Alma Mater Studiorum · Università di Bologna

**Scuola di Scienze**
**Corso di Laurea in Fisica**

# Implementazione, creazione e ottimizzazione di una pipeline per l'analisi biofisica su cluster a basso consumo energetico

**Relatore:**
**Dott. Enrico Giampieri**

**Presentata da:**
**Daniele Dall'Olio**

**Correlatore:**
**Prof. Gastone Castellani**
**Ing. Andrea Ferraro**

**Sessione II**
**Anno Accademico 2016/2017**

# Prefazione

In questa tesi si è studiata l'efficienza computazionale di nodi di calcolo a basso consumo energetico per l'analisi biofisica, confrontati con nodi tradizionali. Questo lavoro è parte di un progetto per valutare la fattibilità dell'utilizzo di macchine a basso consumo energetico per calcolo ad alta performance. Lo scopo della ricerca è provare che l'utilizzo di un insieme di macchine(cluster) a minor dispendio energetico(low power) possano fornire una potenza di calcolo confrontabile con quella delle macchine tradizionali, consumando una minor quantità di energia e contenendo la spesa economica.

Il sistema su cui si è concentrato il lavoro di tesi è uno dei metodi più recenti nella ricerca sulle mutazioni genetiche che sono cause di vari tipi di tumori: il sistema GATK-LOD$_{\mathrm{n}}$. Nel corso della tesi è stata reimplementata una componente di questo metodo in una pipeline nel programma Snakemake, che ha permesso una gestione più accurata delle operazioni previste per ottimizzare l'esecuzione complessiva. Questa tesi prende in esame questo algoritmo di bioinformatica(GATK-LOD$_{\mathrm{n}}$) per valutare se è realmente possibile confrontare le capacità dei nodi low power con quelli tradizionali, in quanto questo richiede alte prestazioni computazionali, di memoria e capacità di storage.

La scelta di adoperare i cluster low power nasce dalla necessità dei moderni organi di ricerca di potenziare le capacità computazionali, ottimizzando l'investimento sulle macchine server. Gli studi nel campo biomedico prevedono infatti la collaborazione di professionisti in analisi dati che utilizzano server ad alta potenza e ad alto consumo, con il compito di fornire analisi su grandi mole di dati ed in tempi ristretti.

Lo sviluppo e il progresso in questi vari settori necessita di un costante incremento della potenza di calcolo delle macchine impiegate. Questo comporta un inevitabile innalzamento dei costi, che induce ad una minor accessibilità alla maggioranza dei gruppi di ricerca. Il grande investimento necessario per l'acquisto di una singola macchina limita inoltre la frequenza con cui si possono acquistare nuove macchine, rendendo quindi difficile sfruttare gli avanzamenti della tecnologia in quanto si è spesso costretti a lavorare con macchine di generazioni precedenti.

L'aumento delle capacità di calcolo comporta inoltre un aumento del consumo energico per la gestione delle macchine, che occupa una percentuale rilevante nelle spese complessive.

Nell'ultimo decennio, tali ragioni hanno portato quindi ad interessarsi a metodi alternativi per la computazione, come l'utilizzo di nodi su cluster a basso consumo energetico, argomento di questa tesi. L'idea di fondo

è permettere ai ricercatori, a parità di investimento, di ottenere risultati comparabili a quelli ottenuti con i nodi tradizionali.

Nel primo capitolo saranno introdotti e approfonditi gli elementi cardine del progetto. Sarà esposto il metodo GATK-$LOD_n$ sia nel funzionamento che nei risultati. Successivamente sarà descritta la componente del metodo che è stata reimplementata tramite Snakemake e saranno approfondite le capacità di questo strumento. Infine, sarà spiegato il significato di "nodo low power" e saranno descritte le caratteristiche dei nodi adoperati nelle analisi.

Nel secondo capitolo sarà spiegato dettagliatamente il funzionamento del programma, approfondendo i parametri utilizzati, e verranno evidenziati i passaggi necessari per un corretto uso del metodo, dall'installazione alla produzione dei dati. In più, saranno descritte le fasi dello studio statistico e sarà spiegata la tipologia di simulazioni effettuate.

Nel terzo capitolo verrano discussi i risultati finali più rilevanti per ciascuna regola della pipeline in termini di tempi di esecuzioni e memoria occupata.

Per terminare saranno delineate le conclusioni, le considerazioni finali e gli sviluppi futuri del progetto, in base ai risultati ottenuti.

# Indice

# Capitolo 1

# Introduzione

Questo capitolo introduce i componenti principali del progetto ed ha il compito di spiegarne le operazioni svolte in maniera approfondita. Tali componenti sono raggruppati nei tre campi su cui è stato condotto il lavoro: la bioinformatica, lo sviluppo informatico e i nodi low power.

La prima sezione, relativa al campo bioinformatico, approfondisce il metodo GATK-LOD$_{\mathrm{n}}$ riguardante la ricerca sull'origine dei tumori attraverso procedure di natura bioinformatica. Inoltre, è descritto il ruolo dei fisici in questo ambiente di lavoro e nella creazione degli algoritmi.

La seconda sezione, inerente allo sviluppo informatico, spiega come le funzionalità del programma Snakemake concedano ampie possibilità sulla gestione delle risorse offerte dai nodi delle macchine server.

La terza sezione descrive l'importanza delle macchine low power e specifica quali tipi di nodi sono stati adoperati nelle analisi.

## 1.1 La ricerca delle mutazioni genetiche

Questa sezione si occupa di dare un'idea generale sugli studi che riguardano le mutazioni genetiche che causano e alimentano i tumori.

