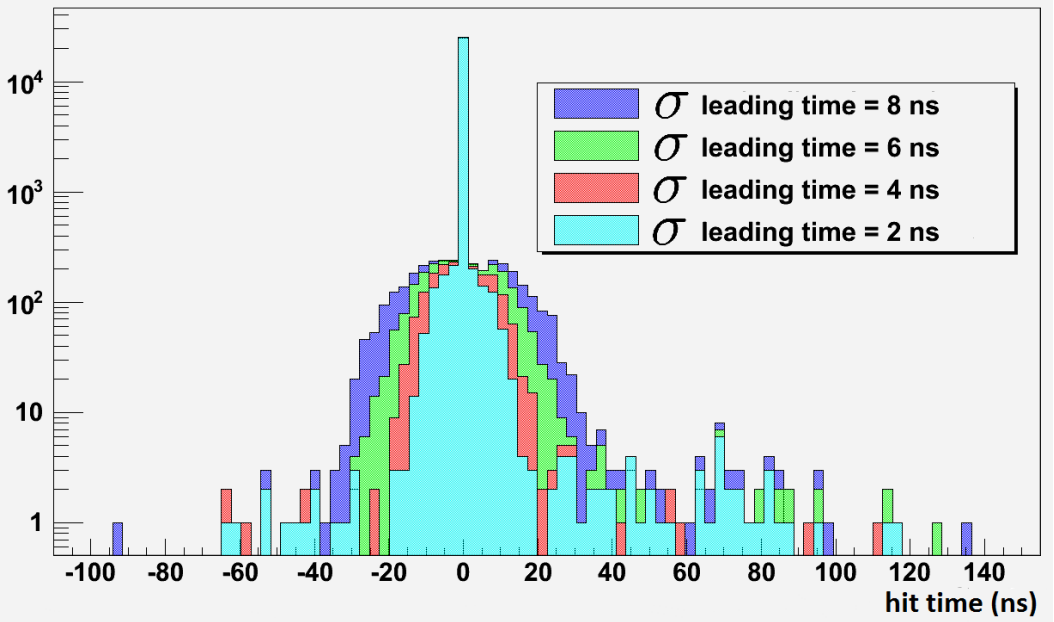


Figura 5.18: Distribuzione del tempo degli *hits* al variare dell'errore sul *leading timestamp*; il picco a 0 ns sono gli *hits* in tempo con l'evento di *trigger*.

Figura 5.19: Percentuale di eventi che non presentano *cluster* di *pileup* al variare dell'errore sul *leading timestamp*.

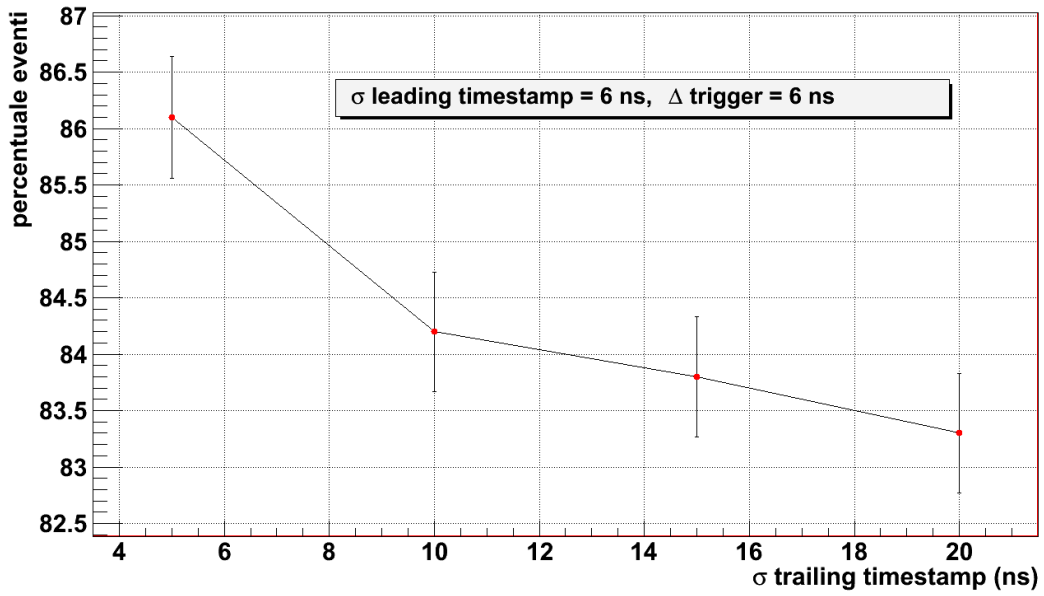


Figura 5.20: Percentuale di eventi che non presentano *cluster* di *pileup* al variare dell'errore sul *trailing timestamp*.

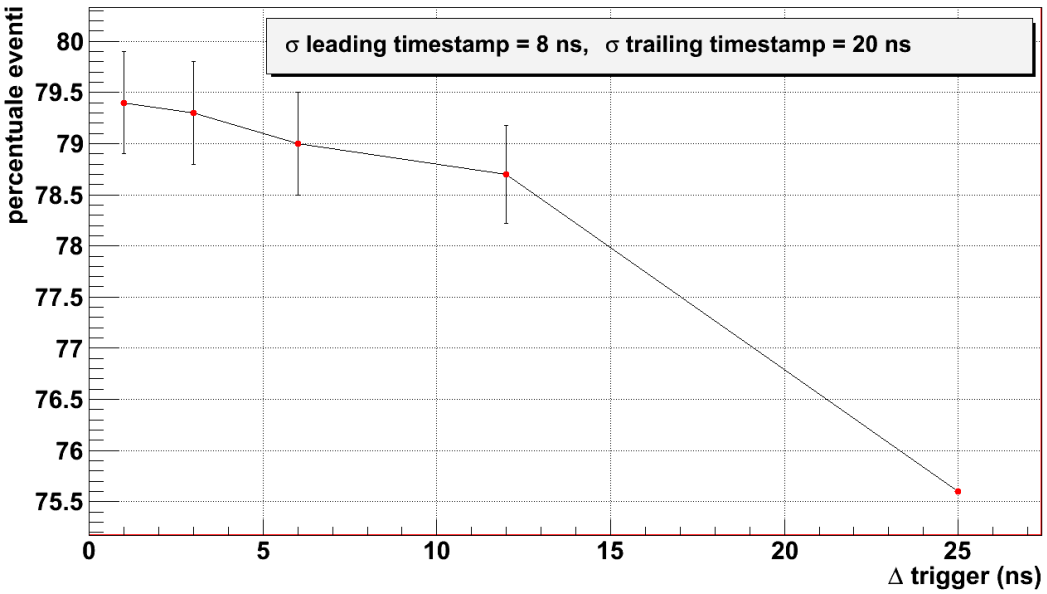


Figura 5.21: Percentuale di eventi che non presentano *cluster* di *pileup* al variare di $\Delta_{trigger}$.

5.3.2 Ricostruzione Delle Tracce E Dei Vertici Di Decadimento

Abbiamo utilizzato la stessa ricostruzione delle tracce utilizzata nello studio del caso senza *pileup*, tranne che per il cambiamento di qualche parametro come il valore massimo della distanza tra il terzo punto e la retta ottenuta dai primi due, che è passato da 3.4 a 7.5 mm a causa del differente raggruppamento in cluster degli *hit*, mentre nell'individuazione dei vertici è stato aggiunto un taglio sulla media dei *trailing time* di tutti gli *hits* che compongono i *cluster* delle tracce convergenti in un vertice. Questo taglio risulta molto efficace in quanto per ogni vertice ricostruito sono stati usati 12 o 18 *hits*, a seconda che ci siano 2 o 3 tracce[9], quindi l'errore sulla media del *trailing time* del vertice è uguale a $\sigma_{trailing\ time}/\sqrt{12}$ o $\sigma_{trailing\ time}/\sqrt{18}$, riuscendo così a ridurre ulteriormente la finestra temporale e a riconoscere alcune tracce di *pileup*.

Per una migliore parallelizzazione dell'algoritmo abbiamo accorpato questo taglio temporale insieme alle richieste relative al vertice nel calcolo di un unico parametro. Unendo in un unico calcolo più tagli si riduce il numero di condizionali utilizzati nell'algoritmo per decidere se rigettare l'evento, in questo modo si rende più veloce la sua esecuzione in quanto nelle GPU la presenza di blocchi condizionali può portare a grandi rallentamenti[10].

Per una maggiore efficienza dell'algoritmo abbiamo utilizzato due parametri leggermente diversi per i casi con 2 tracce e quelli con 3 tracce; in entrambi i casi il parametro consiste nella somma dei quadrati di tre valori normalizzati con delle costanti scelte, in modo da avere un compromesso tra la massimizzazione della reiezione degli eventi $K^+ \to \pi^+\pi^+\pi^-$ e la minimizzazione delle perdite di segnale $K^+ \to \pi^+\nu\bar{\nu}$. Nel caso a due tracce il parametro, chiamato r, (eq. 5.7) è composto dalla distanza in metri tra il punto di intersezione delle due tracce con l'asse z (Δz), dalla distanza in mm del vertice di intersezione delle due tracce dall'asse z ($y_{vertice}$) e dalla media dei *trailing time* ($\overline{trailing\ time\ vertice}$) di tutti gli *hits* che compongono i *cluster* delle due tracce. I valori delle normalizzazioni (K_z, K_y e $K_{trailing\ time}$) insieme ai metodi utilizzati per la loro determinazione saranno illustrati in

[9] Ogni traccia è formata da 3 *cluster* che a sua volta contiene 2 *hits*.

