Relazione di Tirocinio

FFT per grandi moli di dati

Studente: Stéphane Bisinger

Matricola 0900035365

Tirocinio svolto presso l'Istituto di Radioastronomia

1. Obiettivo del tirocinio

Il tirocinio aveva come obiettivo l'utilizzo delle librerie Integrated Performance Primitives (IPP) della Intel, disponibili presso l'indirizzo http://software.intel.com/en-us/intel-ipp/, per la creazione di un programma in grado di effettuare trasformate di fourier ed eventualmente altre trasformazioni su un segnale in entrata. L'obiettivo del programma è di riuscire a processare segnali di grandi dimensioni in tempo reale, così che si possano analizzare direttamente i segnali che provengono dalle rilevazioni dei radioscopi dell'istituto.

2. Sviluppo dell'applicazione

Lo sviluppo dell'applicazione ha previsto diverse fasi:

1. Progettazione
2. Impostazione dell'ambiente per lo sviluppo
3. Librerie e versioni
4. Sviluppo di un generatore di segnali
5. Sviluppo di una prima versione della FFT
6. Aggiunta del Power Spectrum
7. Test della FFT con diversi parametri
8. Sviluppo di una versione che sfrutti il threading

Segue la descrizione delle diverse fasi.

2.1 Progettazione

Durante la progettazione si è scelto di dividere il programma in piccole unità che possano poi venire composte successivamente a seconda delle necessità: l'idea era di sviluppare, in un secondo momento, un programma che permettesse di scegliere da dove acquisire il segnale (file, ftp, ecc.) e quali trasformazioni applicargli (fft, filtro passa alto/basso, ecc.) specificando anche l'ordine delle trasformazioni ed eventuali parametri. Alla fine della catena di trasformazioni si sceglie dove salvare (file, ftp, ecc.) il nuovo segnale generato. Si è scelto di lavorare su interi a 16 bit , anche se tutto rimane facilmente adattabile ad una diversa precisione.

Si è deciso inoltre di utilizzare gli autotools per facilitare la compilazione del progetto.

2.2 Impostazione dell'ambiente di sviluppo

Il primo passo nell'impostare l'ambiente di sviluppo è stato l'installazione delle librerie IPP, le istruzioni incluse sono esaustive e facili da seguire. In seguito si è passati alla configurazione di autoconf e automake: i file di automake sono di semplice scrittura, mentre per autoconf è stato necessario ricercare gli script adatti a verificare la presenza delle librerie IPP e di boost. Trovati questi, l'impostazione del progetto è grosso modo completa e permette già di compilare i sorgenti con il semplice comando ./configure && make.

2.3 Librerie e versioni

2.3.1 IPP

Versione: 6.0.2.076

Le Integrated Performance Primitives sono delle librerie che implementano un insieme molto completo di funzioni e algoritmi relativi al processamento di segnali e crittografia. In particolare implementano funzioni utili al processamento di segnali audio (filtri, codifiche audio, compressione dei dati, ecc.), funzioni per il processamento di immagini (trasformazioni, codifica video, ecc.), funzioni per calcoli su matrice e funzioni crittografiche (crittografia simmetrica, asimmetrica, funzioni hash, ecc.). Il vantaggio delle IPP è che offrono un ampio spettro di funzioni che coprono praticamente tutte le necessità nel processamento dei segnali, oltre ad essere ottimizzate per processori Intel. Inoltre sono state scritte in modo da essere naturalmente utilizzabili in ambienti multithreaded, sfruttando tutte le potenzialità di processori multicore o processori multipli. Tutti questi fattori hanno concorso nell'adozione di questa libreria al posto di altre librerie concorrenti.

2.3.2 Boost

Versione: 1.36.0

Le librerie boost sono molto usate nella programmazione in C++ perché implementano moltissime funzioni spesso usate fornendo una interfaccia molto semplice. L'ottima qualità di queste librerie è confermato dal fatto che spesso alcune sue componenti vengono integrate nello standard C++. In questo progetto vengono utilizzate principalmente per semplificare la lettura di argomenti da linea di comando, la lettura/scrittura di files e il multithreading. A causa di modifiche sostanziali nel modo di sfruttare il multithreading implementato da boost, è necessario usare una versione maggiore o uguale alla 1.36.0

2.3.3 Autotools: Autoconf, Automake

Versione Autoconf: 2.61

Versione Automake: 1.10.2

Gli autotools sono lo strumento di gestione progetti più diffuso nel software open source. Anche se a volte un po' complicati da usare, la semplificazione della gestione del progetto rendono il loro utilizzo indispensabile: con un solo comando, infatti, è possibile compilare, testare e creare un archivio del progetto pronto per essere redistribuito. È importante ricordarsi che questi tool tendono a cambiare comportamento tra le singole versioni, quindi va sempre controllato che non vadano fatti aggiornamenti ai files di configurazione.