Negli ultimi anni l'indagine sulle forme cancerogene basata sulle variazioni che avvengono nel codice genetico ha suscitato sempre più interesse e ciò ha portato allo sviluppo di un discreto numero di programmmi, algoritmi e metodi atti all'analisi del DNA. Ognuna di queste applicazioni ha come fine la determinazione di quelle mutazioni nei geni che comportano l'insorgere delle malattie tumorali oggi conosciute. La conoscenza di una tale correlazione è di vitale importanza per la pianificazione di un piano di cura adeguato e, in altri casi, per un intervento anticipato in grado di prevenire il presentarsi della malattia.

Un altro aspetto delicato è la risposta del cancro alle cure a cui è sottoposto il paziente e che, in taluni casi, si concretizza in una forma di resistenza a questi interventi. La gestione di una tale conseguenza può essere aiutata dalla piena comprensione dell'origine genetica del tumore, la quale agevolerebbe la scelta tra i percorsi di guarigione più opportuni.

Seppur avendo lo stesso obiettivo, gli innumerevoli algoritmi utilizzati spesso si ritrovano in contrasto tra di loro e queste discrepanze sono rilevate quando si procede con il confronto dei risultati finali, che solitamente o non concordano in parte o sono proprio differenti. Questi algoritmi elaborano i dati sperimentali grezzi attraverso vari processi che generalmente adoperano svariati metodi di statistica, le cui radici provengono dai campi di matematica e fisica applicata. La netta differenza di prestazioni, insieme alle diverse metodologie operate, lascia ai ricercatori un arduo compito nella scelta del metodo più idoneo da applicare.

In questa tesi il metodo considerato prende il nome di GATK-$LOD_n$ ed è ideato dalla combinazione di due tra i tools più comuni nel panorama bioinformatico: GATK e MuTect. Prima di esporre questo algoritmo e le sue fondamenta, è necessario definire il percorso storico che ha determinato le tecniche moderne con cui è analizzato attualmente il materiale genetico e le caratteristiche generali di questi procedimenti. Inoltre, sarà sottolineato il ruolo degli studi fisici durante il progresso nei campi di biologia e medicina.

### 1.1.1 L'analisi del DNA

Le ricerche sull'eredità genetica iniziarono dagli studi di Gregory Mendel nella seconda metà del diciannovesimo secolo riguardo all'ibridizzazione tra le piante. Nella prima metà del secolo seguente, un serie di contributi, tra cui l'immagine della diffrazione a raggi X della doppia elica da parte di Roselind Franklin, posero le basi alla moderna ricerca genetica.

Il contributo essenziale fu la determinazione della struttura del DNA a cui contribuirono James Watson e Francis Crick, la quale permise di cominciare la decifrazione del codice genetico. I tentativi di decodifica subirono una svolta significativa nel 1977 grazie alla tecnica di sequenziamento genetico proposta dal biochimico Frederick Sanger, conosciuta ora come metodo Sanger.

Tale procedimento determina l'ordine delle basi in un filamento breve di DNA utilizzando la tecnica dell'elettroforesi e il supporto di alcuni marcatori radioattivi. Questo metodo è considerato rivoluzionario ed è stato utilizzato come capostipite del sequenziamento del genoma umano per più di 40 anni.

Nel 1984 il Department of Energy (DOE) e il National Institutes of Health (NIH) degli Stati Uniti avviarono un programma per lo sviluppo di tecnologie

e metodi per il sequenziamento genetico e la mappatura del codice genetico umano: lo Human Genome Project(HGP). L'obiettivo essenziale del programma fu la decodifica del DNA umano che comprese principalemente la determinazione delle sequenze, cioè l'ordine delle basi, e la mappatura dei geni, ovvero la localizzazione dei geni per la maggior parte dei cromosomi e le connessioni con le caratteristiche fisiche e germinali. Oltre a ciò, il progetto cercò di analizzare anche i genomi di esseri viventi considerati più semplici come ad esempio le mosche.

Molte tecniche per la trattazione del materiale genetico parteciparono alla ricerca, ma il primo procedimento attuato dall'HGP fu il BAC, Bacterial Artificial Chromosome, che sfruttava la clonazione apportata da certi batteri per ottenere un numero elevato di filamenti. Quest'ultimi subivano un ulteriore frammentazione, per poi essere sequenziati con il metodo Sanger ed infine essere riuniti per formare l'unica stringa di DNA.

Dopo oltre un decennio, nell'Aprile del 2003, l'HGP si concluse con successo, essendo stato decodificato il 99% del genoma umano e visto che gli studi furono allargati prematuramente ad altri interessi soprattutto nelle ricerche sulle malattie.

Da quel momento in poi, il termine comune affiliato alle nuove tecniche sul genoma umano è Next-Generation Sequencing(NGS), di cui fanno parte gli algoritmi coinvolti in questa presentazione.

### NGS

La necessità di garantire una maggior rapidità, e anche di ridurre le spese, ha aperto un altro capitolo nella ricerca sul genoma umano: il Next-Generation Sequecing [1]. Tale necessità deriva dal fatto che il metodo Sanger, per via della precisione necessaria, ha richiesto circa tredici anni per ottenere l'intero genoma umano con un investimento di circa $ 2.7 miliardi.

Questa nuova generazione di tecnologie, che sfrutta la parallelizzazione di processi su scale microscopiche [2], concede ai ricercatori strumenti estremamente più veloci, tali da stimare il genoma di un individuo in meno di un giorno, e molto meno costosi. In aggiunta, i numerosi metodi sviluppati all'interno del NGS sono fortemente radicati sulla teoria dei network. Questa branca della fisica e della matematica ha subito nell'ultimo ventennio un notevole progresso e ha acquistato un significato oramai fondamentale nella ricerca medica e biologica.

Un altro fattore importante che supporta lo sviluppo di tecnologie più veloci è l'esistenza del genoma di riferimento prodotto dal HGP, che permette in certi casi il confronto solamente frazioni di DNA senza dover estrarre l'intero codice genetico.