[10] Si hanno possibili divergenze e tutti i *work-item* eseguono le istruzioni all'interno dei blocchi anche se la condizione è falsa.

seguito; nei primi test con l'utilizzo di questo parametro sono stati scelti dei valori intuitivi e non ottimizzati per queste costanti.

$$r = \left(\frac{\Delta z}{K_z}\right)^2 + \left(\frac{y_{vertice}}{K_y}\right)^2 + \left(\frac{\overline{trailing\ time\ vertice}}{K_{trailing\ time}}\right)^2 \tag{5.7}$$

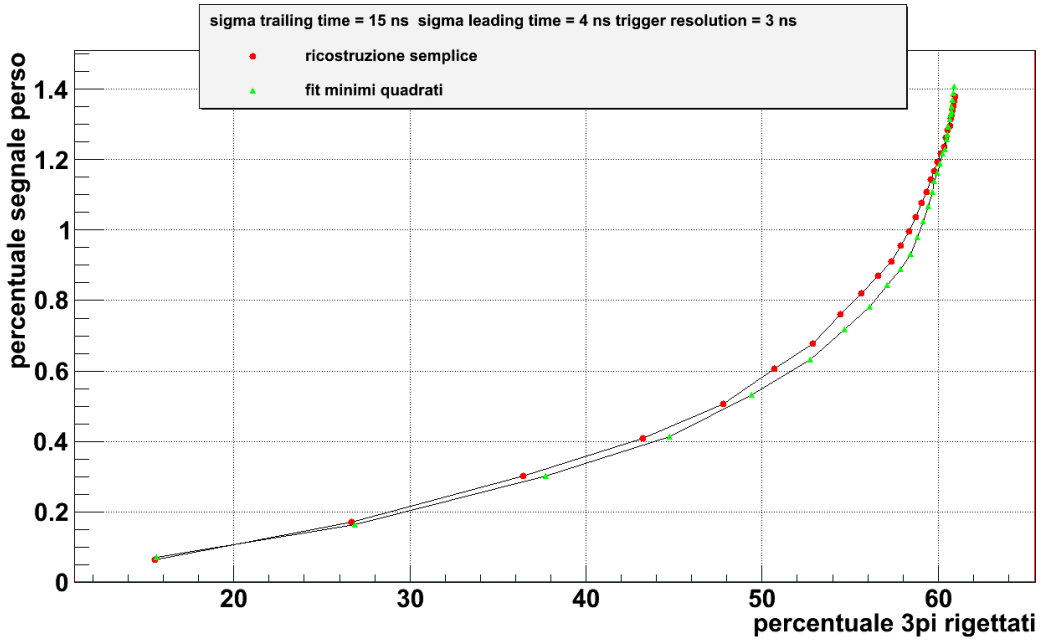


Figura 5.22: Confronto tra la percentuale di segnale perso in funzione della percentuale di eventi $K^+ \to \pi^+\pi^+\pi^-$ rigettati, sia utilizzando la ricostruzione semplice che utilizzando i minimi quadrati; il grafico è stato ottenuto variando il taglio sul parametro r.

Oltre al taglio su questo parametro "r", l'evento, per essere considerato un $K^+ \to \pi^+\pi^+\pi^-$ e quindi rigettato, deve soddisfare le seguenti condizioni:

1. le due tracce devono avere pendenze di segno opposto.
2. il valore assoluto delle pendenze delle tracce deve essere inferiore a 0.006.
3. la z del vertice deve avere un valore compreso tra 93 m e 183 m.

La prima condizione non è sempre soddisfatta dagli eventi che vogliamo rigettare in quanto non è presente una traccia del decadimento e quindi può capitare siano state ricostruite le due tracce che hanno la pendenza con lo stesso segno, comunque l'utilizzo di questo taglio permette di limitare notevolmente le perdite di segnale a fronte di una minima riduzione della reiezione. Il taglio sulla pendenza è dovuto all'angolo massimo con l'asse z nel piano y-z delle tracce degli eventi $K^+ \to \pi^+\pi^+\pi^-$ (8 mrad) ed il suo valore è stato scelto come compromesso tra la reiezione del fondo e la perdita del segnale; allo stesso modo sono stati scelti i valori riguardanti la posizione z del vertice ricostruito.

Nel caso a tre tracce abbiamo dovuto usare tre parametri (eq. 5.8) perché le tracce, se considerate due a due, formano tre intersezioni. Ognuno dei tre parametri prende in considerazione una delle intersezioni utilizzando, oltre alla distanza in mm del vertice di intersezione dall'asse z ($y_{vertice}$) come nel caso a due tracce, la distanza lungo y in mm tra il punto dell'intersezione e la restante traccia (Δy). La terza componente dei parametri

consiste nella media dei *trailing time* ($\overline{trailing\ time\ vertice}$) degli *hits* che compongono i *cluster* di tutte e tre le tracce, questa componente è uguale per tutti e tre i parametri.

$$r_{1,2,3} = \left(\frac{(y_{vertice})_{1,2,3}}{(K_y)_{1,2,3}}\right)^2 + \left(\frac{(\Delta y)_{1,2,3}}{(K_{\Delta y})_{1,2,3}}\right)^2 + \left(\frac{\overline{trailing\ time\ vertice}}{K_{trailing\ time}}\right)^2 \tag{5.8}$$

In questo caso per essere considerato un evento $K^+ \to \pi^+\pi^+\pi^-$, almeno uno dei tre parametri ($r_1,\ r_2,\ r_3$) deve risultare inferiore ad un valore stabilito r_{max} e due delle tracce devono avere pendenza di segno opposto. Non sono stati utilizzati ulteriori tagli, come quelli usati nel caso a due tracce, in quanto non sono risultati vantaggiosi nella reiezione del fondo $K^+ \to \pi^+\pi^+\pi^-$ e nella limitazione delle perdite di segnale.

Per migliorare l'efficienza di reiezione dell'algoritmo e ridurre le perdite, abbiamo provato ad implementare il metodo dei minimi quadrati in modo da ottimizzare la ricostruzione delle tracce e conseguentemente migliorare la risoluzione dei vertici di decadimento. Abbiamo confrontato le due ricostruzioni attraverso un grafico bidimensionale (fig. 5.22) che mette in relazione la percentuale di eventi $K^+ \to \pi^+\pi^+\pi^-$ riconosciuti e rigettati con la percentuale di segnale perso al variare del taglio sul parametro r. Dal grafico si nota un leggero miglioramento utilizzando il metodo dei minimi quadrati che però non è sufficiente per giustificare il probabile rallentamento del tempo di esecuzione del algoritmo con il suo uso sulla GPU dovuto all'aumento dei calcoli e al maggiore utilizzo di memoria della scheda. Per questo motivo abbiamo scelto di continuare ad usare la ricostruzione semplice, preferendo la rapidità di esecuzione dell'algoritmo alla sua precisione.

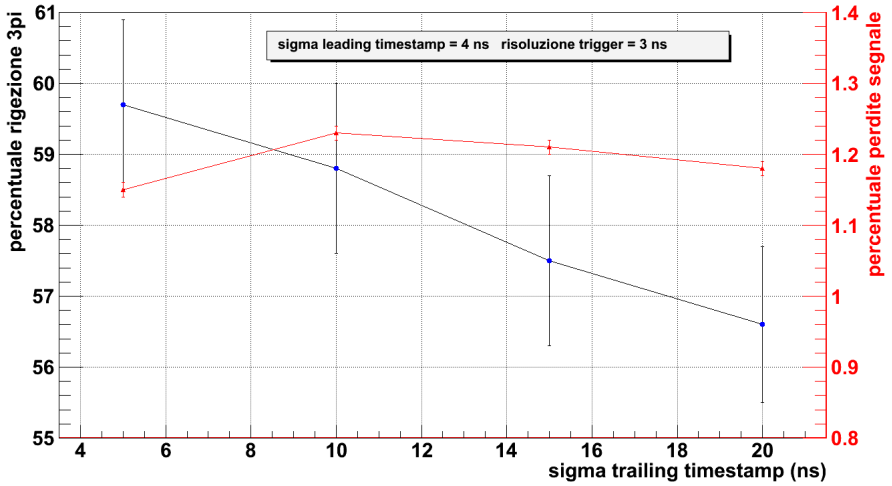


Figura 5.23: Percentuale di eventi $K^+ \to \pi^+\pi^+\pi^-$ rigettati (in nero) e di segnale perso (in rosso) al variare dell'errore sul *trailing timestamp*.