2.3.4 Git

Versione: 1.6.x.x

Git è un sistema di versioning distribuito ed è usato per gestire la storia del progetto. Benché il suo apprendimento sia piuttosto difficile inizialmente, le potenzialità dello strumento e le sue ottime prestazioni diventano presto irrinunciabili. Inoltre la sua natura distribuita ha permesso di creare e gestire un repository senza bisogno di un server centrale.

Il progetto è comunque presente su github: http://github.com/Kjir/ira/

2.4 Sviluppo di un generatore di segnali

Il primo programma sviluppato è stato un generatore di segnali che fornisca dei risultati noti una volta effettuata la trasformata di Fourier. A questo scopo nelle librerie IPP esiste una funzione che genera una sinusoide, riducendo quindi la difficoltà della creazione del programma solamente a lettura delle opzioni e scrittura dell'output. Il programma serve anche da base per lo sviluppo degli altri programmi.

2.5 Sviluppo di una prima versione della FFT

La prima versione sviluppata della FFT serve principalmente a testare la funzione delle IPP, valutare la sua correttezza e capire il suo comportamento, oltre a valutare la velocità di calcolo. Oltre ad utilizzare la funzione delle IPP, va letto l'input, vanno interpretati gli argomenti e va scritto il risultato in output. Una volta scritta la funzione, si può testarne il funzionamento dall'output del generatore. È a questo punto che si è presentata la necessità di scrivere un piccolo programma in Python che disegni un grafico dai dati di output dei precedenti programmi, sfruttando le possibilità offerte dalle librerie numpy e matplotlib.

2.6 Aggiunta del Power Spectrum

Dopo aver verificato il corretto comportamento della trasformata, si è scelto di aggiungere il calcolo dello spettro di potenza per rappresentare il segnale in frequenza, operazione gestita da una funzione IPP e controllata da un parametro da linea di comando. Contestualmente è stata aggiunta la possibilità di sommare tra loro i risultati di due o più trasformate consecutive sullo stream di dati in modo da far emergere chiaramente i segnali dal rumore di fondo.

2.7Test della FFT con diversi parametri

A questo punto si è reso necessario testare l'andamento della trasformata al variare di parametri. A questo scopo è stato sviluppato un programma che misura il tempo impiegato a fare le trasformazioni e che provi diverse combinazioni di opzioni. Il primo test osserva l'andamento della trasformata al variare dell'ordine del segnale (la sua lunghezza, quindi) con e senza integrazioni. Infine si procede ad osservare il comportamento della funzione quando si aumentano il numero di somme consecutive da eseguire.

2.8 Sviluppo di una versione che sfrutti il threading

I test eseguiti al punto precedente hanno mostrato che il threading interno alla libreria non basta a sfruttare ottimamente le risorse disponibili e quindi si è deciso di creare una versione

della trasformata che non lavori sequenzialmente quando deve eseguire le somme su tratti consecutivi del segnale, ma che lavori in concorrenza. La struttura del programma viene quindi leggermente modificata per fare in modo che possano essere create due versioni della stessa funzione che possano essere sostituiti per verificare il comportamento in entrambi i casi. La nuova versione ha un thread che legge l'input dal file, mentre altri thread si occupano di calcolare la trasformata. A questo modo si sfrutta meglio la CPU durante la lettura da file.

3. Risultati dei test e conclusioni

I test mostrano che le librerie IPP sono ben adatte al tipo di performance che si ricerca per il tipo di lavoro che si vuole svolgere. In particolare si riescono ad effettuare FFT di lunghezza 2^{23} in meno di un secondo, permettendo quindi di effettuare diverse elaborazioni in tempo reale oltre alla FFT. I grafici sui tempi, inoltre mostrano che il tempo di calcolo cresce esponenzialmente all'aumentare dell'ordine di grandezza del segnale [grafico1] e che si comporta allo stesso modo sia con integrazioni che senza [grafico2], mentre cresce linearmente con l'aumentare del numero di integrazioni [grafico3]. L'attivazione o meno del threading della libreria non varia il tempo impiegato a calcolare una FFT [grafico4 e grafico5], mentre l'aver sviluppato una versione che effettui in parallelo le FFT parte di una integrazione hanno migliorato le prestazioni, anche se non di grandi valori come ci si poteva aspettare [grafico6 e grafico7]. Tuttavia l'utilizzo dei diversi core della CPU mostra come la seconda versione sfrutti meglio il multiprocessing, facendo sospettare quindi che la lettura dell'input influisca forse anche più del calcolo della trasformata.

