STEFANO DI LORENZO

Consortium GARR | THE ITALIAN EDUCATION & RESEARCH NETWORK

GPU implementation of the Levenberg-Marquardt least square minimization algorithm

GIORNATA DI INCONTRO
BORSE DI STUDIO GARR
"ORIO CARLINI"
ROMA

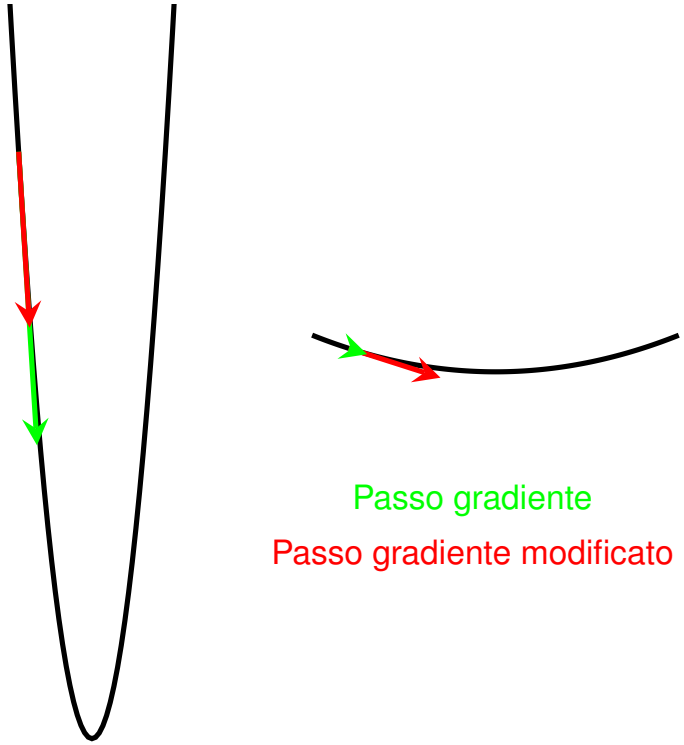
L'Aquila 25/11/2020

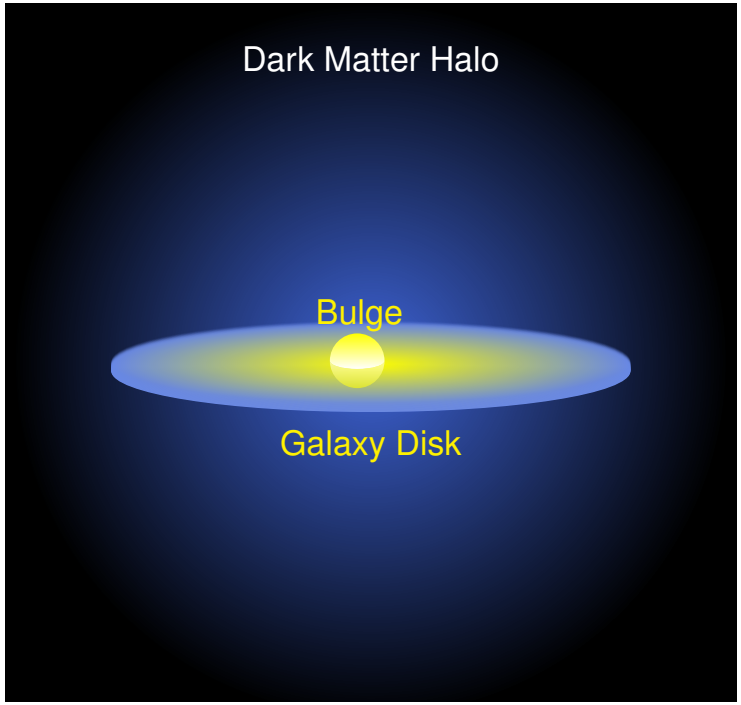
Borsisti Day 2020

L'algoritmo di Levenberg-Marquardt

È un metodo per trovare di minimizzazione basato sulla discesa lungo il gradiente e con un a parametro di scala

L'idea di base è quella di variare il passo del gradiente per velocizzare la convergenza.



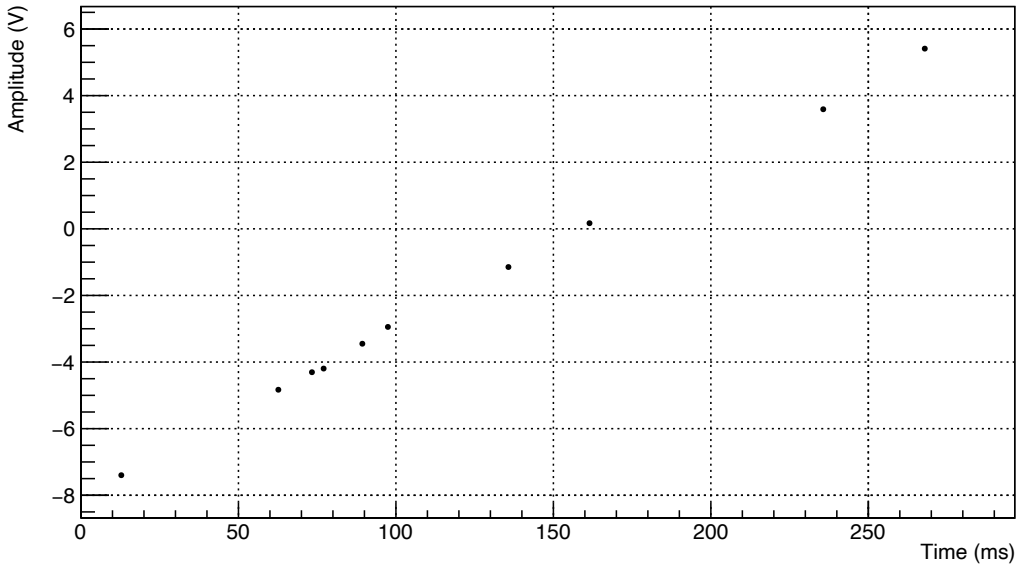


1) A cosa serve un algoritmo di minimizzazione nella ricerca di Materia Oscura ?

2) Perché è una buona idea implementarlo su GPU?

1) A cosa serve un algoritmo di minimizzazione nella ricerca di Materia Oscura ?

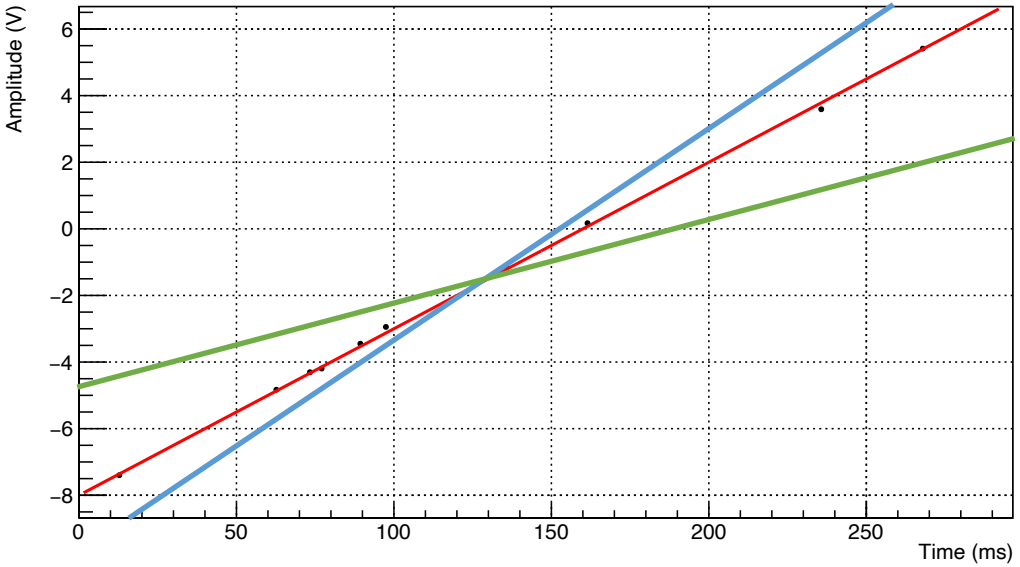
Uno dei tipici problemi in fisica è quello di trovare i parametri migliori per descrivere dei dati