Successivamente abbiamo verificato il comportamento delle percentuali di reiezione del fondo $K^+ \to \pi^+\pi^+\pi^-$ e di perdita di segnale al variare degli errori sul *trailing timestamp* (fig. 5.23), sul *leading timestap* (fig. 5.24) e al variare di $\Delta_{trigger}$ (fig. 5.25). Dal grafico (fig. 5.23) risulta che la variazione dell'errore sul *trailing timestamp* influisce più sulla percentuale di reiezione del fondo rispetto alle perdite di segnale, questo effetto però potrebbe essere dovuto ad una mancata ottimizzazione delle costanti del parametro r relative alla media dei *trailing time* degli *hits* del vertice. L'errore sul *leading time* risulta sempre il parametro più importante, infatti la sua variazione (tra 2 ns a 8 ns) comporta un cambiamento di $\sim 6\%$ nella reizione del fondo e di $\sim 1\%$ per le perdite di segnale (fig. 5.24). L'errore di quantizzazione temporale del *trigger* risulta importante per le perdite di segnale mentre

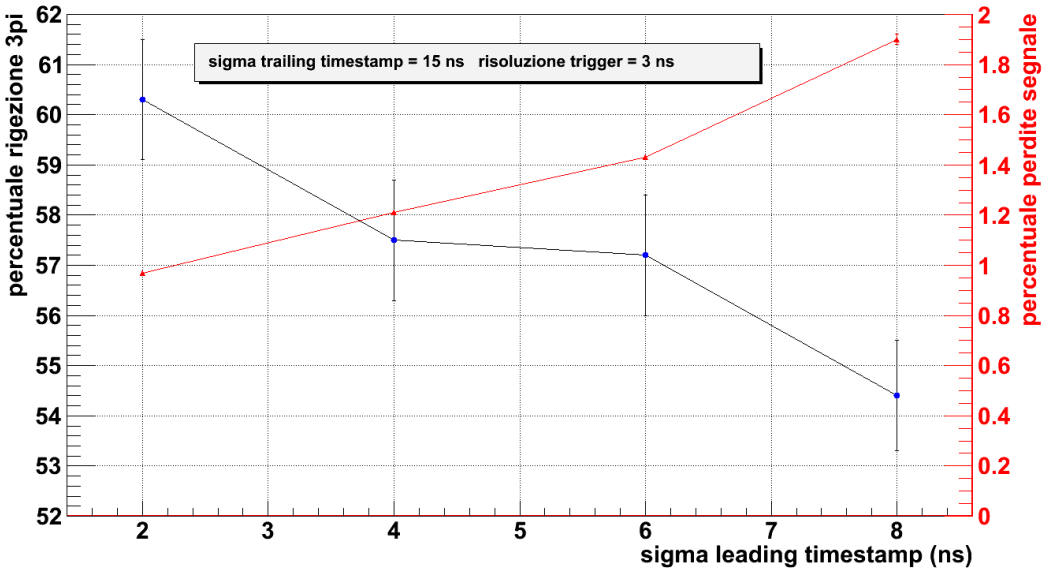


Figura 5.24: Percentuale di eventi $K^+ \to \pi^+\pi^+\pi^-$ rigettati (in nero) e di segnale perso (in rosso) al variare dell'errore sul *leading timestamp*.

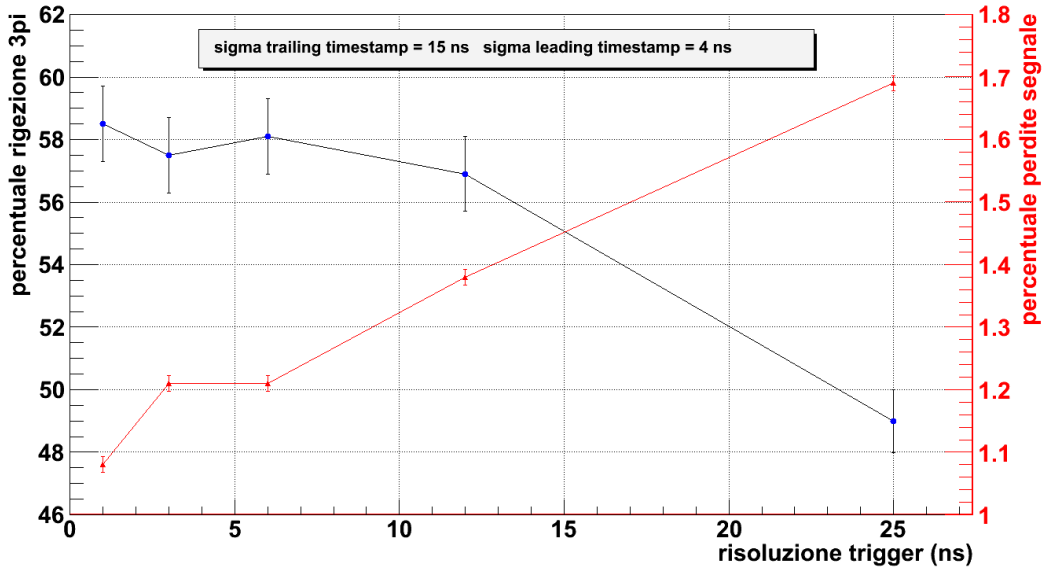


Figura 5.25: Percentuale di eventi $K^+ \to \pi^+\pi^+\pi^-$ rigettati (in nero) e di segnale perso (in rosso) al variare di $\Delta_{trigger}$.

influisce in maniera sostanziale ($\sim 8\%$) sulla reiezione del fondo solo nel passaggio tra 12 ns e 25 ns.

A questo punto ci siamo focalizzati sull'ottimizzazione del taglio sul parametro r e delle costanti K_z, K_y e $K_{trailing\ time}$ (eq. 5.7). Per questa ottimizzazione abbiamo scelto di utilizzare i seguenti errori temporali in quanto sembrano quelli più verosimili dagli ultimi test.

$$\sigma_{leading\ timestamp} = 4 \text{ ns} \quad \sigma_{trailing\ timestamp} = 15 \text{ ns} \quad \Delta_{trigger} = 3 \text{ ns}$$

Visto che dobbiamo scegliere la migliore combinazione per le quattro costanti inerenti al taglio sul parametro r (K_z, K_y, $K_{trailing\ time}$ e r_{max}, che è il massimo valore di r per cui si considerano riconosciuti gli eventi come $K^+ \to \pi^+\pi^+\pi^-$), abbiamo suddiviso l'ottimizzazione in due parti. Nella prima parte abbiamo fissato le costanti K_z e r_{max} mentre con un doppio ciclo abbiamo sviluppato due grafici tridimensionali per la percentuale di reiezione del fondo (fig. 5.26) e per la percentuale di perdite di segnale (fig. 5.27) in funzione della variazione dei valori di K_y, costante inerente la distanza del vertice dall'asse z, e $K_{trailing\ time}$, riguardante la media dei *trailing time* di tutti gli *hits* delle due tracce. Per scegliere i valori di K_y e

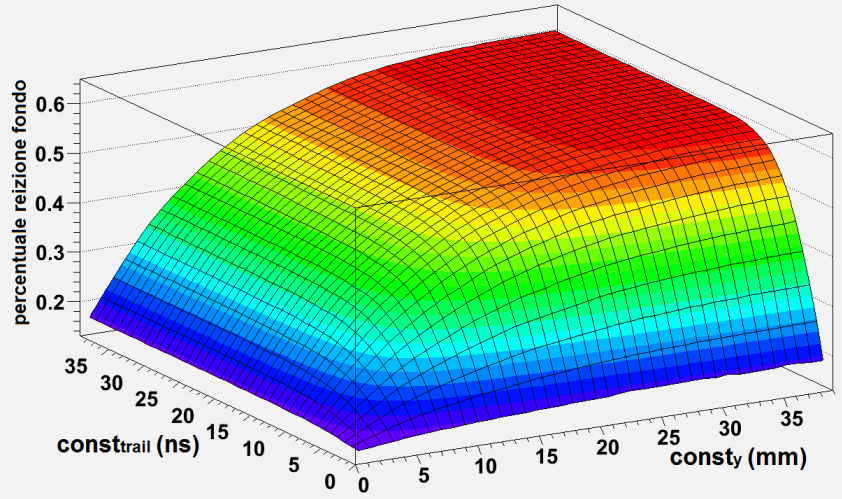


Figura 5.26: Percentuale di eventi di fondo $K^+ \to \pi^+\pi^+\pi^-$ rigettati al variare di K_y e $K_{trailing\ time}$.

$K_{trailing\ time}$ abbiamo preso la proiezione del piano orizzontale delle due costanti dei due grafici tridimensionali e li abbiamo sovrapposti in modo da trovare un punto del piano che permetta un buon compromesso tra reiezione degli eventi di fondo $K^+ \to \pi^+\pi^+\pi^-$ e le perdite di segnale $K^+ \to \pi^+\nu\bar{\nu}$. Abbiamo deciso di scegliere il punto che permette la maggiore reiezione del fondo con perdite del 1.2% di segnale, quindi abbiamo tracciato sulla proiezione orizzontale della reiezione del fondo la curva indicante il luogo dei punti con perdita di segnale pari al 1.2% (fig. 5.28), estratta dalla proiezione orizzontale del grafico tridimensionale della perdita di segnale. In questo modo abbiamo identificato il punto con $K_y = 22$ mm e $K_{trailing\ time} = 14$ ns che permette una reiezione del fondo pari al 56.5% e un perdita del 1.2% del segnale. Dopo aver scelto i valori delle prime due costanti, allo stesso modo abbiamo sviluppato due grafici tridimensionali per la percentuale di reiezione del fondo (fig. 5.29) e per la percentuale di perdite di segnale (fig. 5.30) in funzione del variare di K_z, inerente la distanza tra le intersezioni delle due rette con l'asse z, e r_{max}. Con lo stesso procedimento usato per la precedente coppia di costanti abbiamo preso la proiezione orizzontale dei due grafici in modo da confrontali e da trovare un compromesso tra reiezione del fondo e perdite di segnale. Quindi abbiamo tracciato sulla proiezione orizzontale della reiezione del fondo la curva indicante il luogo dei punti con perdita di segnale pari al 1.1%