Tutti i test sono stati eseguiti su una CPU AMD Athlon 64 X2 Dual Core 5600+ su sistema operativo Linux e kernel 2.6.25 per processori a 64 bit.

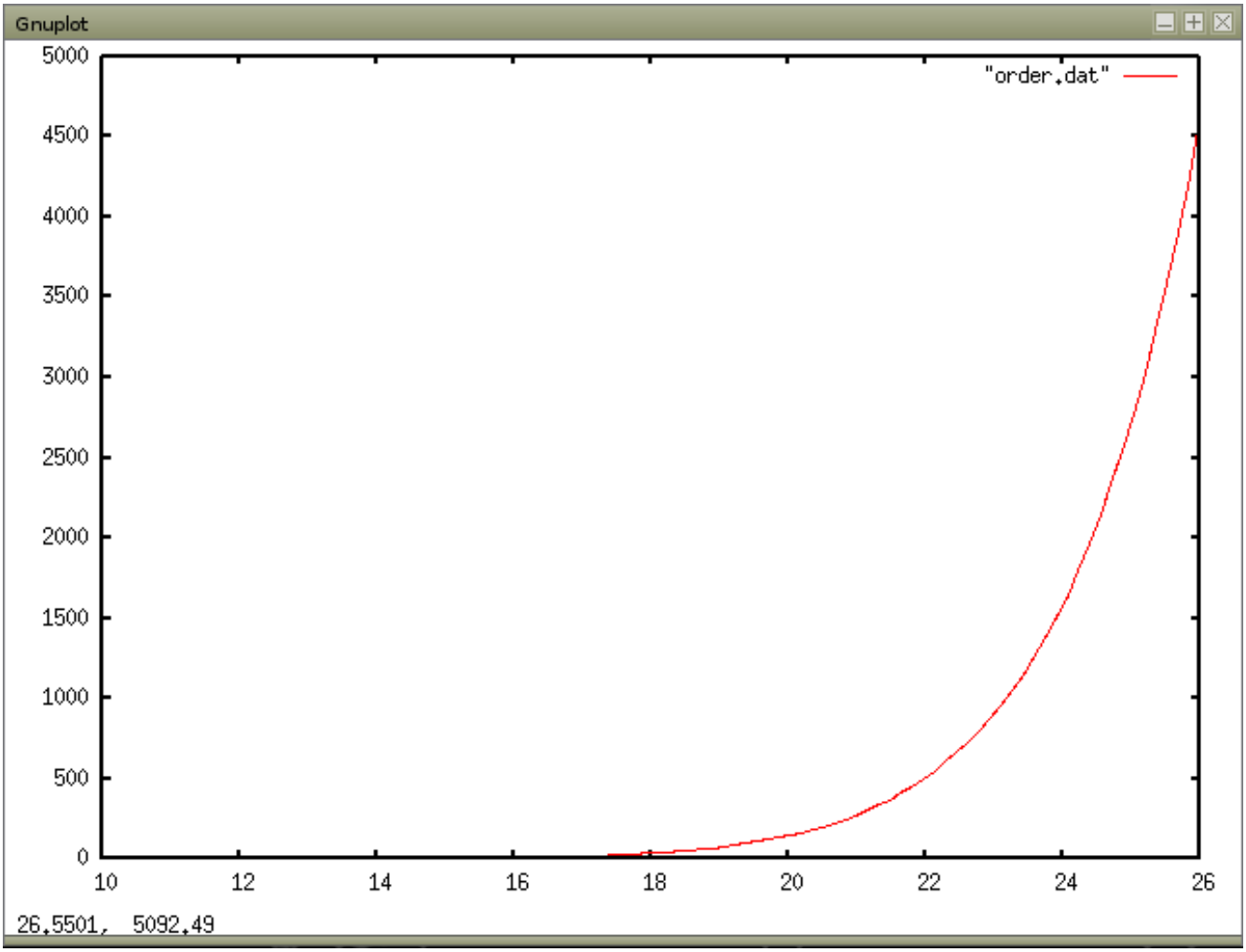