Un ulteriore contributo è dato anche dalla contemporanea crescita delle aree che cooperano con la biologia, tra cui l'evoluzione delle tecniche di computazione.

Le tecniche della NGS sono più convenienti del metodo Sanger, dati i tempi richiesti, ma è ancora da trovare il giusto compromesso tra accuratezza dei risultati e diminuzione delle tempistiche e delle spese economiche. Infatti, nello studio di sequenze a piccola scala la tecnologia Sanger resta tuttora una tra le più affidabili, mentre su letture ad ordini superiori le applicazioni del NGS possono vantare la miglior efficacia.

Pur adoperando diverse tecnologie, gli algoritmi per NGS prevedono solitamente gli stessi passaggi, tra cui l'allineamento delle letture con un riferimento e l'investigazione sulle possibili variazioni.

La difficoltà nello scegliere quale usare tra le diverse opzioni disponibili può essere superata confrontando i risultati finali dei procedimenti, in relazione agli scopi preposti. Tali risultati includono la qualità delle informazioni ricavate, le tempistiche previste e altre caratteristiche come l'assemblaggio e gli allineamenti.

Negli ultimi anni si sono aggiunti alle ricerche del NGS due software chiamati GATK e MuTect, i cui metodi hanno condotto allo sviluppo dell'algoritmo coinvolto nel progetto presentato in questa tesi: il GATK-LOD$_{\mathrm{n}}$.

### 1.1.2 Il metodo GATK-LOD$_{\mathrm{n}}$ e le sue radici

L'ideazione di questo algoritmo é dovuta ad un gruppo di ricercatori del Dipartimento di Fisica e Astronomia dell'Università di Bologna nell'ambito dello studio sulla scoperta di polimorfismi somatici del singolo nucleotide nel sequenziamento dell'esoma. Precisamente, questo genere di polimorfismo è denonimato con la sigla SNP, single nucleotide polymorphism, e indica quelle variazioni nei singoli nucleotidi che si verificano con frequenza significativa in una specifica posizione del genoma. In particolare, l'esoma comprende quelle regioni del genoma in cui sono codificate le istruzioni per l'RNA e per la sintesi delle proteine.

Le mutazioni genetiche si distinguono in due tipologie: germinali e somatiche. Le prime identificano le variazioni che avvengono nello zigote sono condivise in tutte le cellule dell'organismo. Le seconde, invece, sono mutazioni avvenute successivamente allo sviluppo e possono presentarsi a livello di singola cellula o tessuti. L'algoritmo GATK-LOD$_{\mathrm{n}}$ si focalizza sulle mutazioni somatiche perchè esse hanno un ruolo chiave nella progressione della malattia e nella resistenza alla chemioterapia.

L'interesse nel proporre questo metodo nasce dal desiderio di poter predisporre di uno strumento che non si aggiunga al gruppo di software già

esistenti, bensì che ottimizzi e potenzi alcuni tra questi. É per questo che il team di studiosi ha considerato due applicazioni standard, GATK e MuTect, e li ha composti in modo da migliorare i prodotti finali di entrambi. Infatti alla completa esecuzione di GATK è stato applicato una componente di MuTect, ovvero un classificatore Bayesiano conosciuto come $LOD_n$, il cui scopo è verificare un ulteriore volta i risultati ottenuti. I passaggi previsti dall'algoritmo sono vari e di seguito saranno esposti coloro ritenuti più rilevanti.

Dopo aver raccolto i campioni normali e quelli per alcune specie di tumori attraverso specifiche metodologie sperimentali, essi sono stati sottoposti ad un controllo di qualità tale da rimuovere le letture considerate a bassa confidenza. A questo punto, sono state applicati in successione i tools BWA-MEM e Picard, dove il primo allinea le reads e il secondo le ordina, indicizza e in più ne marca i duplicati. Una volta completata una nuova fase di riallineamento locale e di ricalibrazione sulla qualità delle letture, grazie all'utilizzo di alcuni strumenti del Genome Analysis Toolkit, sono stati eseguiti GATK e MuTect per la ricerca delle varianti sui singoli nucleotidi(SNV).

La differenza procedurale tra queste due applicazioni è che, mentre MuTect ritrova le mutazioni contemporaneamente tra i campioni normali e tumorali, GATK le chiama indipendentemente. Si sottolinea che i due metodi scovano varianti che non sono condivise da entrambi, indicando la natura incompleta dei sistemi utilizzati. Un ulteriore distinzione tra GATK e MuTect è data dal tipo di risultati raccolti poichè, se il primo è dotato di una maggiore sensibilità alle mutazioni, dalle 3 alle 20 volte superiore al secondo, quest'ultimo possiede una maggiore specificità degli SNVs. Avendo GATK un elevato numero di falsi positivi nella chiamata alle varianti, è stato aggiunto in fondo all'algoritmo il classificatore Bayesiano di MuTect per cercare di ridurre questo errore. Il compito del $LOD_n$ consiste nel calcolare il rapporto tra i due seguenti eventi probabilistici. Il primo è il caso in cui le mutazioni nel campione normale siano dovute a rumori di fondo e quindi in realtà non esistano. Il secondo, invece, considera il caso in cui la mutazione esista davvero nel campione normale e sia dovuta ad una variante germinale eterozigote. A questo punto, se il rapporto tra le probabilità (il Log Odds, da cui la sigla LOD) eccede un valore di soglia fissato, il classificatore definisce la variante come somatica.

Nella ricerca sono stati confrontati i prodotti finali dei tre algoritmi ed è stato verificato che l'uso di GATK-$LOD_n$ riduce notevolmente il numero di chiamate degli SNVs di GATK, mantenendo una sensibilità nettamente superiore a MuTect. Questa riduzione è dovuta all'eliminazione di un sostanziale numero di falsi positivi, la cui entità dipende dalla tipologia di tumore esaminato.