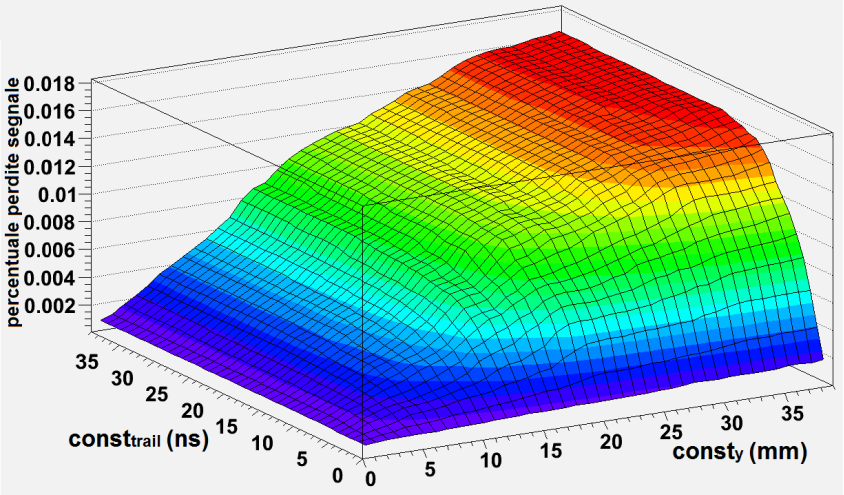


Figura 5.27: Percentuale di eventi di segnale $K^+ \to \pi^+\nu\bar{\nu}$ persi al variare di K_y e $K_{trailing\ time}$.

(fig. 5.31) e abbiamo identificato il punto con $K_z = 25$ m e $r_{max} = 1.7$ che permette una reiezione del fondo pari al 56.7% e un perdita del 1.1% del segnale.

In modo simile è stato affrontato il caso a tre tracce, per il quale abbiamo scelto le seguenti costanti.

$$(K_y)_{1,2,3} = 53 \text{ mm}, \quad (K_{trailing\ time})_{1,2,3} = 11 \text{ ns}$$
$$(K_{\Delta y})_{1,2,3} = 40\text{mm}, \quad (r_{max})_{1,2,3} = 1.1$$

Completate le ottimizzazioni abbiamo confrontato i risultati dell'algoritmo con quelli ottenuti precedentemente alle ottimizzazioni attraverso un grafico bidimensionale (fig. 5.32) che mette in relazione la percentuale di eventi $K^+ \to \pi^+\pi^+\pi^-$ riconosciuti e rigettati con la percentuale di segnale perso al variare del taglio sul parametro r. Il grafico (fig. 5.32) mostra che a seconda del punto scelto abbiamo ottenuto con le ottimizzazioni una riduzione della percentuale di eventi persi fino a $\sim 0.2\%$.

Successivamente abbiamo provato a eseguire una variazione sui tagli temporali applicati nella creazione dei cluster in modo da verificare se ci sono cambiamenti nella reiezione del fondo e nelle perdite di segnale. Quindi abbiamo cambiato il fattore moltiplicativo (numero di deviazioni standard) applicato a $\sigma_{somma\ leading\ time}$ e a $\sigma_{trailing\ time}$ nei rispettivi tagli (descritti nel paragrafo precedente eq. 5.1 e 5.5) provando oltre a 3, che è stato utilizzato fino ad ora, il fattore 2 ed uno intermedio uguale a 2.67. Dal grafico (fig. 5.33) per reiezioni superiori al 54%, utilizzando il fattore moltiplicativo 2.67 per i tagli relativi al *trailing time* e alla somma dei *leading time* si ottiene un miglioramento sulle perdite di segnale di $\sim 0.2\%$ rispetto al taglio utilizzato fino ad ora. L'utilizzo di 2 come fattore moltiplicativo risulta conveniente solo se si accettano reiezioni del fondo $K^+ \to \pi^+\pi^+\pi^-$ inferiori al 52%.

Infine, dopo queste ottimizzazioni, abbiamo deciso di generare un maggior numero di eventi, 100000 di fondo $K^+ \to \pi^+\pi^+\pi^-$ e 50000 di segnale[11] $K^+ \to \pi^+\nu\bar{\nu}$, in modo da avere maggiore precisione nelle percentuali di perdita di segnale in funzione delle percentuali di reiezione del fondo al variare del taglio sul parametro r. Questo grafico (fig. 5.34) sarà utile per decidere, dopo l'analisi dei primi dati reali, il giusto compromesso tra reiezione del fondo e perdite di segnale. Utilizzando ad esempio $r_{max} = 1.7$ si ottiene una reiezione del fondo $K^+ \to \pi^+\pi^+\pi^-$ pari a $57.6 \pm 0.2\%$ e si perde $1.09 \pm 0.01\%$ del segnale $K^+ \to \pi^+\nu\bar{\nu}$.

[11] Il numero di eventi è superiore di un fattore 500 al numero di eventi di segnale che la collaborazione mira a raccogliere.

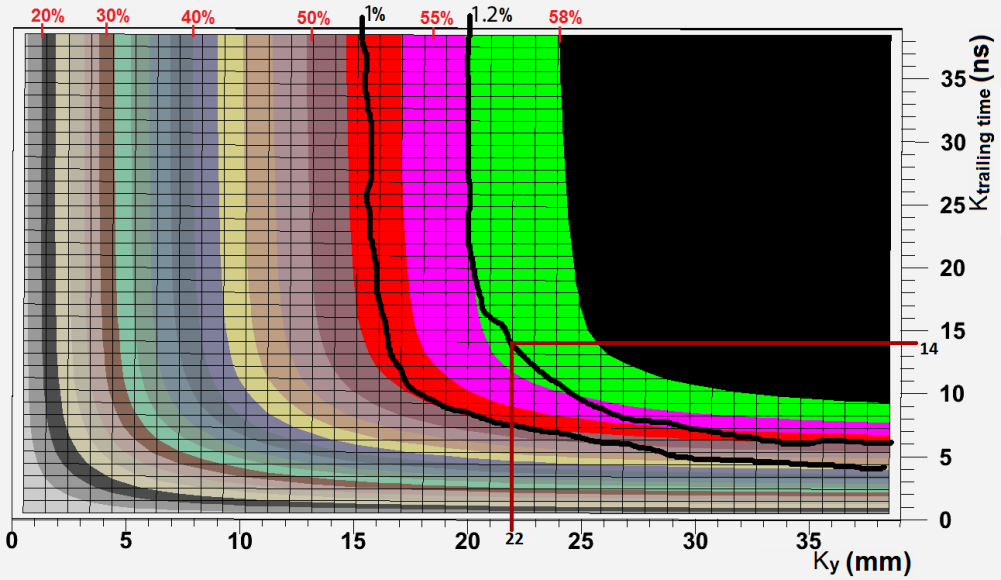


Figura 5.28: Proiezione sul piano orizzontale del grafico tridimensionale della percentuale di eventi di fondo rigettati al variare di K_y e $K_{trailing\ time}$. In rosso sono indicate le percentuali di fondo rigettate mentre sono sovrapposte due curve in nero, ricavare dalla proiezione del grafico tridimensionale della percentuale di segnale perso, che mostrano gli andamenti con perdita di segnale del 1% e del 1.2%. Con due proiezioni rosse parallele agli assi è mostrato anche il punto nel piano bidimensionale delle due costanti che è stato scelto.

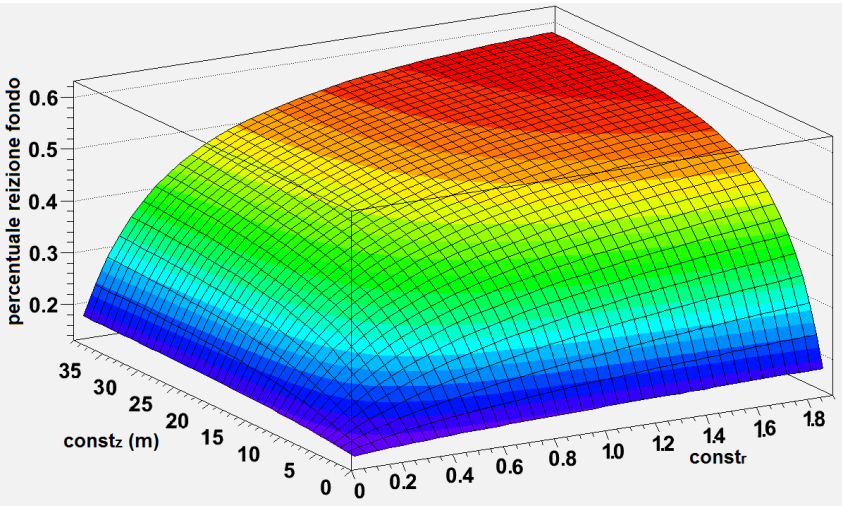


Figura 5.29: Percentuale di eventi di fondo $K^+ \to \pi^+\pi^+\pi^-$ rigettati al variare di K_z e r_{max}.