Grafico 1: Tempo impiegato all'aumentare dell'ordine della FFT

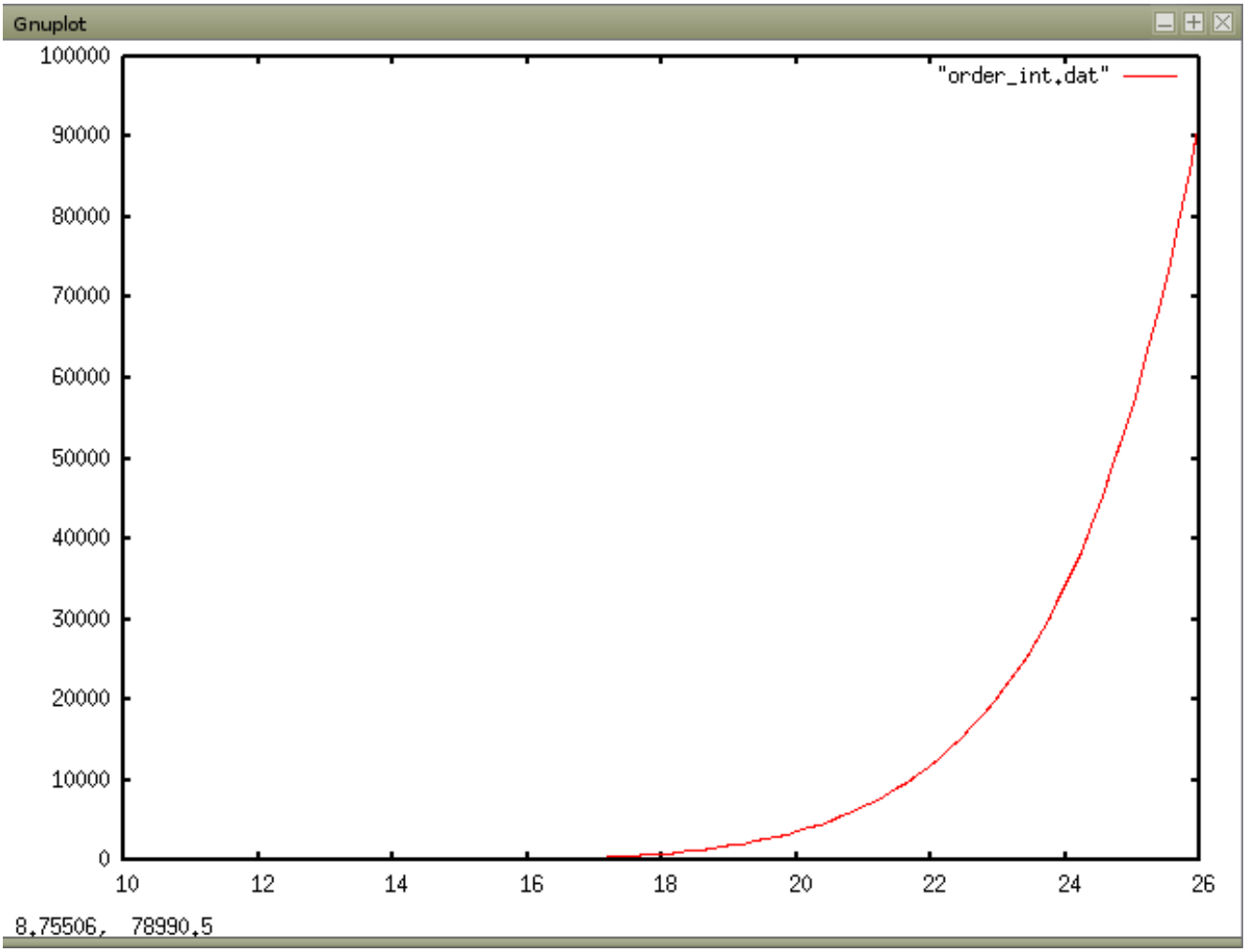

Grafico 2: Tempo impiegato all'aumentare dell'ordine della FFT, con 30 integrazioni