Il miglioramento dei precedenti metodi, però, non si limita solo alla filtrazione dei falsi positivi ma anche al mantenimento della sensibilità di GATK e ad un aumento della specificità. Quest'ultimo è indicato dal fatto che GATK-LOD$_n$ abbia presentato frequenze di validità superiori e un miglior PPV(Positive Predictive Value) rispetto a GATK su vari numeri di reads di VAF(Variant Allelic Frequency). Le metodologie adoperate per confrontare la sensibilità e la specificità non sono state ritenuti utili allo scopo della presentazione e per questo non verranno trattate.

A posteriori delle indagini sperimentali, GATK-LOD$_n$ si è rivelato uno strumento utile ad allargare le capacità di GATK e a individuare varianti non trovate da MuTect senza dover rinunciare alla specificità e sensibilità.

Visti i risultati positivi ricavati dall'algoritmo, l'articolo originale sostiene che un metodo di questo tipo possa aiutare a definire con maggior dettaglio le mutazioni somatiche di genomi cancerogeni, favorendo le valutazioni mediche e gli approcci sui percorsi di cura.

Per fornire una conoscenza più approndita delle operazioni che avvengono nel sequenziamento, saranno esposti i passaggi fondamentali alla base delle applicazioni utilizzate nel metodo GATK-LOD$_n$: GATK e MuTect.

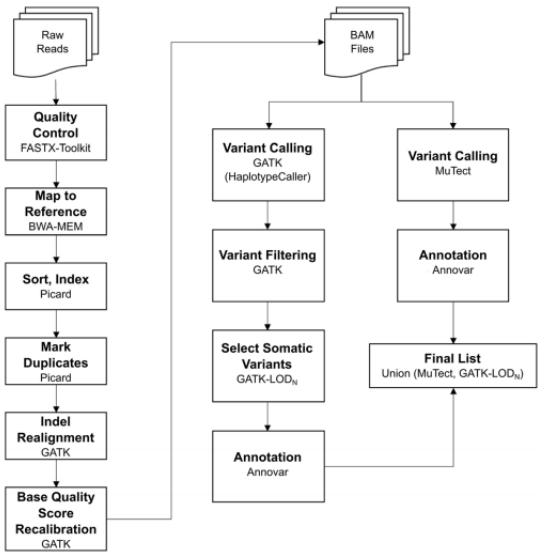



Figura 1.1: Pipeline di GATK-LOD$_n$(presa da [3]).

### GATK

Il Genome Analysis Toolkit è un framework di programmazione creato da un gruppo di ricercatori di Boston, Massacchussets, assieme al Broad Institute di Harvard e l'MIT in Cambridge, Massacchussets, per facilitare lo sviluppo

di programmi che analizzano l'enorme mole generati dal NGS, Next Generation DNA Sequencing [4]. Infatti, l'utilizzo di questo sistema permette di sviluppare tools più solidi e performanti per il sequenziamento genetico.

La mancanza di strumenti flessibili e sofisticati dediti alla manipolazione dei dati di sequenziamento in maniera programmatica ha portato alla creazione di GATK. Infatti, la maggior parte dei software che supportano l'analisi del DNA concedono alte prestazioni sono nella specifica area di interesse, senza mantenerle su differenti ambiti. Questo deficit e l'emergere di un formato specifico per i prodotti del sequenziamento (SAM), hanno dato l'opportunità di ideare un software per la semplificazione delle analisi sui set di dati.

L'architettura di base per GATK è il MapReduce, il cui funzionamento implica la separazione delle computazioni in due passaggi. Nel primo, l'intero problema è suddiviso in tanti elementi discreti indipendenti, i quali sono correlati alla funzione Map; nel secondo step, l'operatore Reduce riunisce gli esiti di Map in un unico risultato finale. Siccome solo in certi casi, come la ricerca degli SNPs, vi è un adattamento naturale del sistema, GATK è costruito su traversals e walkers. Le traversals sono schemi che provvedono alla preparazione e divisione dei dati; mentre le walkers consistono nei differenti moduli di analisi che computano i dati provenienti dalle prime. Anche se GATK ha un numero ridotto di traversals, questo basta per soddisfare le esigenze della maggior parte della comunità di ricerca. I due trasversal standard sono il "by each sequencer read" e il "by every read covering single base position in a genome", il cui uso è sfruttato per operazioni standard come la chiamata agli SNPs. Il meccanismo di questi schemi non è riportato in quanto non rilevante per il lavoro di tesi.

Uno dei punti di forza dell'algoritmo è la capacità di gestire l'enorme quantità di materiale ricavato dal sequenziamento. In particolare, GATK divide il tutto in pezzi chiamati "shard" che, al contrario della maggior parte dei sistemi di suddivisione, sono di una dimensione di poche migliaia di basi. In questo modo non sono limitate le capacità della memoria e le prestazioni nel caso di parallelismo. Questi shards contengono tutte le informazioni della regione genomica associata e sono trasmesse al trasversal prescelto.

Altre caratteristiche di GATK sono la possibilità di selezionare solo certe regioni del genoma, la parallelizzazione delle azioni da svolgere, l'organizzazione dei files di input grazie ad un'operazione di merging e la presenza di un walker relativo alla depth of coverage(DoC).

Il metodo stima il genotipo più probabile attraverso un semplice algoritmo Bayesiano, che ha funzione sia di punto di partenza per sviluppare nuovi classificatori; che di mettere in luce le capacità di parallelizzazione e di otti-

mizzazione della memoria disposte da GATK. É importante sottolineare che è proprio la semplicità dell'operatore la causa di molti falsi positivi chiamati.

Il Genome Analysis Toolkit è quindi un framework che, grazie al suo nuovo approccio ai big data e alla piena libertà di sviluppo, fornisce strumenti importanti per l'elaborazione di algoritmi più specifici come il GATK-LOD$_{\mathrm{n}}$.