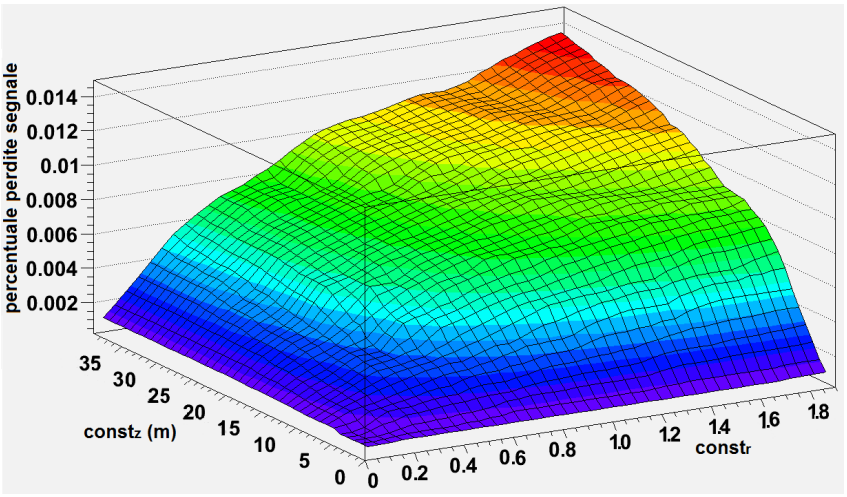


Figura 5.30: Percentuale di eventi di segnale $K^+ \to \pi^+ \nu\bar{\nu}$ persi al variare di K_z e r_{max}.

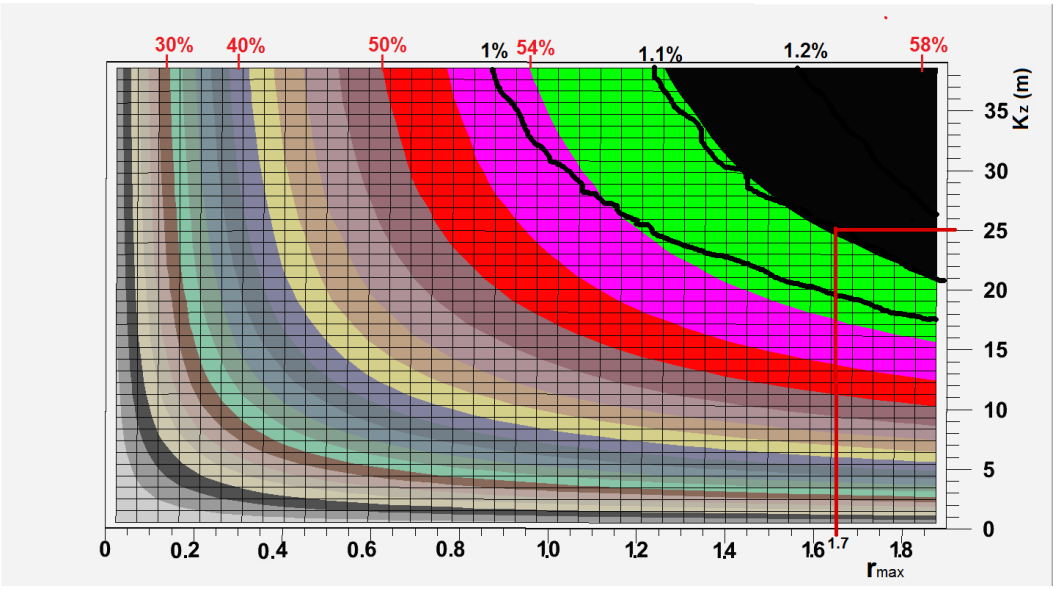


Figura 5.31: Proiezione sul piano orizzontale del grafico tridimensionale della percentuale di eventi di fondo rigettati al variare di K_z e r_{max}. In rosso sono indicate le percentuali di fondo rigettate mentre sono sovrapposte tre curve in nero, ricavare dalla proiezione del grafico tridimensionale della percentuale di segnale perso, che mostrano gli andamenti con perdita di segnale del 1%, 1.1% e 1.2%. Con due proiezioni rosse parallele agli assi è mostrato anche il punto nel piano bidimensionale delle due costanti che è stato scelto.

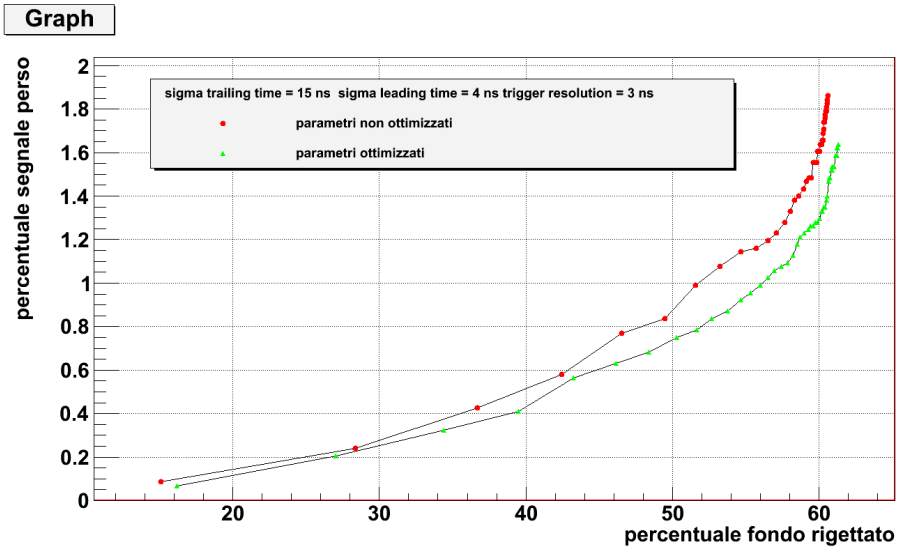


Figura 5.32: Confronto tra la percentuale di segnale perso in funzione della percentuale di eventi $K^+ \to \pi^+\pi^+\pi^-$ rigettati prima e dopo le ottimizzazioni; il grafico è stato ottenuto variando il taglio sul parametro r.

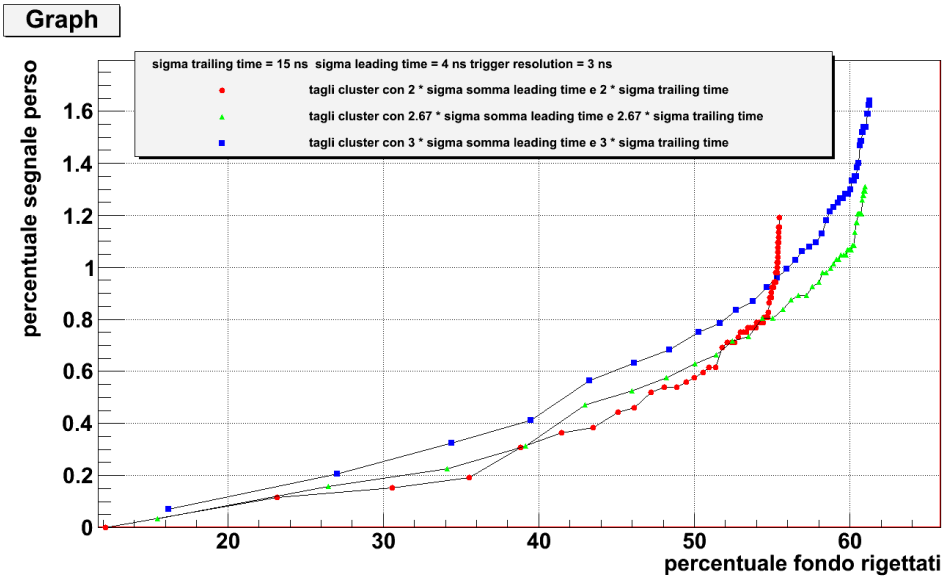


Figura 5.33: Confronto tra la percentuale di segnale perso in funzione della percentuale di eventi $K^+ \to \pi^+\pi^+\pi^-$ rigettati variando i tagli sulla somma dei *leading time* e sul *trailing time*; il grafico è stato ottenuto variando il taglio sul parametro r.

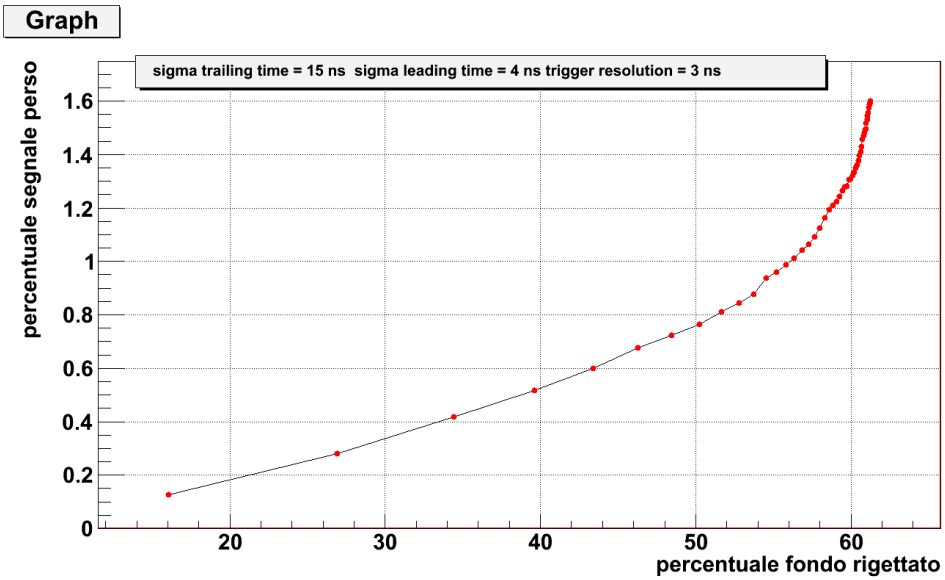


Figura 5.34: Percentuale di segnale perso in funzione della percentuale di eventi $K^+ \to \pi^+\pi^+\pi^-$ rigettati dopo le ottimizzazioni dell'algoritmo; il grafico è stato ottenuto variando il taglio sul parametro r.