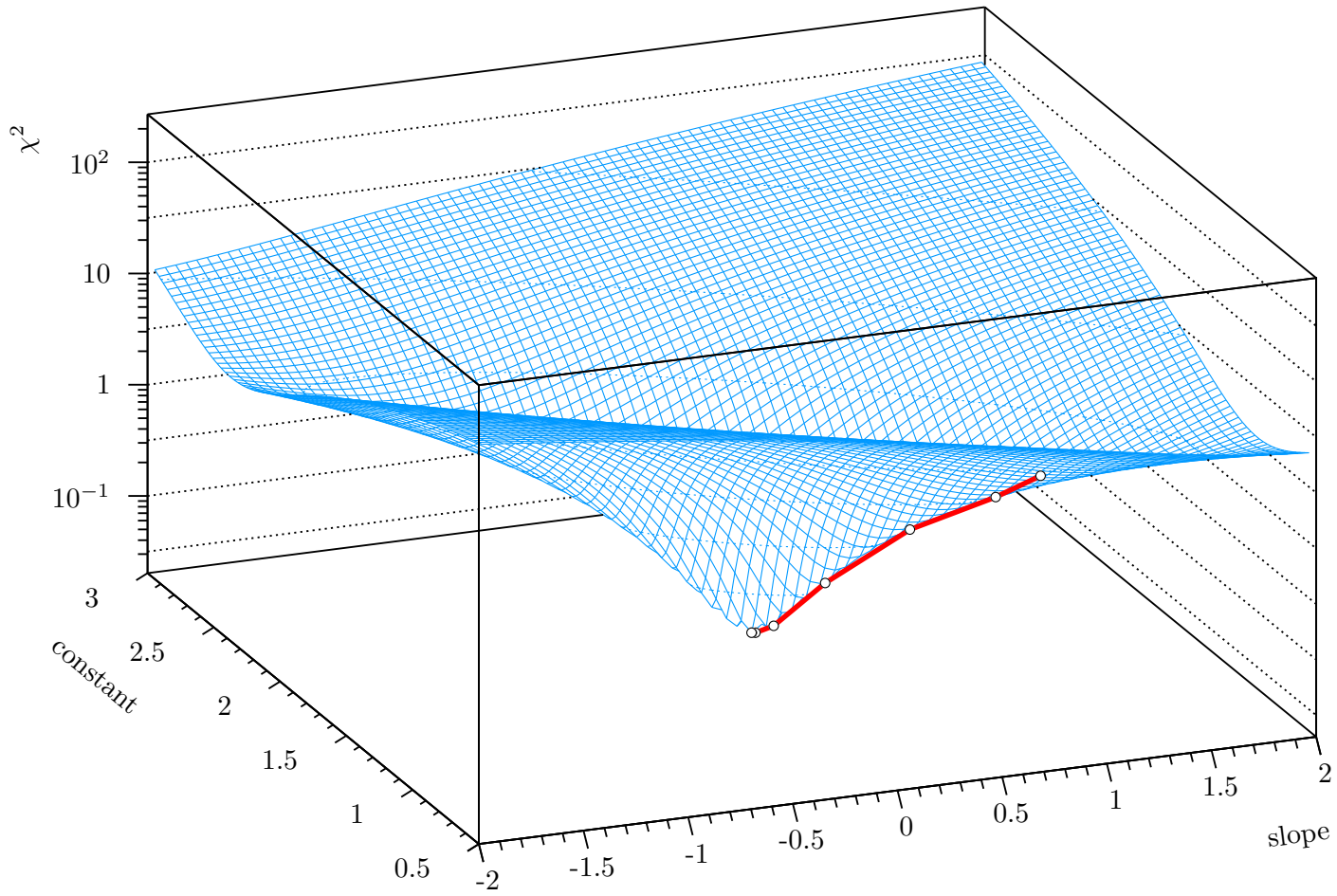
1) A cosa serve un algoritmo di minimizzazione nella ricerca di Materia Oscura ?

Uno dei tipici problemi in fisica è quello di trovare i parametri migliori per descrivere dei dati

Questo si fa **minimizzando** lo scarto tra la funzione in esame al variare i parametri



La strada verso il minimo



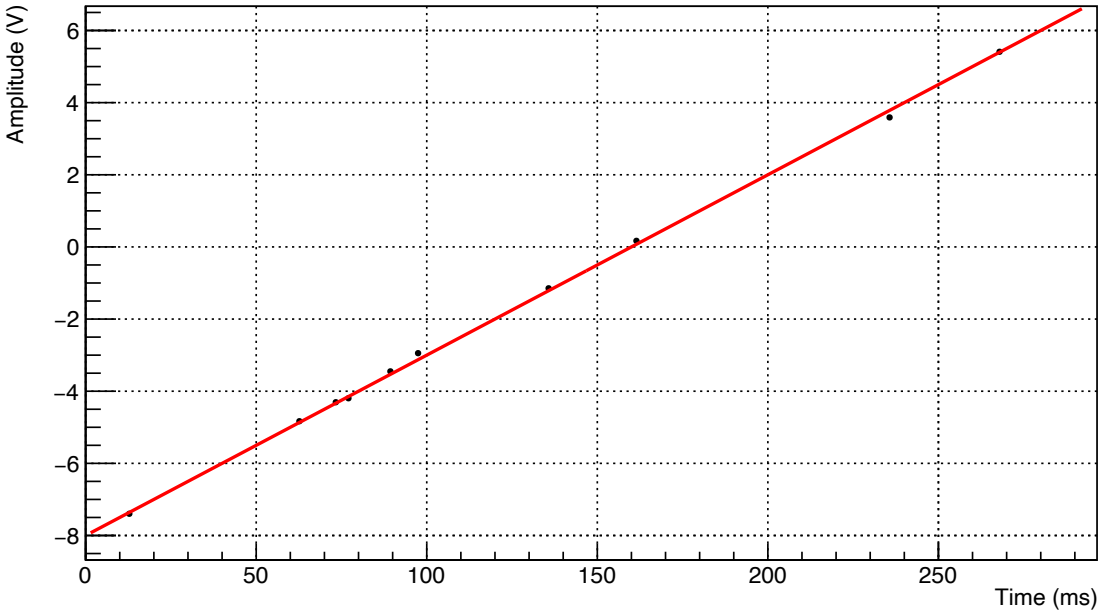
E qui entra in scena l'algoritmo di Levenberg-Marquardt

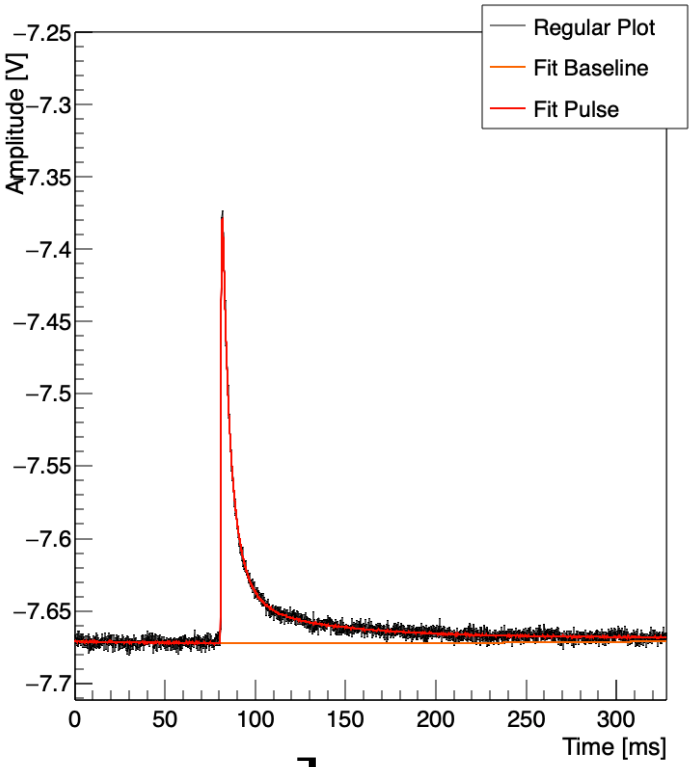
$$\left(J^T J + \lambda I\right) \delta = J^T \left[y - f(\beta)\right]$$

J : Jacobiano	*n x k*
J^T : Jacobiano trasposto	*k x n*
I : Matrice identità	*k x k*
λ : parametro di scala	
y : vettore dei dati	*n x 1*
$f(\beta)$: funzione valutata in β	*n x 1*
β : vettore dei parametri	*k x 1*
δ : variazione del vettore dei parametri	*k x 1*

2) Perché è una buona idea implementarlo su GPU?