**MuTect**

Il metodo MuTect è un algoritmo per la rilevazione delle mutazioni genomiche, che è stato creato per superare le scarse prestazioni dei meccanismi fino ad allora presenti [5]. Infatti, quest'ultimi fornivano livelli di sensibilità e specificità considerati insoddisfacenti per una sufficiente comprensione delle anomalie.

Il sistema si focalizza soprattutto sull'individuazione delle varianti a bassa frazione allelica, ovvero quelle regioni del DNA che originano e nutrono il tumore. Queste frazioni sono tanto importanti quanto difficili da scovare poichè, mentre il meccanismo al loro interno determina il tipo di alterazione genomica, sia le regioni in cui si manifestano che la frequenza di occorrenza sono basse.

La conoscenza di questi tratti specifici favorisce lo studio sulle evoluzioni delle forme cancerogene e aiuta a proporre terapie di cura più affidabili.

L'esistenza di metodi a scarsa sensibilità e specificità ha quindi indotto un team di ricercatori dell'università di Boston, Massachussetts, assieme al Broad Insitute di Harvard e all'MIT in Cambridge, Massacchussets, a sviluppare il tool MuTect. Questo strumento si è dimostrato sensibile e specifico nella scoperta degli eventi a bassa frequenza allelica, mantenendo alte prestazioni di specificità anche a frequenza superiori.

Nessuno dei modelli precedenti supporta tutti gli errori dei processi di sequenziamento ed è per questo che MuTect sfrutta due approcci di benchmarking per migliorare la performance: il downsampling e i 'virtual tumors'. Il primo misura la sensibilità con cui le mutazioni vengono chiamate, grazie a subset di dati con mutazioni già riconosciute. Questo approccio è però limitato da alcuni aspetti che comprendono: il basso numero di eventi verificati, la sovrastima della sensibilità e l'impossibilità alla misurazione della specificità. A causa dei limiti del downsampling è presente anche un secondo approccio, 'virtual tumors', che genera dei tumori virtuali conosciuti in ogni dettaglio. I questa maniera i due metodi sono complementari e mentre il primo usa dati reali ma è limitato, il secondo è libero ma consuma materiale virtuale generato. La sintesi dei due approcci permette di ricavare valori più veritieri della sensibilità, misurare la specificità e colmare la maggior parte delle lacune derivate dal downsampling.

Previo allineamento ed esecuzione dei processi standard preanalisi, MuTect riceve i dati sequenziati sia dei campioni normali che dei cancerogeni, per poi eseguire quattro operazioni principali: la rimozione di dati a bassa qualità, la ricerca delle varianti, il filtraggio dei falsi positivi e la classificazione delle varianti. La ricerca delle varianti consiste nell'applicazione di un primo metodo bayesiano, detto $LOD_T$, che adopera il rapporto tra due eventi probabilistici per determinare se è presente un variante. Siccome il calcolo è condizionato da errori di sequenziamento, prima di ricavare la probabilità del variante è necessario applicare una serie di filtri che ne elimini la maggior parte, evitando così la sottostima dei falsi positivi. Successivamente, si utilizza il secondo classificatore bayesiano $LOD_n$, implementato nel metodo GATK-$LOD_n$, per definire se il variante è somatico, germinale o indeterminato.

Gli esperimenti di verifica sulla sensibilità hanno evidenziato come MuTect sia uno strumento ad alto rilevamento soprattutto nelle mutazioni a frazioni alleliche basse. Riguardo alla specificità sono due le fonti principali di falsi positivi: l'eccessiva chiamata a varianti, dovuta ad errori di sequenziamento, e la scarsa individuazione di eventi germinali, causato dalle insufficienti letture sul campione normale. La gestione di tali errori è risolta con il miglior compromesso verificato tra sensibilità e specificità, che risulta nella scelta di mantenere alta la seconda a discapito della prima.

Complessivamente MuTect è un metodo che valorizza il compromesso tra il grado di rilevazione delle varianti e la loro corretta classificazione, producendo risultati affidabili. In aggiunta, riporta sostanziali miglioramenti sulle analisi delle mutazioni a bassa frequenza allelica, la cui importanza è essenziale per le future ricerche biomediche.

### 1.1.3 Il ruolo della fisica nella ricerca biomedica

Nella prima metà del Novecento, dopo che l'introduzione della relatività di Einstein e la nascita della meccanica quantistica avevano ribaltato il pensiero della comunità scientifica, numerosi fisici si interessarono a problemi di biologia. Il contributo fornito ad una così diversa area era sia di tipo matematico, dove la necessità di utilizzare equazioni e formule continuava ad aumentare nel corso degli anni, che di tipo teorica, ove l'uso di modelli già presenti nella fisica erano utili ad una comprensione più approfondita dei quesiti di biologia. Il libro "What is life?" del fisico austriaco Erwin Schroedinger promosse il ruolo della fisica negli studi sull'ereditarietà a tal punto che gli scienziati Watson e Crick, che svelarono la struttura del DNA, lo accreditarono nelle loro pubblicazioni [6]. In seguito, numerosi fisici hanno contributo alla nascita della biologia molecolare e questa branca ha continuato a beneficiare

della fisica sia per interpretare i problemi sotto un'ottica diversa, che per la confidenza che i fisici hanno riguardo alla modellizzazione dei sistemi complessi. Difatti, la richezza della fisica nel saper gestire tali sistemi, l'elasticità nell'ampliare le proprie conoscenze a differenti campi di studio, insieme ad un approccio critico ma propositivo, si è rivelata un ingriedente necessario per il progresso della materia [7].