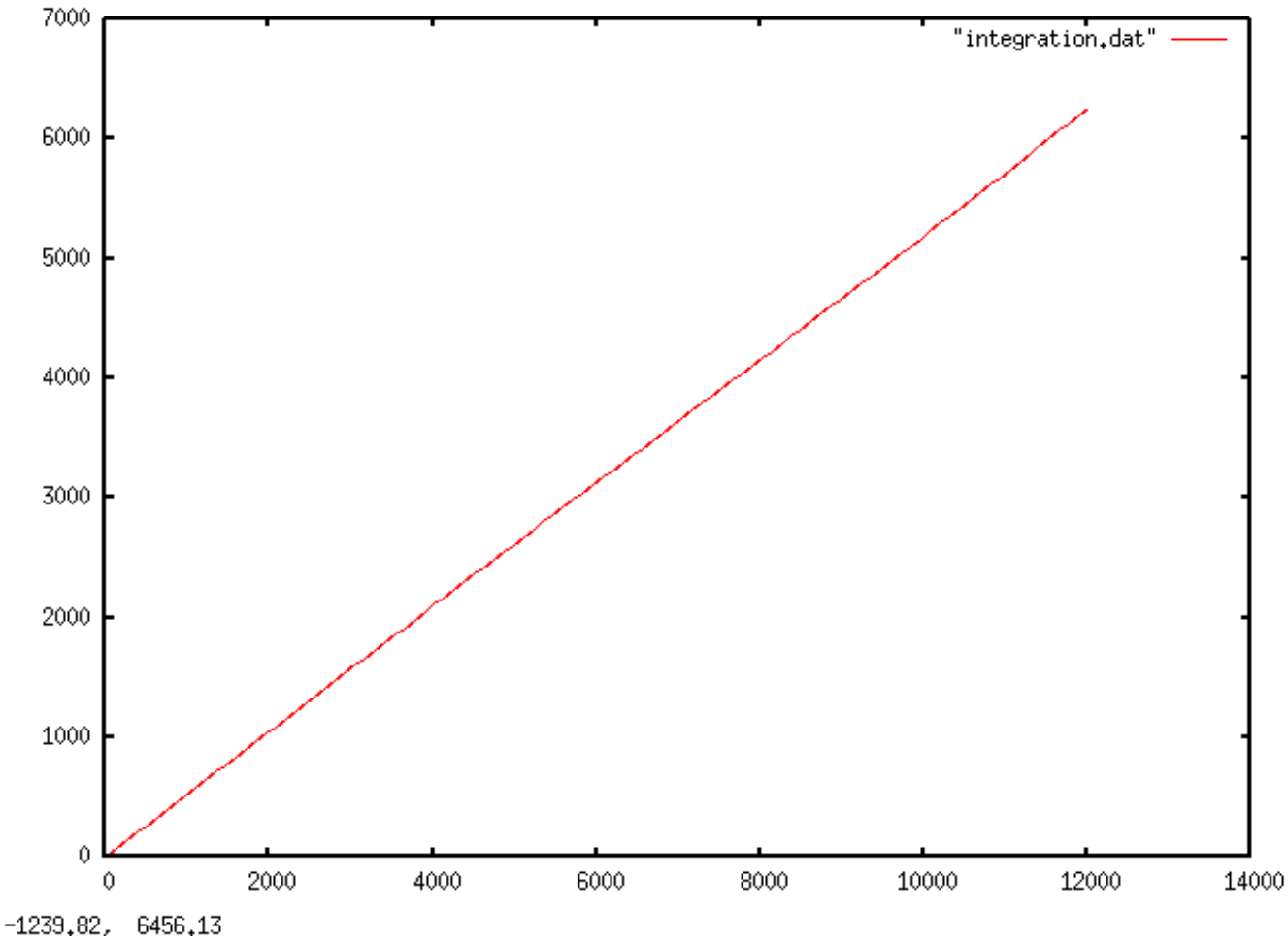

Grafico 3: Tempo impiegato all'aumentare del numero di integrazioni

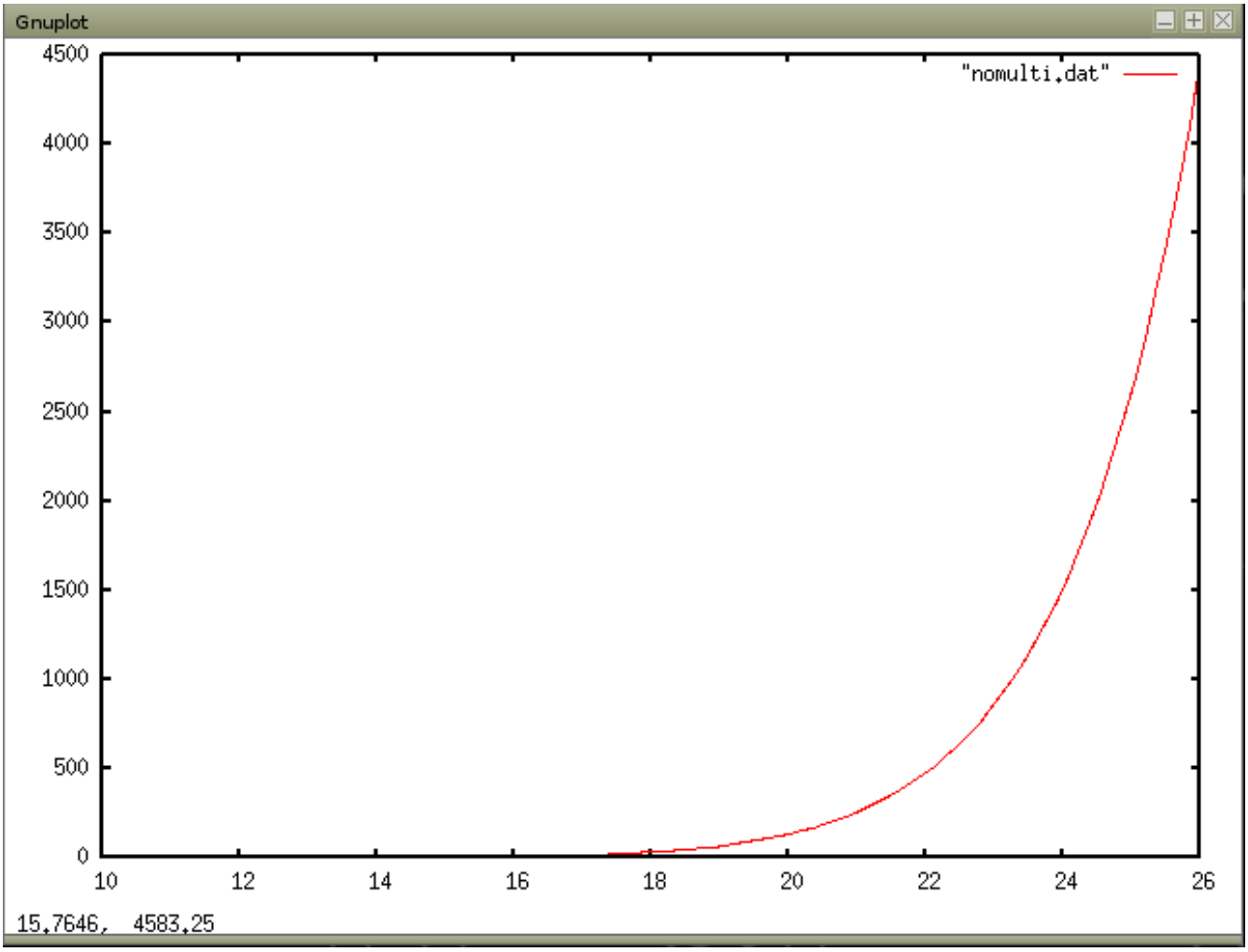

Grafico 4: Tempo impiegato a calcolare la FFT con ordine crescente, senza multithreading