5.4 Implementazione ed Ottimizzazione dell'Algoritmo sulla GPU

Ultimato l'algoritmo lo ho implementato sulla GPU, dedicandomi prima di tutto alla sua parallelizzazione. La parallelizzazione di questo algoritmo non è risultata semplice in quanto la sua struttura globale è sequenziale e composta da quattro parti da eseguire una dopo l'altra: il raggruppamento degli hit in *cluster* con l'applicazione dei tagli temporali, il completamento dei cluster della vista y con quelli provenienti dalle viste u e v,la ricostruzione delle tracce e l'identificazione dei vertici di decadimento. In ognuna di queste parti c'è un largo uso di condizionali (*if/else*), che rallentano notevolmente l'esecuzione dell'algoritmo con la verifica delle condizioni causali perché tutti i *work-item* eseguono sia le istruzioni dentro l'*if* che quelle dentro l'*else*, e cicli (*for*) annidati l'uno dentro l'altro che rallentano notevolmente l'esecuzione in quanto il tempo di esecuzione di un intero *wavefront* è determinato dal tempo di esecuzione del *work-item* che esegue il *loop* il maggior numero di volte.

Un altra limitazione dell'algoritmo rispetto al tempo di esecuzione è dovuta all'utilizzo della *local memory* che può ridurre il numero di *work-group* attivi (sezione 4.5): facendo processare ogni evento da un singolo *compute unit* nella sua *local memory* si devono allocare tre vettori float3 (vettori tridimensionali composti da float), con 140 allocazioni ciascuno, per contenere gli *hit* delle tre viste prima del raggruppamento in *cluster* con le informazioni del numero del tubo, del piano e della camera dell'*hit*, e i valori del *leading timestamp* e del *trailing timestamp* dell'*hit*. Visto che solo l'allocazione di questi tre vettori comporta l'utilizzo di 5.04 kB di *local memory*, abbiamo deciso di riutilizzare questi vettori per i successivi passaggi riuscendo ad occupare solo 6.03 kB complessivi di *local memory*, permettendo così di mantenere attivi fino a 5 *work-group* contemporaneamente.

Nella prima parte dell'algoritmo, che si occupa di raggruppare in *cluster* gli *hit* e di ridurre gli effetti del *pileup* tramite i tagli sui loro *trailing time* e *leading time*, abbiamo utilizzato un *work-item* per ogni vista di ogni camera in modo che 12 *work-item* contemporaneamente si occupino del raggruppamento degli *hit* e dei tagli temporali. Sarebbe possibile un'ulteriore parallelizzazione attraverso l'utilizzo di un *work-item* per ogni *hit* del primo e secondo piano di ogni camera e vista, ma questa comporterebbe l'utilizzo di 8.64 kB di *local memory* addizionali, e quindi limiterebbe ad uno il numero di *work-group* attivi.

Nella parte di completamento alla vista y abbiamo utilizzato un *work-item* per ogni cluster della vista u; ogni *work-item* esegue un ciclo *for* sugli *hit* della vista v ed uno sugli *hit* della vista y per eliminare le coppie che portano a *hit* già presenti nella vista y e per aggiungere quelli non presenti. Questa parte è risultata efficientemente parallelizzata, infatti il suo contributo al tempo di esecuzione totale dell'algoritmo è dell'ordine di soli 500 ns.

Per la parallelizzazione della terza parte dell'algoritmo, che si occupa di ricostruire le tracce utilizzando almeno 3 *cluster* provenienti da camere diverse, abbiamo utilizzato il metodo degli automi cellulari (sezione 3.6). Ogni *cellula* è formata dalla retta passante per una coppia di *cluster* appartenenti alla prima e seconda camera o alla terza e quarta camera. Abbiamo utilizzato coppie di *cluster* sia delle prime due camere che delle ultime due perché possono mancare dei *cluster* nelle prime due camere a causa di inefficienze, della accettanza geometrica e dei tagli fatti nella fase di raggruppamento. Ad ogni *cellula* è assegnato un *work-item* che si deve occupare di verificare le condizioni di sopravvivenza:

- l'intersezione della retta con l'asse z deve avvenire all'interno della regione di decadimento precedente alle camere (tra 93 m e 178 m dal bersaglio di produzione) in modo da minimizzare il numero di tracce finte ricostruite;
- la retta deve avere il valore assoluto della pendenza inferiore a 0.006 (vedi paragrafo 5.3.2);
- deve essere presente un terzo *cluster* in una camera non utilizzata nella formazione della cellula con distanza tra questo cluster e la retta inferiore a 20.5 mm.

Se le condizioni non vengono soddisfatte la *cellula* smette di esistere senza ricostruire la traccia passante per la coppia di *cluster*. La parallelizzazione di questa fase comporta una perdita di efficienza in quanto precedentemente si usavano dei cicli *for* che provavano tutte le combinazioni di tre punti in tutte le camere fino a trovare una, se c'era, che soddisfacesse le richieste. In seguito alla parallelizzazione vengono provate soltanto un terzo delle possibili combinazioni, riducendo il numero di tracce che rispettano i requisiti. Per rimediare a questa inefficienza abbiamo deciso di aumentare a 20.5 mm la distanza massima accettata tra il terzo cluster e la retta, anche se questo comporta un aumento delle perdite di segnale. La reiezione del fondo $K^+ \to \pi^+\pi^+\pi^-$ si è ridotta a $55.42 \pm 0.25\%$ e le perdite di segnale $K^+ \to \pi^+\nu\bar{\nu}$ sono salite a $1.27 \pm 0.01\%$.

Nell'ultima fase, in cui vengono ricostruiti i vertici, abbiamo utilizzato un solo *work-item* per ogni evento, che ricostruisce i vertici e verifica che vengano rispettati i tagli sui criteri r (vedi paragrafo 5.3.2).

Nella parallelizzazione dell'algoritmo e la sua scrittura in OpenCL sono stati utilizzati i metodi di ottimizzazione studiati nel precedente capitolo con l'algoritmo DOMH. In seguito alle ottimizzazioni abbiamo misurato il tempo di esecuzione dell'algoritmo per singolo evento variando il numero di eventi analizzati in un pacchetto (fig. 5.35 e tab. 5.2). Dal grafico

numero eventi	tempo esecuzione(μs)
20	166.4 ± 22.5
40	97.1 ± 1.5
100	91.6 ± 5.1
200	79.9 ± 4.5
400	74.7 ± 1.2
700	69.3 ± 0.6
1000	65.9 ± 0.7
2000	62.8 ± 0.7
3000	62.8 ± 0.2
5000	62.5 ± 0.09
10000	62.3 ± 0.06
15000	61.7 ± 0.05

Tabella 5.2: Tempi di esecuzione del kernel al variare del numero di eventi processati in un pacchetto

(fig. 5.35) si nota che sopra i 2000 eventi per pacchetto il tempo di esecuzione si stabilizza intorno a 62.5 μs per evento. Il tempo di esecuzione del *kernel* si riduce a 47.0 ± 0.8 μs per evento, processando un pacchetto di 2000 eventi, se non utilizziamo le viste u e v per completare i *cluster* della vista y; questo aumento di velocità comporta però una minore efficienza di reiezione del fondo ($52.46 \pm 0.23\%$) ma anche la riduzione delle perdite di segnale ($1.11 \pm 0.01\%$). La scelta se usare le viste u e v per completare la vista y come la scelta del valore del parametro r_{max}, che influisce sull'efficienza di reiezione del fondo e sulle perdite di segnale, verrà fatta in base all'analisi dei primi dati reali dell'esperimento.