2) Perché è una buona idea implementarlo su GPU?

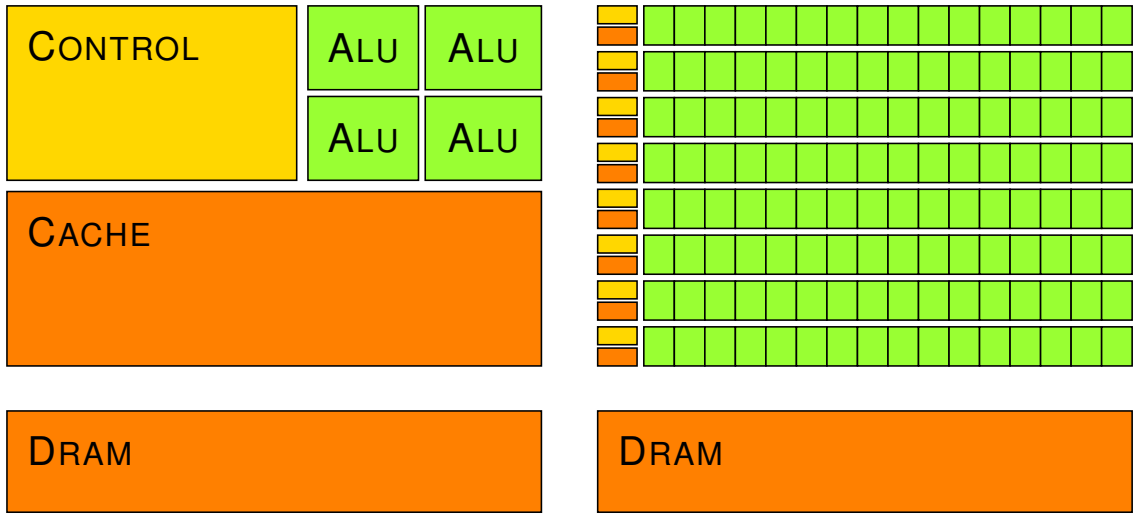




$$\left(J^T J + \lambda I\right) \delta = J^T \left[y - f(\beta)\right]$$

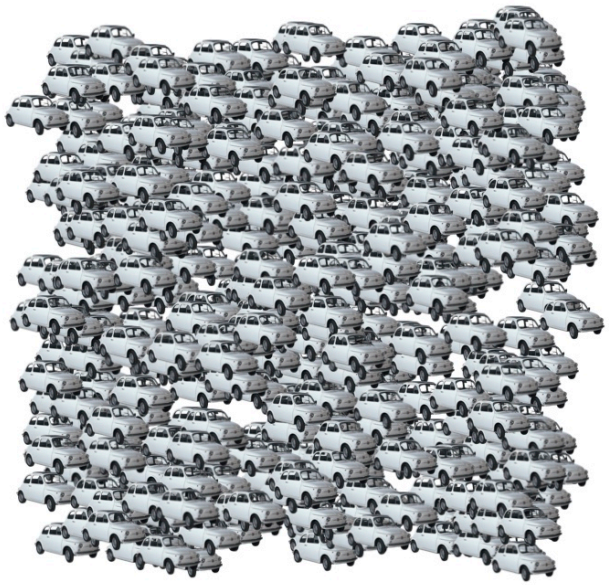
2) Perché è una buona idea implementarlo su GPU?

- La loro struttura è profondamente differente rispetto a quella delle CPU
- È il calcolo matriciale è parallelizzabile