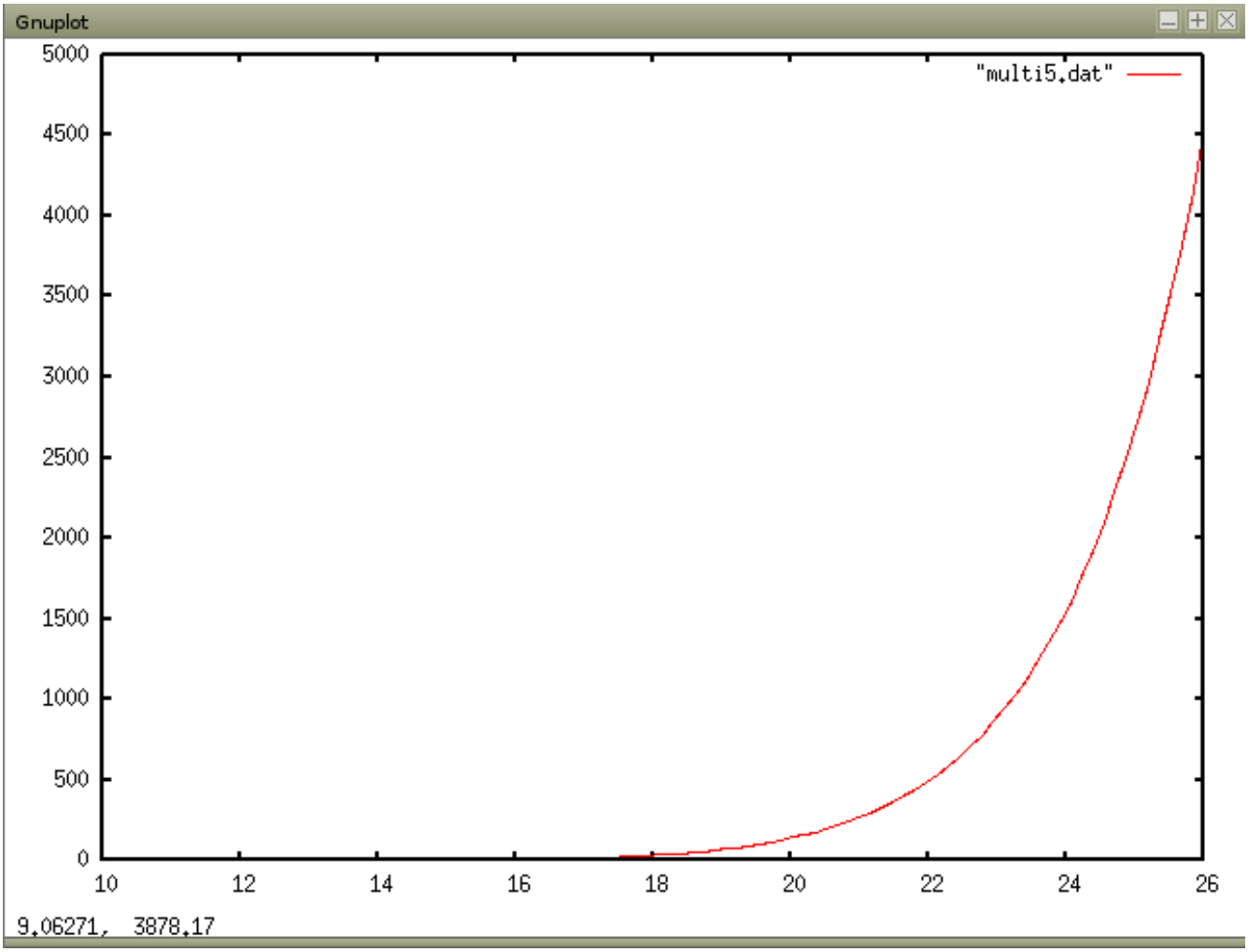

Grafico 5: Tempo impiegato a calcolare la FFT con ordine crescente, con 5 thread (libreria)