Sarebbe possibile far eseguire la prima parte dell'algoritmo, cioè quella che si occupa di raggruppare in *cluster* gli *hit* e di effettuare i tagli temporali per ridurre gli effetti del *pileup*, alla CPU invece che alla GPU, in quanto essa è risultata essere la meno parallelizzabile, e passare alla GPU i vettori già contenenti i *cluster* per il successivo completamento dei *cluster* della vista y con quelli delle viste u e v, per la ricostruzione delle tracce e dei vertici di decadimento. Senza questa prima parte il tempo di esecuzione dell'algoritmo sulla GPU risulta notevolmente inferiore (fig. 5.36 e tab. 5.3) anche se bisognerà misurare il tempo di esecuzione della prima parte dell'algoritmo utilizzando le CPU ed implementare un efficiente trasferimento di dati tra le due. Il grafico in figura (fig. 5.36) mostra lo stesso andamento con lo "scalino" riscontrato per i grafici del tempo di esecuzione al variare del numero di

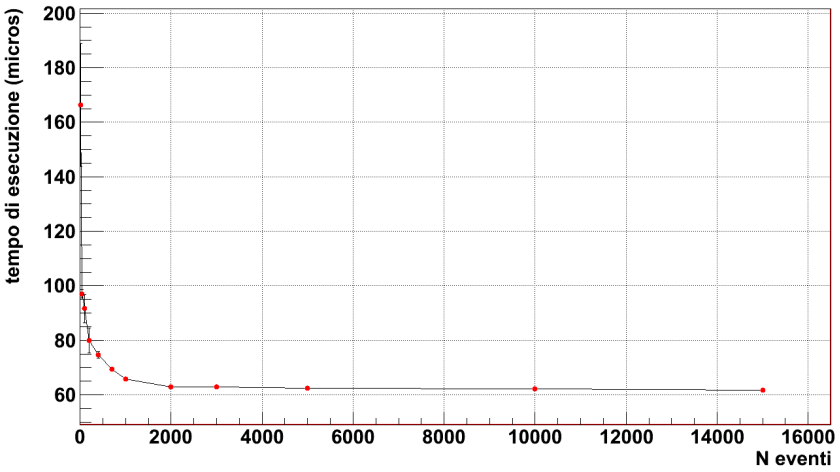


Figura 5.35: Grafico del tempo di esecuzione del *kernel*, per singolo evento, al variare del numero di eventi processati.

numero eventi	tempo esecuzione(μs)
20	14.75 ± 0.04
40	8.74 ± 0.03
100	7.61 ± 0.07
200	11.77 ± 0.25
400	11.27 ± 0.52
700	7.70 ± 1.75
1000	7.04 ± 1.11
2000	7.05 ± 0.75
3000	7.09 ± 0.50
5000	6.98 ± 0.28
10000	6.94 ± 0.11
15000	6.98 ± 0.11

Tabella 5.3: Tempi di esecuzione del kernel senza la prima parte di raggruppamento in cluster degli hit al variare del numero di eventi processati.

eventi per l'algoritmo DOMH (sezione 4.1) dimostrando che questo particolare effetto non dipende dall'algoritmo utilizzato ma dalla architettura della GPU.

Successivamente abbiamo misurato i tempi di trasferimento dei dati iniziali dalla CPU alla GPU e dei risultati dalla GPU alla CPU (fig. 5.37). I tempi di trasferimento dei dati iniziali sono di un ordine di grandezza superiori ai quelli dell'algoritmo DOMH per il RICH (sezione 4.9). Questa grande differenza è dovuta alla differenza delle dimensioni dei *buffer* di memoria da trasferire: mentre per l'algoritmo DOMH vengono trasferiti 256 byte per evento per l'algoritmo di ricostruzione dei vertici sono 5.04 kB per evento. L'utilizzo di *buffer* di memoria cosi grandi è dovuto al fatto che in ogni evento ci possono essere fino a 140 *hit* per ognuna delle tre viste.

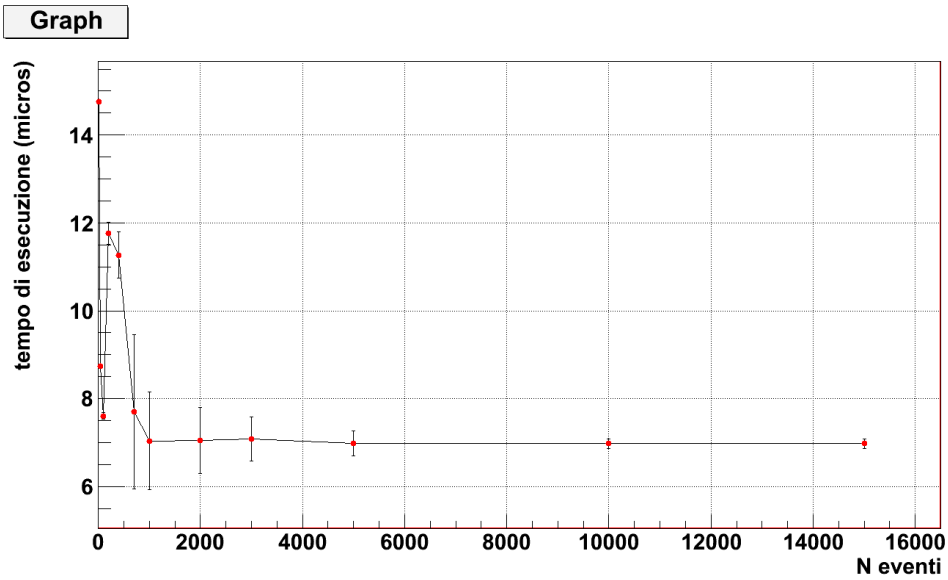


Figura 5.36: Grafico del tempo di esecuzione del *kernel* senza la prima parte di raggruppamento in cluster degli *hit*, per singolo evento, al variare del numero di eventi processati.

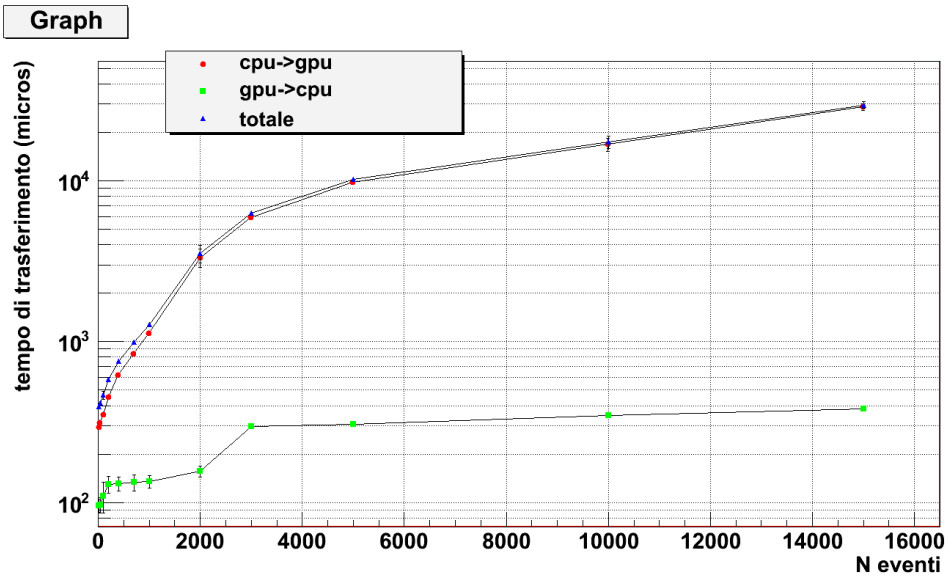


Figura 5.37: Grafico dei tempi di trasferimento dei dati iniziali dalla CPU alla GPU e dei risultati dalla GPU alla CPU in funzione del numero di eventi processati per pacchetto.

5.5 Conclusioni

Facciamo una veloce stima del tempo totale che il sistema impiega nel processare un pacchetto di 2000 eventi, sommando il tempo di esecuzione algoritmo con i tempi di trasferimento dalla CPU alla GPU e viceversa. Il tempo di esecuzione per 2000 eventi, utilizzando l'algoritmo più lento, è $(2000 * 62.8 =)\ 125600 \pm 1400$ μs, il tempo di trasferimento dei dati dalla CPU alla GPU è 3.37 ± 0.43 ms mentre quello dei risultati dalla GPU alla CPU è 0.16 ± 0.1 ms per un totale di 129.13 ± 1.47 ms. Usando un pacchetto di 2000 eventi il tempo di processamento dei dati è di un ordine di grandezza inferiore alla latenza massima del trigger di livello 1 (O(1s)). Per calcolare il tempo totale di processamento di un evento bisogna aggiungere, alla latenza ottenuta sopra, i tempi dovuti al trasferimento dei dati al PC (inferiore al ms e al collezionamento degli eventi nel buffer di memoria ($\sim$ 10 ms per 2000 eventi a 200 kHz). Si ottiene quindi un tempo totale di processamento degli eventi (O(140 ms)) di un ordine di grandezza inferiore alla massima latenza (O(1 s)), dominato dal tempo di esecuzione dell'algoritmo all'interno della GPU.

Visto che ci aspettiamo un *rate* d'ingresso di $\sim$ 200 kHz gli eventi devono essere processati in un tempo medio di ~ 5 μs; usando l'algoritmo completo con pacchetti da 2000 eventi otteniamo un tempo medio di esecuzione circa 13 volte superiore a quello richiesto. Per ovviare a questo problema potremo usare 13 o più GPU in parallelo; non usando le viste u e v basterebbero 10 GPU in parallelo dato che il tempo di esecuzione del *kernel* si riduce a 47.0 ± 0.8 μs. Altrimenti facendo eseguire la prima parte dell'algoritmo alla CPU sarebbe possibile utilizzare anche solo 2 GPU ed il numero di CPU da usare dipenderebbe dal tempo necessario alla CPU a raggruppare gli eventi in *cluster* ed ad eseguire i tagli temporali; utilizzando la CPU e la GPU contemporaneamente il tempo di esecuzione minore dei due algoritmi non conta ai fini del tempo medio di processamento degli eventi. Inoltre il numero delle GPU potrebbe ridursi con l'aumento delle prestazioni delle prossime generazioni di GPU e il prossimo AMD SDK Stream dovrebbe permettere l'utilizzo di entrambe le GPU nelle schede doppie, come la ATI Radeon HD 5970 che abbiamo usato in questa tesi, permettendo di ridurre a 6-7 il numero di schede necessarie per non perdere eventi a causa del loro *rate*.