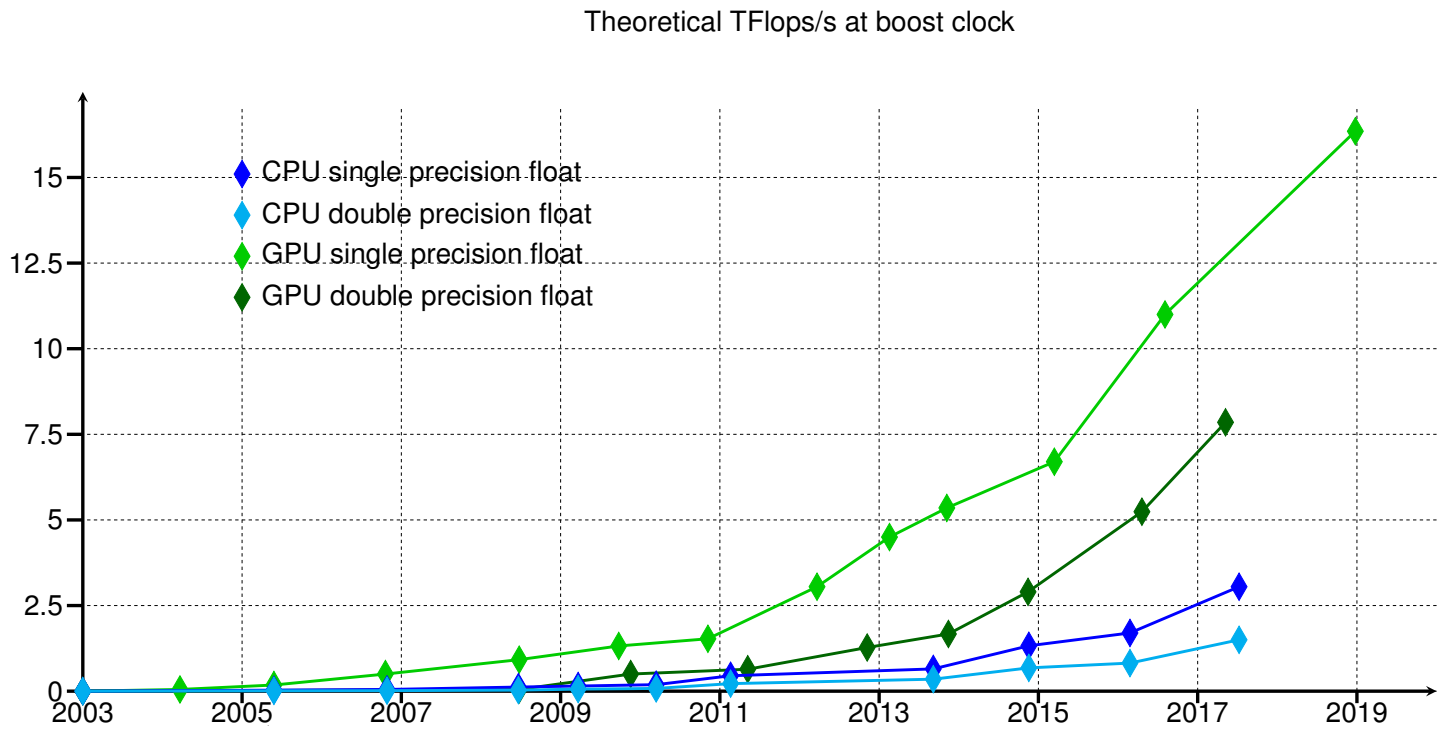


Usiamo un esempio chiarificatore: un trasloco

Usiamo un esempio chiarificatore: un trasloco

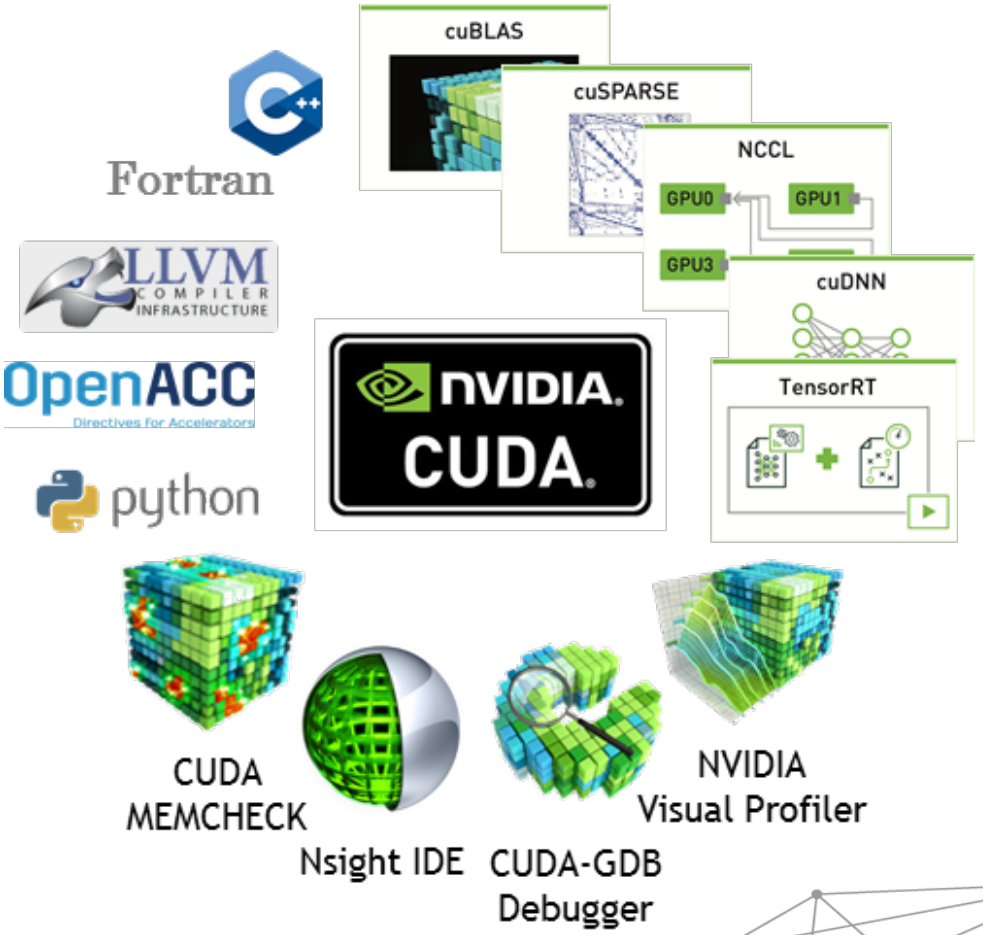


2) Perché è una buona idea implementarlo su GPU?