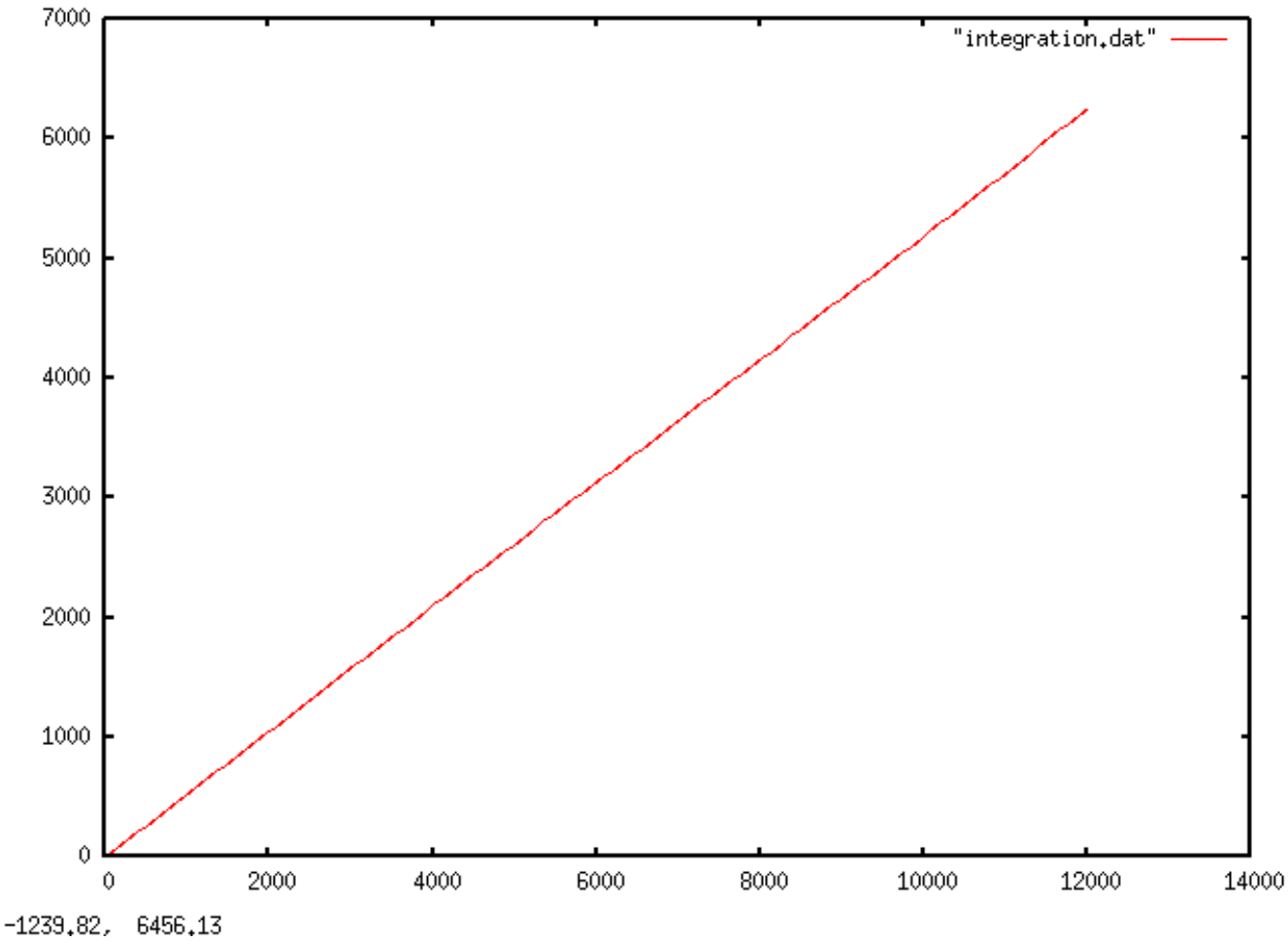

Grafico 6: Crescere del tempo all'aumentare del numero di integrazioni, senza threading