Al passare del tempo e con il crescere delle ricerche, soprattutto nel campo biomedico, la presenza della fisica ha acquistato importanza sia nel ramo sperimentale che in quello teorico. Nel primo, difatti, per anni le tecniche della cristallografia e della risonanza magnetica sono stati essenziali per i biologi molecolari, e, tuttora, le innovative tecniche della NGS concentrano gli sforzi su scovare quelle differenze proprietà fisiche che permettono di stabilire più velocemente i nucleotidi [8]. Nel secondo, invece, la fisica coopera con i bioinformatici sia nell'individuazione di modelli efficaci con cui affrontare i big data provenienti dai laboratori, che per l'analisi delle sequenze di DNA, sfruttando metodi provenienti esattamente dalla fisica statistica.

Nello specifico, la maggioranza dei modelli utilizzati e studiati sono basati sulla teoria dei network, la quale è stata ha acquisito uno spessore essenziale negli studi di biologia e di medicina.

### Teoria dei network

Per network si intende un modello che mira a rappresentare un sistema complesso, cercando di modellizzare le caratteristiche collettive. Un network è composto da elementi chiamati nodi, i quali sono collegati tramite percorsi, detti path, e tra cui sono presenti relazioni, denominate link. La teoria dei network è stata adottata e sviluppata per spiegare quelle leggi della natura di carattere generico e che hanno origine da manifestazioni stocastiche. Gli strumenti matematici adoperati in questa teoria sono prevalentemente le matrici e i grafi, le cui proprietà garantiscono la massima espressione di connettività e di evoluzione che contraddinguono i sistemi complessi.

I sistemi presenti in biologia sono generalmente un insieme di oggetti fortemente interagenti di cui si conoscono le proprietà; come possono essere ad esempio le reti geniche o il sistema nervoso umano. Grazie alle tecniche provenienti dalla teoria dei network non è più indispensabile ricostruire il codice genetico puntualmente, dato che risulta possibile ricombinare tante stringhe casuali dello stesso filamento di DNA(shotgun sequencing). Questa novità è ciò che contraddistingue le nuove tecniche del NGS dal metodo Sanger e che permette a queste di ricavare i genomi di un individuo in tempi decisamente più ristretti.

La ricostruzione del genoma avviene, in particolare, a partire da tutte le possibili sovrapposizioni dei frammenti sperimentali estratti. La scelta riguardo al ruolo dei frammenti e delle sovrapposizioni, ovvero stabilire chi è nodo e chi è link, è ciò che determina il modello dei network da perseguire. Nel caso in cui i primi fossero i nodi e le seconde i link condurrebbe all'utilizzo di un path Hamiltoniano, cioè un percorso dove tutti i nodi vengono usati una volta sola. All'opposto si avrebbe un path Euleriano, dove ogni link deve essere processato una singola volta e dove il problema è identificato dal grafo di De Brujin. La differenza più netta tra i due path è la crescita di complessità, visto che il primo ha andamento non polinomiale mentre il secondo lineare, ed è da ciò che si valutano le capacità computazionali.

In conclusione, la teoria dei network, unita all'utilizzo di macchine server performanti, ha accelerato lo sviluppo di meccanismi sempre più raffinati per la ricostruzione del DNA, superando l'esaminazione prolungata in laboratorio con l'applicazione di congetture di fisica statistica.

## 1.2 Lo strumento di sviluppo: Snakemake

Snakemake è un sistema di gestione dei flussi di lavoro che semplifica l'esecuzione di algoritmi particolarmente complessi grazie all'utilizzo di un ambiente di sviluppo nitido ed intuitivo [9]. In più, questo software è specializzato nella scalabilità dei lavori, da singoli core fino all'uso di cluster, le cui transizioni non implicano pesanti modifiche al procedimento del sistema.

Questo programma è basato sul linguaggio di programmazione Python [10] e la sua formazione è fortemente influenzata dal noto tool Make del sistema operativo Linux. Ciò significa che Snakemake è modellato su una strategia di tipo *pull* proprio come Make. In un procedimento che segue la strategia *pull*, ogni lavoro coinvolto specifica ciò di cui ha bisogno per essere eseguito ed il programma esegue automaticamente altri lavoro che posso soddisfare queste richieste; la sequenza di esecuzione è quindi determinata a partire dal lavoro finale. Al contrario, in un procedimento che segua la strategia *push* ogni lavoro determina quale sarà il lavoro eseguito successivamente. La strategia *pull* richiede una maggior pianificazione per la scrittura ma fornisce diversi vantaggi fra cui una maggior semplicità di parallelizzazione e la possibilità di riprendere l'esecuzione del procedimento dopo un fallimento critico senza dover ripetere tutti i passi precedenti.

Il contenitore del codice Python in Snakemake è chiamato di default *Snakefile* e l'ordine di esecuzione predefinito da linea di comando è:

```
$ snakemake
```

Il sistema è strutturato, come Make, da un insieme di regole che rappresentano i compiti da svolgere nell'algoritmo, dove ognuna di esse contiene le tre informazioni fondamentali: input, output e azione.

```
rule 'nome':
        input:

        output:

        shell/run/script:
```

Come si può vedere dal codice riportato, l'operazione avviene etichettando la regola con un certo 'nome' e inserendone all'interno le keyword *input*, *output* ed una tra *shell*, *run* e *script*. Rispettivamente le tre chiavi implicano l'utilizzo di comandi da terminale, l'esecuzione di un codice Python e l'avvio di uno script esterno. La dichiarazione dei file di input e di output esprime le condizioni iniziali per la regola e il risultato atteso, così permettendo al programma di riconoscere le relative dipendenze e stabilire l'ordine di successione dei singoli lavori. Per facilitare la comprensione del codice, è possibile creare, grazie al comando *dot* della libreria Graphviz, un diagramma che schematizzi la sequenza delle regole. Infatti questa sequenza, che ha il nome di *DAG* (Directed Acyclic Graph), è visualizzata da *dot* in una struttura ramificata, in cui i lavori sono rappresentati da nodi e le dipendenze sono semplicemente descritte da linee congiungenti. Questa schematizzazione consente anche di identificare quali dei lavori sono parallelizzabili e quali, invece, devono mantenere una prestabilita sequenzialità. Ciò è possibile proprio dal fatto che il comando *dot* rappresenta la successione delle regole solo e unicamente in base alle dipendenze reciproche, mantenendo quindi, ad esempio, la ripetizione di alcune regole sullo stesso livello di operatività. Questo sistema di riconoscimento dei lavori potenzialmente simultanei da un contributo importante per impostare meccanismi di parallelizzazione.