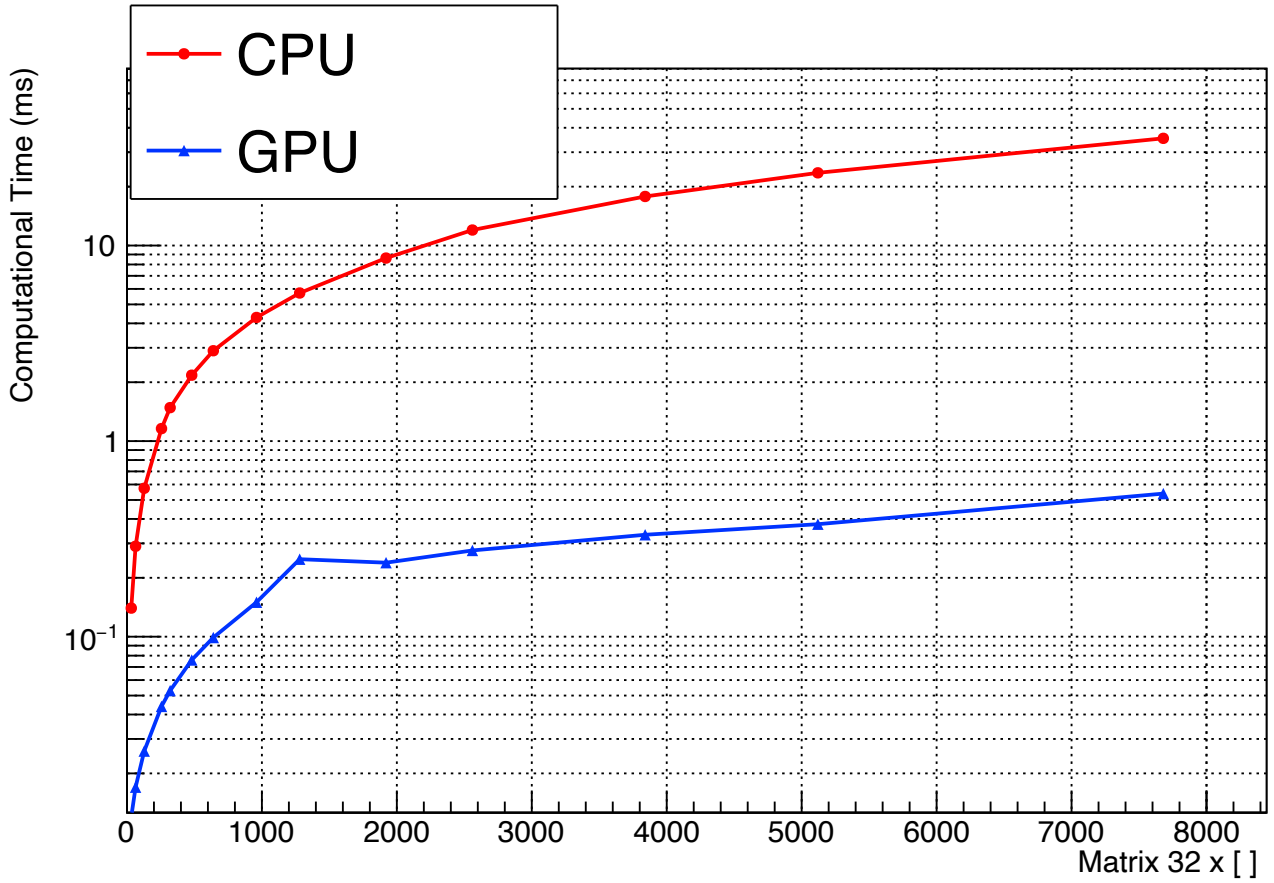


2) Perché è una buona idea implementarlo su GPU NVIDIA? CUDA

CUDA è un toolkit completo di sviluppo fornito da NVIDIA



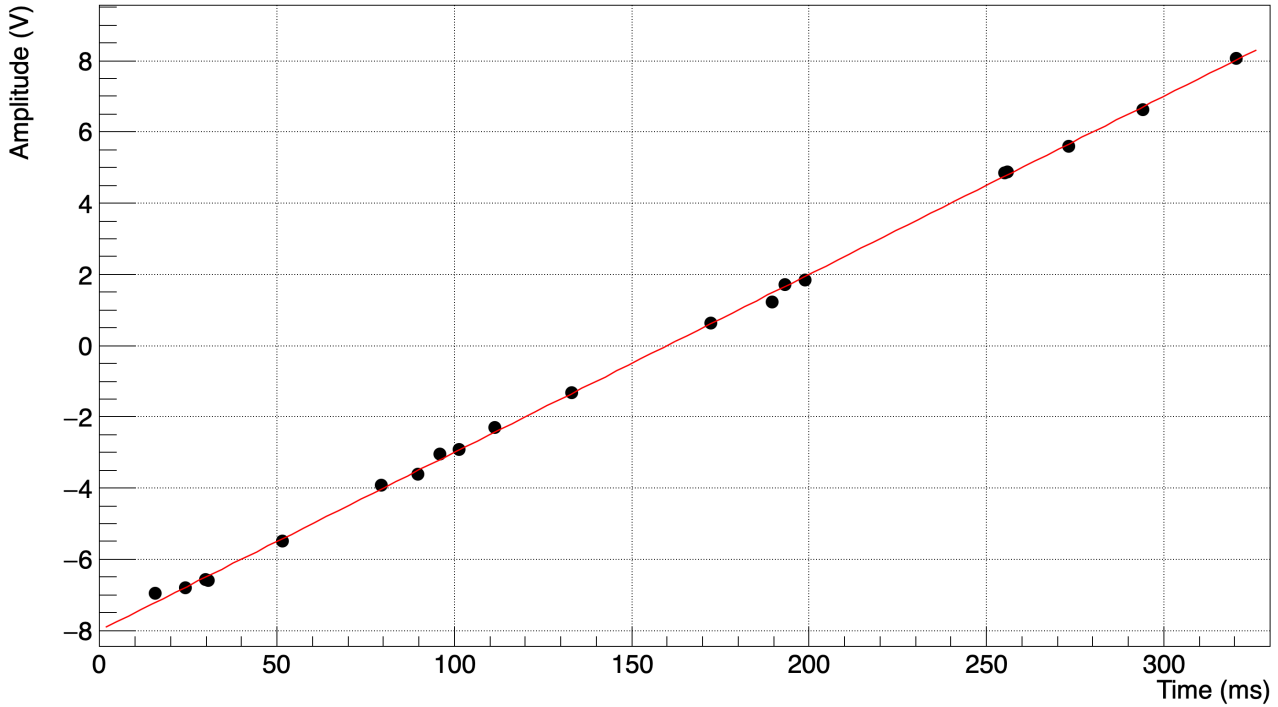
I primi risultati



Nel semplice prodotto tra matrici le GPU mostrano subito di cosa sono capaci

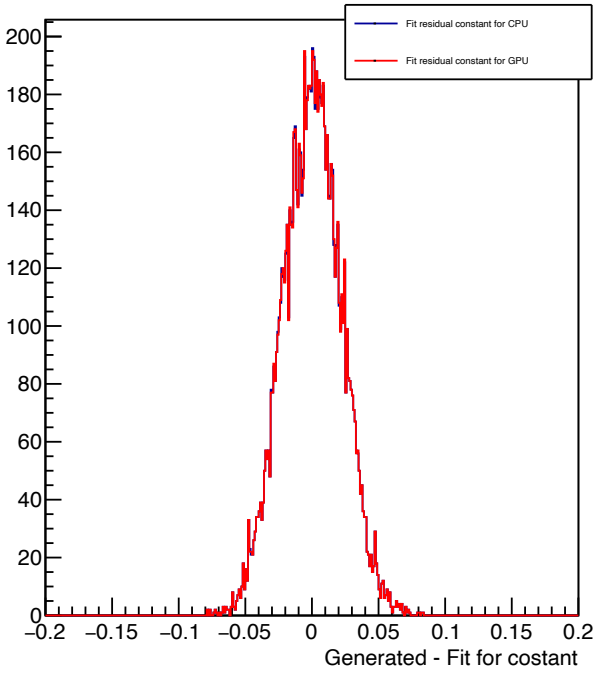
Fit di una retta con GPU, primi test dell'algoritmo

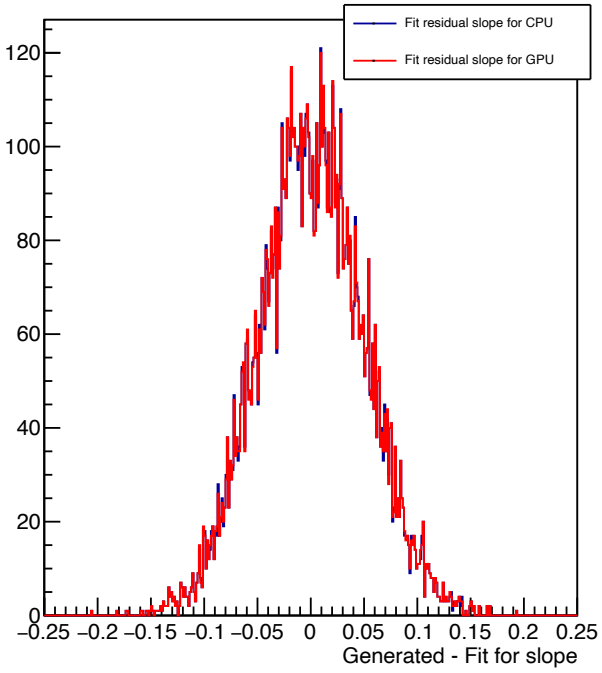
Generazione di rette casuali
Per ogni retta generati 20 punti con rumore
Successivo fit con GPU e CPU per confronto