In conclusione con questo lavoro di tesi abbiamo mostrato la possibilità e l'utilità dell'utilizzo delle GPU come *trigger* di livello 1 e di livello 0 per esperimenti di alte energie: i tempi di esecuzione e di trasferimento dati delle moderne GPU sono risultati compatibili con le richieste di sistemi di *trigger* nel caso di rate elevati come quelli dell'esperimento NA62 ed in particolare per il RICH e le camere a STRAW. Inoltre l'algoritmo studiato per le camere a STRAW permette una reiezione superiore al 50% del fondo $K^+ \to \pi^+\pi^+\pi^-$ con $\sim 1\%$ di perdite di segnale $K^+ \to \pi^+\nu\bar{\nu}$, la percentuale precisa della reizione del fonde e delle perdite di segnale può essere calibrata a seconda delle esigenze variando un parametro dei tagli (r_{max}).

Inoltre dai risultati dell'algoritmo si conclude che un miglioramento del sistema di *trigger* per fornire il *timestamp* del *trigger* con bit meno significativo di 12.5 ns (anziché 25 ns) è utile, ma non è importante spingersi oltre.

BIBLIOGRAFIA

[1] J. K. Ahn et al. "Search for the Decay $K_L^0 \to \pi^0\nu\overline{\nu}$". In: *Physical Review Letters* 100 (20 2008) (cit. a p. 7).

[2] J. Allison et al. "Geant4 developments and applications". In: *Nuclear Science* 53.11 (1 2006), pp. 270–278 (cit. a p. 76).

[3] A. Ambrosino et al. *NA62 Technical Design Document*. 2010. URL: `http://na62.web.cern.ch/na62/Documents/TechnicalDesign.html` (cit. alle pp. 1, 16, 31, 36).

[4] *AMD Accelerated Parallel Processing OpenCL Programming Guide*. URL: `http://developer.amd.com/sdks/AMDAPPSDK/assets/AMD_Accelerated_Parallel_Processing_OpenCL_Programming_Guide.pdf` (cit. alle pp. 40, 55, 57, 58).

[5] Joshua A. Anderson et al. "Rigid body constraints realized in massively-parallel molecular dynamics on graphics processing units". In: *Computer Physics Communications* 182.11 (1993), pp. 2307–2313 (cit. a p. 39).

[6] S. Anvar et al. "The beam and detector for the NA48 neutral kaon CP violation experiment at CERN". In: *Nuclear Instruments and Methods in Physics Research* A574 (3 2007), pp. 433–471 (cit. alle pp. 13, 23).

[7] A: V: Artamonov et al. "New Measurement of the $K^+ \to \pi^+\nu\bar{\nu}$ branching ratio". In: *Physical Review Letters* 101, 191802 (2008) (cit. a p. 7).

[8] H. W. Atherton et al. *Precise Measurements of Particle Production by 400 GeV/c Protons on Beryllium Targets*. Yellow Report CERN 80-07. CERN, 1980. URL: `http://sba.web.cern.ch/sba/Documentations/docs/atherton.pdf` (cit. a p. 16).

[9] I. Augustin et al. "The drift chamber electronics and readout for the NA48 experiment at the CERN SPS". In: *Nuclear Instruments and Methods in Physics Research* A403 (1998), pp. 472–480 (cit. a p. 30).

[10] M. Battaglia et al. *The CKM Matrix and the Unitarity Triangle*. workshop. CERN, 2003 (cit. a p. 4).

[11] Massimo Bernaschi et al. "A flexible high-performance Lattice Boltzmann GPU code for the simulations of fluid flows in complex geometries". In: *Concurrency and Computation: Practice and Experience* 22.1 (2010), pp. 1–14 (cit. a p. 39).

[12] C. Bovet et al. *The CEDAR Counters for Particle Identification in the SPS Secondary Beams*. Report CERN 82-13. CERN, 1982 (cit. a p. 17).

[13] D. Bryman e R: Tschirhart. *The Project X Kaon Physics Research Program*. 2010. URL: `https://indico.fnal.gov/getFile.py/access?resId=1&materialId=0&confId=3579` (cit. a p. 12).

[14] Gerhard Buchalla e Andrzej J. Buras. "$K \to \pi\nu\bar{\nu}$ and high precision determinations of the CKM matrix". In: *Physical Review* D 54 (11 1996), pp. 6782–6789 (cit. a p. 5).

[15] Gerhard Buchalla e Andrzej J. Buras. "QCD corrections to rare K- and B-decays for arbitrary top quark mass". In: *Nuclear Physics* B400 (1993), pp. 225–239 (cit. a p. 2).

[16] Gerhard Buchalla e Andrzej J. Buras. "The rare decays $K^+ \to \pi^+\nu\bar{\nu}$ and $K_L \to \mu^+\mu^-$ beyond leading logarithms". In: *Nuclear Physics* B412 (1994), pp. 106–142 (cit. a p. 2).

[17] Gerhard Buchalla e Andrzej J. Buras. "The rare decays $K \to \pi\nu\bar{\nu}$, $B \to X\nu\bar{\nu}$ and B –> l+l-: An update". In: *Nuclear Physics* B548 (1999), pp. 309–327 (cit. alle pp. 1, 2, 5).

[18] Gerhard Buchalla e Andrzej J. Buras. "Waiting for precise measurements of $K^+ \to \pi^+\nu\bar{\nu}$ and $K_L \to \pi^0\nu\bar{\nu}$". In: *Reviews of Modern Physics* 80 (3 2008), pp. 965–1007 (cit. a p. 2).

[19] Andrzej J. Buras et al. "Charm quark contribution to $K^+ \to \pi^+\nu\bar{\nu}$ at the Next-to-Next-to-Leading Order". In: *Journal of High Energy Physics* 11, 002 (2006) (cit. a p. 2).

[20] Andrzej J. Buras et al. "Rare Decay $K^+ \to \pi^+\nu\bar{\nu}$ at the Next-to-Next-to-Leading Order in QCD". In: *Physical Review Letters* 95, 261805 (2005) (cit. a p. 2).

[21] NA62 Collaboration. *NA62/P-326*. Status Report CERN-SPSC-2007-035. CERN, 2007. URL: `http://na62.web.cern.ch/na62/Documents/spsc-2007-035.pdf` (cit. alle pp. 16, 17).

[22] NA62 Collaboration. "Proposal to measure the Rare Decay $K^+ \to \pi^+\nu\bar{\nu}$ at the CERN SPS". In: *CERN-SPSC-2005-013* (2005) (cit. alle pp. 8, 11).

[23] G. Collazuol et al. "Fast online triggering in high-energy physics experiments using GPUs". sottoposto per la pubblicazione in Nuclear Instruments and Methods in Physics Research A. 2011 (cit. alle pp. 48, 56).

[24] M. De Beer et al. "High resolution drift chambers for the NA48 experiment at CERN". In: *Nuclear Instruments and Methods in Physics Research* A367 (1995), pp. 88–91 (cit. a p. 30).

[25] *Flyo62 Monte Carlo*. URL: `http://pierazzini.unipi.it/giuseppe/FlyoHtml/flyostore/downloads.html` (cit. a p. 84).

[26] M. Gardner. "Mathematical games: The fantastic combinations of John Conway's new solitaire game Life". In: *Scientific American* 223 (1970), pp. 120–123 (cit. a p. 49).

[27] S. L. Glashow, J. Iliopoulos e L. Maiani. "Weak Interactions with Lepton-Hadron Symmetry". In: *Physical Review* D2 (7 1970), pp. 1285–1292 (cit. a p. 3).

[28] A. Glazov et al. "Filtering tracks in discrete detectors using a cellular automaton". In: *Nuclear Instruments and Methods in Physics Research* A329 (1993), pp. 262–268 (cit. a p. 49).

[29] G.Y. Lim. "Kaon experiments at J-PARC". In: *Nuclear Physics* B - Proceedings Supplements - 210-211 (2011), pp. 210–215 (cit. a p. 12).

[30] Mikolaj Misiak e Jörg Urban. "QCD corrections to FCNC decays mediated by Z-penguins and W-boxes". In: *Physics Letters* B451 (1999), pp. 161–169 (cit. alle pp. 1, 2).

[31] *NA62 Monte Carlo*. URL: `http://sergiant.web.cern.ch/sergiant/NA62FW/html/` (cit. a p. 76).

[32] K. Nakamura et al. "Review of Particle Physics". In: *Journal of Physics* G37 (2010) (cit. a p. 6).

[33] *NVIDIA CUDA C Programming Guide*. URL: `http://developer.download.nvidia.com/compute/DevZone/docs/html/C/doc/CUDA_C_Programming_Guide.pdf` (cit. a p. 55).

[34] M. T. Schiller. "Standalone track reconstruction for the Outer Tracker of the LHCb experiment using a cellular automaton". Diploma thesis. University of Heidelberg, 2007 (cit. a p. 49).

[35] Alexandra Scott. *A Preliminary Look at the 2010 Straw*. Note NA62-10-03. CERN, 2010. URL: `http://na62.web.cern.ch/na62/Documents/NotesDoc/NA62-10-03.pdf` (cit. a p. 86).

[36] *The OpenCL Specification*. URL: `http://www.khronos.org/registry/cl/specs/opencl-1.1.pdf` (cit. a p. 55).

[37] *vedi ad esempio*. URL: `http://en.wikipedia.org/wiki/GPGPU#Applications` (cit. a p. 39).

[38] Lincoln Wolfenstein. "Parametrization of the Kobayashi-Maskawa Matrix". In: *Physical Review Letters* 51 (21 1983), pp. 1945–1947 (cit. a p. 3).