A proposito della gestione tecnica dei lavori, Snakemake ha un certo numero di proprietà che consentono di selezionare le caratteristiche computazionali desiderate. Due tra le funzionalità più rilevanti sono *resources* e *cores* che stabiliscono, da linea di comando, rispettivamente quali risorse delle macchine sono a disposizione e quanti core sono fruibili. Le risorse possono includere ad esempio delle direzioni sulla gpu o sulla memoria accessibile (opzione *mem*), mentre il numero di cores è essenziale per gestire i threads.

I threads sono fondamentali per un'esecuzione simultanea e il loro utilizzo avviene inserendo l'attributo *threads* nel dominio di una regola. L'assegnazione di un determinato numero di threads è comunque influenzato da un'eventuale opzione sui cores, dato che il numero di threads non può eccedere il

numero di cores utilizzabili. In particolare, oltre ai threads, è possibile specificare singolarmente per ogni regola anche le risorse disponibili, aggiungendo la voce *resources* nel dominio.

Altre due proprietà di Snakemake sono la portabilità e un innovativo meccanismo di inferenza. La prima manifesta le poche dipendenze di installazione, dato che è in generale sufficiente dotarsi di Python; mentre la seconda rappresenta un moderno supporto all'inferenza per nome che si compie grazie a wildcards nominate nelle regole.

Le wildcards consistono in nomi che agiscono come parametri e che servono ad automatizzare le operazioni di riconoscimento delle dipendenze tra regole. In dettaglio, quando una variabile è associata ad una wildcard, è naturale che questa sia associata a più valori e che, quindi, il nome della variabile sia semplicemente una chiave. In presenza di una wildcard nell'input di una regola, i valori in essa contenuti sono ricercati negli output delle altre regole e, dopo aver tracciato le dipendenze, tale regola è eseguita una volta per ogni valore attribuito alla chiave. Così agendo, le ripetizioni delle regole su diversi valori della wildcard sono sullo stesso piano di esecuzione, favorendo la parallelizzazione dei lavori.

Nel corso di questa tesi sono stati implementati alla pipeline altri attributi forniti da Snakemake. Infatti, eccetto le chiavi di base, esistono diverse funzionalità che arrischiscono l'impostazione e la realizzazione delle regole, tra cui *params*, *threads*, *benchmark* e *conda*. I primi due contengono rispettivamente i parametri indispensabili introdotti nell'azione e il numero di threads su cui l'azione può essere eseguita. Quest'ultima proprietà richiama ad uno tra gli aspetti più importanti che hanno determinato la scelta di Snakemake, ovvero la capacità di gestire l'esecuzione della pipeline sulle risorse tecniche dei nodi di calcolo adoperati. Le istruzioni sul numero di core da utilizzare, il numero di threads e la quantità di memoria da mettere a disposizione consentono di ricavare le configurazioni più efficienti, il tutto a beneficio dello sviluppo di metodi più potenti e ottimizzati.

A differenza delle prime due direttive, *benchmark* e *conda* richiedono una trattatazione più ampia.

In seguito alla spiegazioni di questi, è necessario dedicare un breve accenno ai file di configurazione spesso affiancati agli Snakefile.

### benchmark

Quando è utilizzata la direttiva *benchmark* in una regola, Snakemake trascrive su un file di testo i dettagli tecnici dell'operazione svolta.

Per primi sono riportati il tempo impiegato dal nodo per completare la regola sia in secondi che in ore, minuti e secondi.

A seguire, dal terzo al sesto sono mostrate le informazioni sull'uso della memoria. In particolare:

- max_rss è la massima memoria fisica *non swapped*, che il processo usa(Resident Set Size);
- max_vms è la massima quantità totale di memoria virtuale utilizzata(Virtual Memory Size);
- max_uss è il massimo di memoria affidata unicamente al singolo lavoro, che esso impiega(Unique Set Size);
- max_pss è il massimo della quantità condivisa tra tutti i processi, che la regola sfrutta(Proportional Set Size).

Riguardo agli ultimi tre dettagli, sono presenti io_in e io_out che identificano le caratteristiche di input e output del processo; e mean_load che descrive il carico medio sulla CPU.

### Conda

Conda è una piattaforma per la gestione di svariati pacchetti ed è un pratico amministratore degli ambienti di elaborazione. La cooperazione tra Snakemake e Conda, che avviene, come mostrato nel codice sottostante, inserendo la direttiva *conda* nel dominio, favorisce un uso più flessibile degli ambienti.

```
conda:
        "path/to/directory/config_file.yaml"
```

Difatti, prima che lo Snakefile sia eseguito, Snakemake riconosce la voce *conda* nella regola e richiama Conda per la creazione o attivazione dell'ambiente su cui essa sarà completata. Le informazioni sull'ambiente da formare, nel caso esso debba essere creato, sono procurate da un file di configurazione indicato nel dominio ed è proprio questo che consente a Conda di generare l'ambiente e dotarlo dei requisiti richiesti. Una volta che Conda ha terminato la creazione, l'ambiente è attivato e comincia l'esecuzione. Chiaramente, questa fase di creazione avviene alla prima richiesta di un ambiente con determinate caratteristiche, così da procedere semplicemente con l'attivazione nelle successive computazioni.