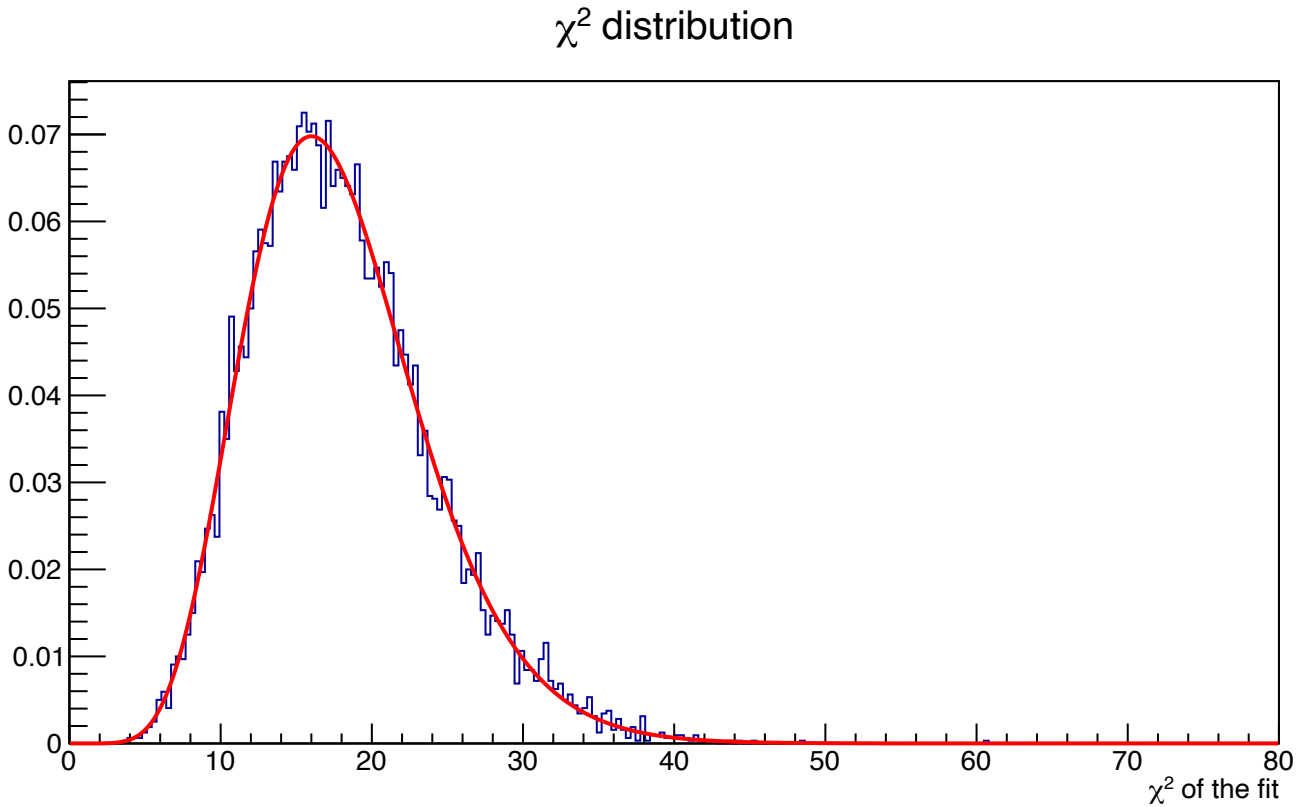
Risultati per il fit di una retta

Ottimo accordo tra CPU e GPU





Risultati per il fit di una retta



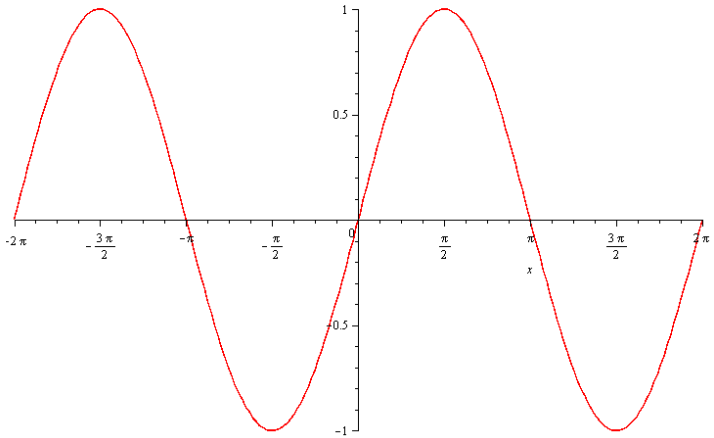
20 punti - 2 parametri
χ^2 con 18 gradi di libertà

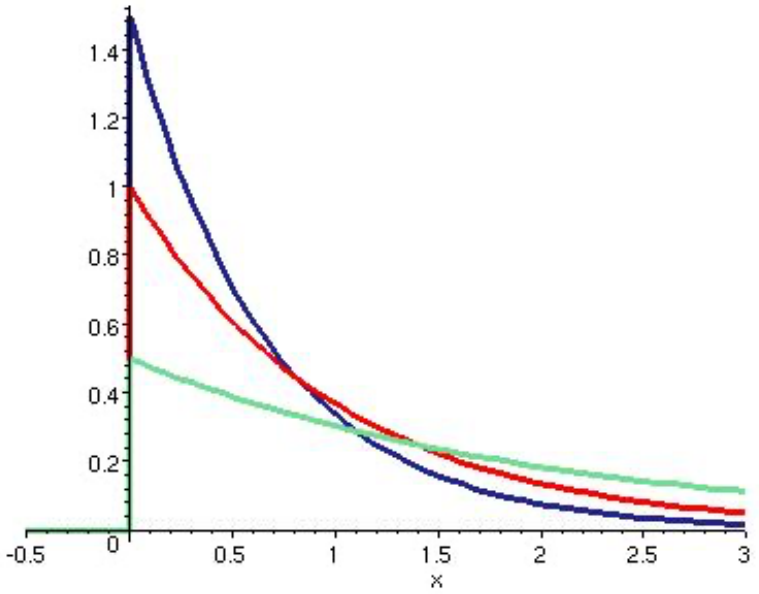
La curva teorica si sovrappone perfettamente con i dati

Fin qui tutto facile…

… si poteva fare anche a mano!!

Esponenziali, seni e coseni sono il vero **STRESS TEST**





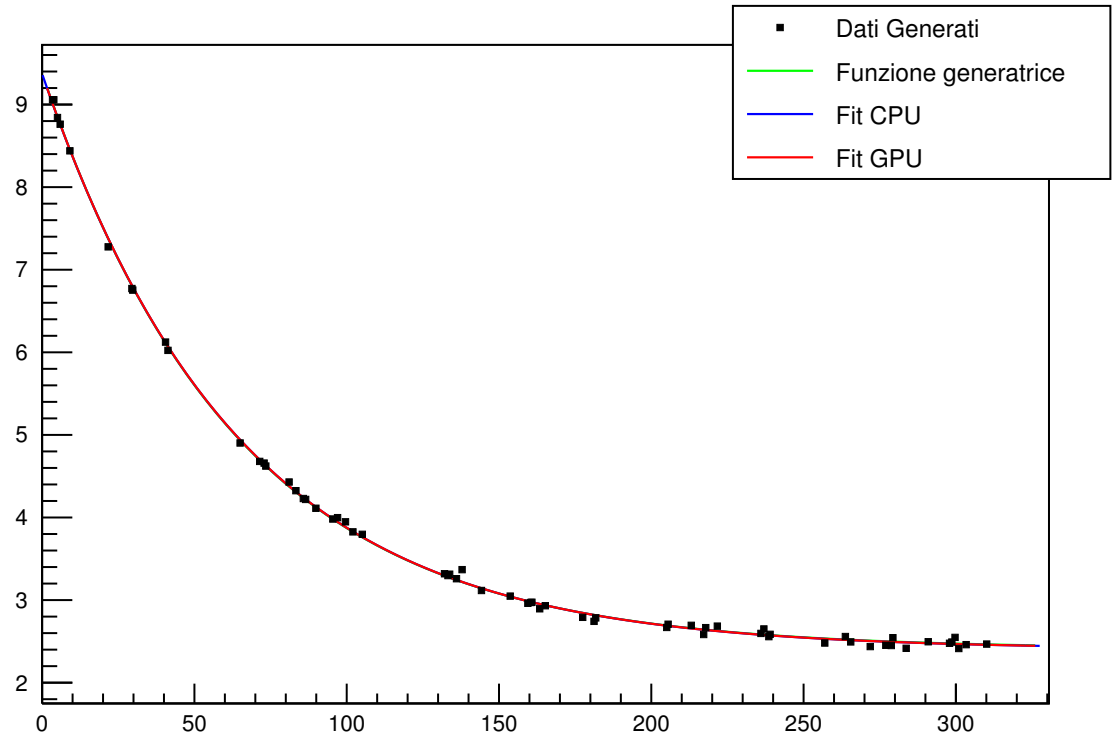
Primo differenza: il Jacobiano

Mentre nel caso di rette era fisso.

In caso di funzioni non lineari va aggiornato e ricalcolato ad ogni iterazione

$$J^T_{j,i} = \frac{\partial f(x_i, p_j)}{\partial p_j} \simeq \frac{f(x_i, p_j + \delta p_j) - f(x_i, p_j)}{\delta p_j}$$

Risultati per fit esponenziali



La funzione più utile ai fini della ricerca di materia oscura è l'esponenziale.