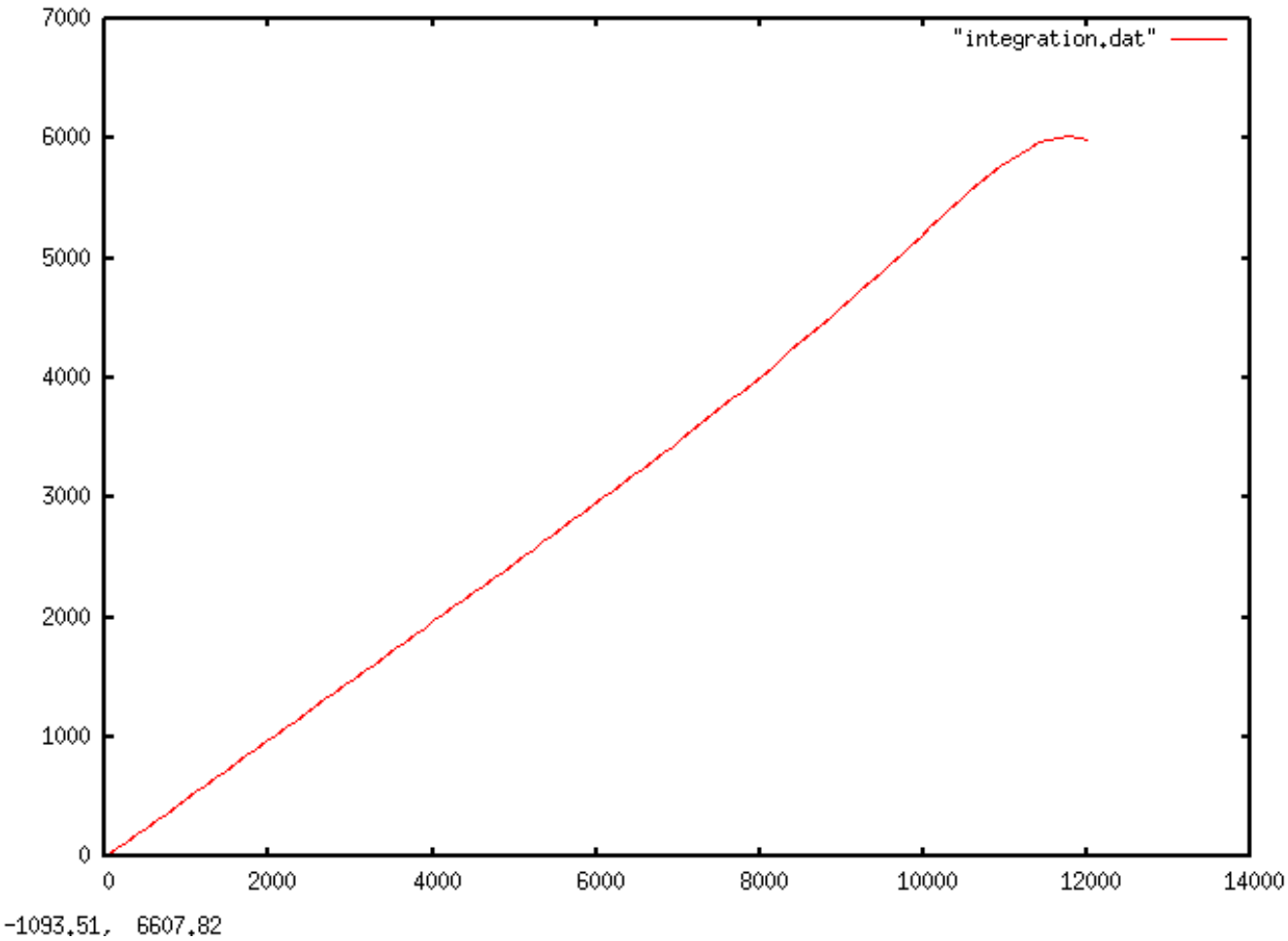

Grafico 7: Crescere del tempo all'aumentare del numero di integrazioni, con multithreading