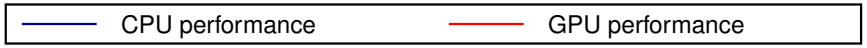
Perciò ho svolto i primi test su questo tipo di funzioni

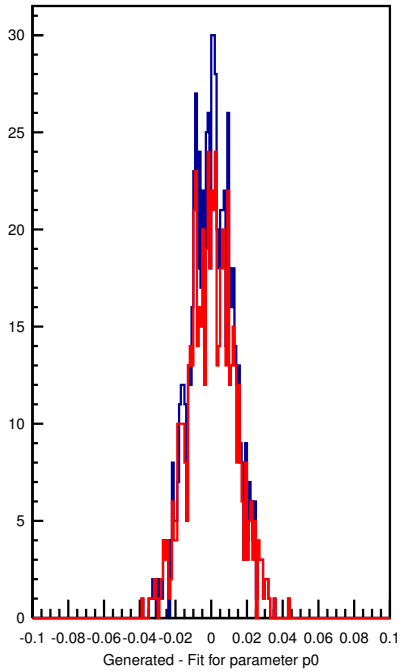
Funzione generatrice:

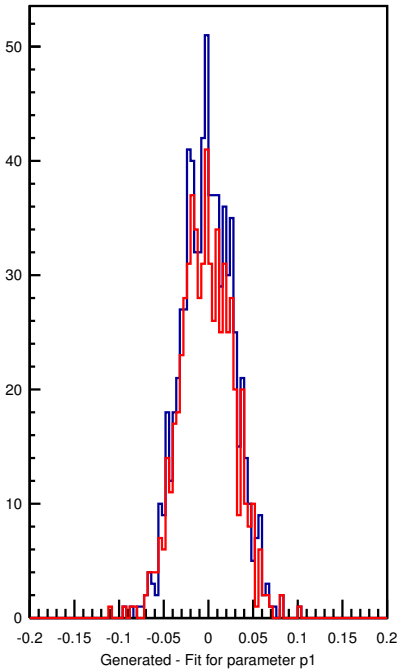
$$f(x,p) = p_0 + p_1 \exp\left(-\frac{x}{p_2}\right)$$

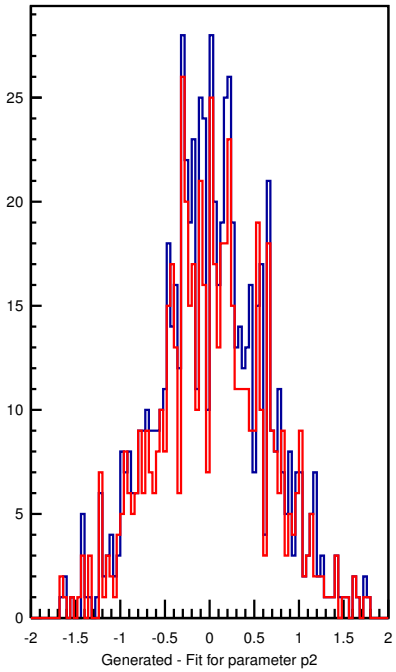
64 punti generati per ogni simulazione

Risultati per fit esponenziali



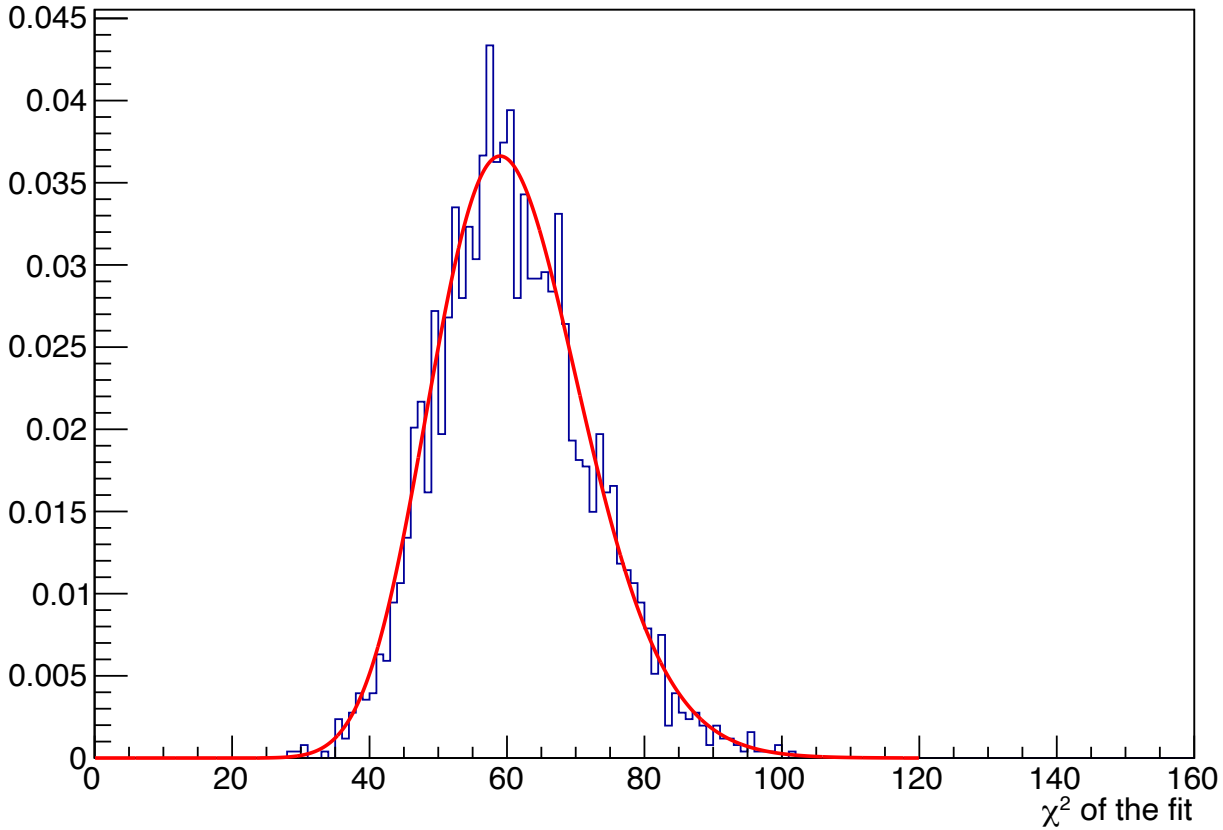






Ottimo accordo tra CPU e GPU

Risultati per fit esponenziali



64 punti - 3 parametri

χ^2 con 61 gradi di libertà

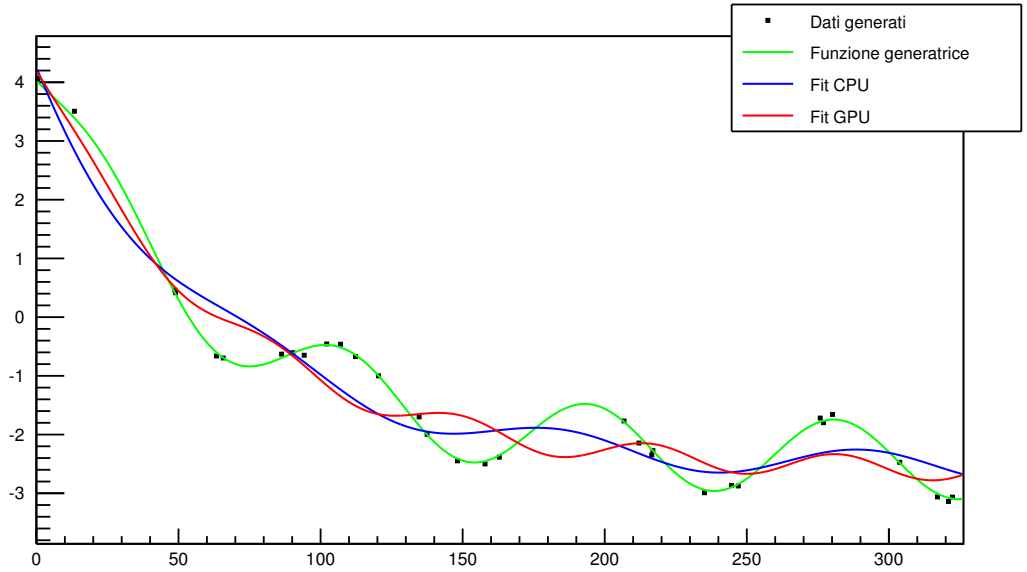
La curva teorica si sovrappone perfettamente con i dati

Ultimo test svolto: esponenziali oscillanti

L’algoritmo mostra i primi suoi limiti.

Il valore iniziale dei parametri è cruciale per un buon fit.

$$f(x,p) = p_0 + p_1 \exp\left(-\frac{x}{p_2}\right) + p_3 \sin\left(\frac{x}{p_4}\right)$$

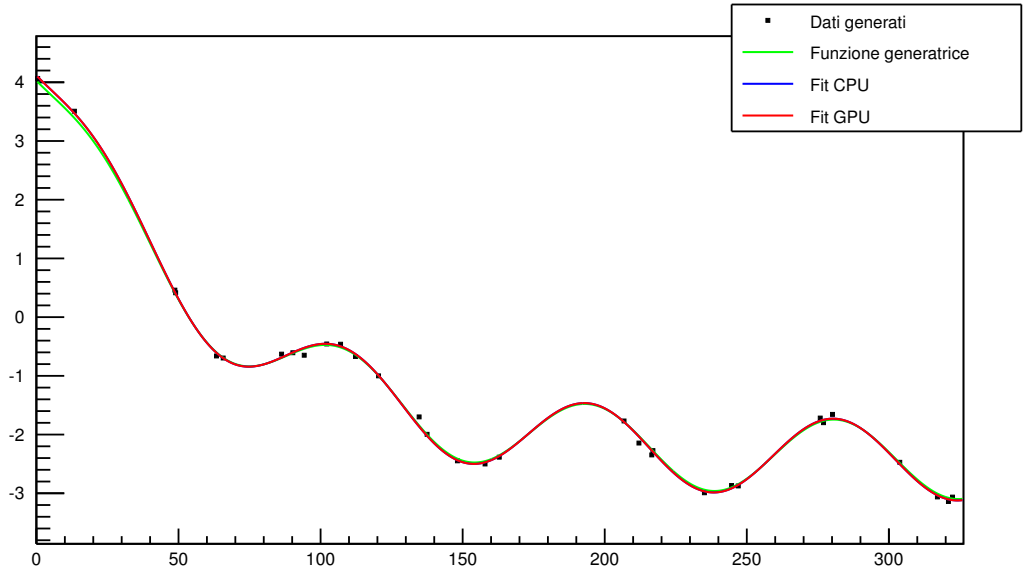


Ultimo test svolto: esponenziali oscillanti

L’algoritmo mostra i primi suoi limiti.

Il valore iniziale dei parametri è cruciale per un buon fit.

$$f(x,p) = p_0 + p_1 \exp\left(-\frac{x}{p_2}\right) + p_3 \sin\left(\frac{x}{p_4}\right)$$

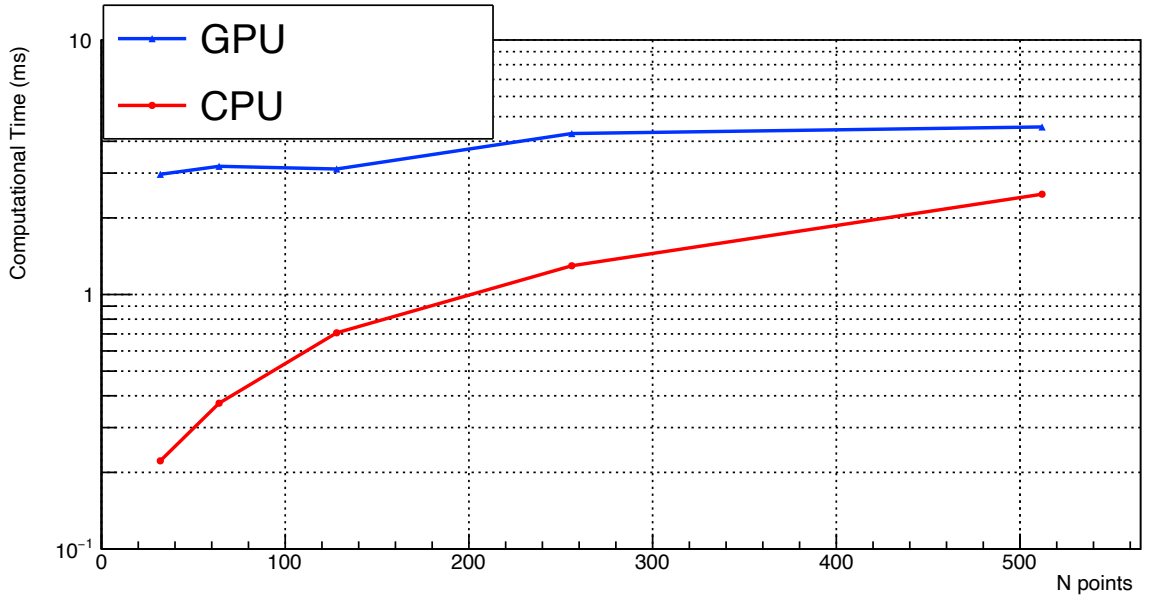


Ma vengono subito risolti!

Primi test temporali

I primi test temporali fatti per fit esponenziali mostrano l'andamento previsto.

Le CPU mostrano un vantaggio che si riduce all'aumentare dei punti.

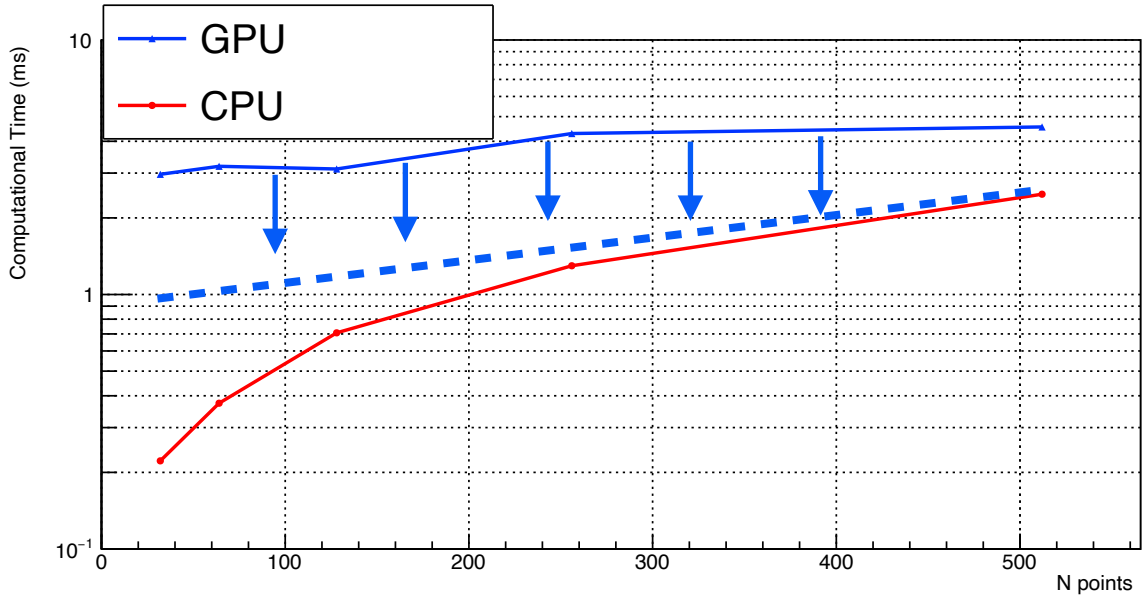


Primi test temporali

I primi test temporali fatti per fit esponenziali mostrano l'andamento previsto.

Le CPU mostrano un vantaggio che si riduce all'aumentare dei punti.

Con una scheda GPU adatta al calcolo scientifico un miglioramento di un fattore ~10



Conclusioni

- Durante l'anno ho iniziato a sviluppare l'algoritmo di Levenberg-Marquardt
- I primi test fatti con l'algoritmo sono molto soddisfacenti in termini di performance
- La valutazione delle performance temporali dell'algoritmo

- I prossimi passi sono i fit degli impulsi di CRESST

Grazie per l’attenzione

Spares

Studio del parametro λ

$\lambda\uparrow/\lambda\downarrow \backslash^{\lambda\downarrow}$	2	3	9	15
1	2.17	2.16	2.05	2.23
1.15	2.24	2.16	2.19	2.14
1.5	2.17	2.17	2.17	2.09
2	2.10	2.27	2.23	2.14
3	2.22	2.19	2.18	2.13

CRESST detector