Una tale opzione non costringe altri utilizzatori ad impostare manualmente l'ambiente principale come voluto dallo Snakefile, dato che ne sarà creato un apposito per la regola, e ciò ne alleggerisce l'utilizzo. In aggiunta, questo meccanismo rende plastica la realizzazione del codice, vista la possibilità di dotare ogni regola di un proprio ambiente.

### 1.2.1 I file di configurazione in Snakemake

I file di configurazione sono oggetti, solitamente in formato yaml, dedicati alle istruzioni sui parametri contenuti nei codici principali. Il ruolo di tali file è stabilire il valore delle chiavi che rappresentano i parametri. Ad esempio, nel caso di Snakemake, un tipico file di configurazione ha forma: `chiave: 'valore'`. Il valore può essere un numero, una parola, un percorso o un file, ed esso rimane costante se non modificato direttamente nel codice sorgente o da linea di comando. Nell'ambito di Snakemake, il file di configurazione deve essere citato inizialmente con la linea `configfile:'config_file.yaml'` e l'associazione tra chiavi e parametri è conseguita inizializzando una variabile secondo la seguente modalità.

```
parametro = configfile['chiave']
```

I valori dei parametri possono essere modificati da terminale nel comando di avvio dello Snakefile, grazie all'argomento *–config* seguito da una nuova inizializzazione, ad esempio `chiave='nuovo_valore'`.

Il file di configurazione che si indica nella direttiva *conda* ha, invece, una composizione differente, come mostrato nel codice riportato.

```
channels:
  - esempio_canale
dependencies:
 -  esempio_dipendenza
```

Il dominio *channels* determina su quale canale Conda deve lavorare, mentre *dependencies* specifica quali pacchetti devono essere installati nell'ambiente. In questo modo, dopo che Conda è chiamato da Snakemake per l'attivazione del particolare ambiente, esso controlla se ne è già presente uno con tali proprietà e, se esistente, lo attiva o, al contrario, prima lo istanzia.

Il contenuto dei file di configurazione è, in conclusione, determinante per il corretto, oltre che miglior, funzionamento del sistema.

## 1.3 Le macchine low power e i nodi utilizzati

I gruppi di ricerca impegnati per l'analisi dei big data in campo biomedico, hanno a disposizione server sempre più sofisticati e veloci. Purtroppo le migliori prestazioni e funzionalità hanno comportato costi e consumi elettrici sempre piu' elevati (anche se negli ultimi anni sono state introdotte anche per i server tecnologie avanzate per il power-saving e il power-monitoring). D'altra parte processori low-power normalmente utilizzati in ambito low-end o embedded/mobile stanno acquisendo un ruolo sempre piu' importante.

Si è quindi creato nel mondo computazionale scientifico un inedito scenario: accanto ai tradizionali server high-end, costosi ed energivori, si iniziano a intravedere soluzioni ibride con processori low-power o acceleratori che stanno progressivamente riducendo il divario prestazionale rispetto ai server high-end.

I laboratori di calcolo e i data center scientifici e commerciali seguono con interesse l'evoluzione delle tecnologie di calcolo low-power per un duplice motivo: costo elevato delle macchine server attuali e costi dell'energia elettrica per il consumo dei server e raffreddamento. Il costo medio per macchine server high-end raggiungono ormai cifre molto elevate, dell'ordine delle migliaia di euro. I costi energetici per il mantenimento in servizio dei server high-end e quello per il raffreddamento hanno raggiunto ormai livelli proibitivi. Un esempio è il data center di medie dimensioni a Bologna del INFN-CNAF (Centro Nazionale delle Tecnologie Informatiche dell'INFN) dove sono ospitati migliaia di server il cui costo annuale d'esercizio per l'energia elettrica è dell'ordine delle centinaia di migliaia di euro.

Un altro fattore penalizzante è dato dalla mancanza di scalabilità e flessibilità per aggiornare l'hardware dei server, visto il notevole investimento per l'acquisto giustificato da un utilizzo almeno triennale dell'hardware prima di essere sostituito. D'altro canto schedine low-power, grazie a costi particolarmente contenuti, consentono una notevole flessibilità nell'acquisto di nuovo hardware non appena viene rilasciato dal produttore.

La tecnologia di calcolo low-power utilizza prevalentemente Systems-on-Chip (SoCs) spesso di derivazione embedded o mobile che sono stati disegnati per garantire le massime prestazioni computazionali coi minimi consumi elettrici. L'ottimo rapporto prestazioni/consumi dei SoC è dovuto all'impegno dei costruttori per soddisfare la domanda di schede e dispositivi sempre più sensibili alle performance e alla riduzione dei consumi elettrici da parte dell'emergente industria mobile ed embedded. Perciò, visti i notevoli miglioramenti degli ultimi anni, i SoC stanno seriamente diventando un'interessante alternativa per le applicazioni scientifiche senza sacrificare troppo performance e funzionalità dei server tradizionali. Inoltre la cooperazione fra diversi nodi low-power, configurati in cluster, potrebbe ambire confrontarsi con i cluster tradizionali di server high-end.

Questa tesi descrive una ricerca introduttiva per valutare prestazioni e funzionalità su hardware low-power di un'applicazione scientifica che gira normalmente su macchine server.

Le macchine low-power coinvolte in questa tesi sono state scelte con caratteristiche tecniche differenti pur mantenendo la stessa architettura X86_64. In più, è stato considerato anche un server tradizionale X86_64 per poter comparare le diverse esecuzioni.

Le macchine low-power sono state scelte, installate e configurate grazie alla collaborazione con il CNAF (Centro Nazionale per la Ricerca e lo Sviluppo sulle Tecnologie per l'Informazione e la Comunicazione) dell'Istituto Nazionale di Fisica Nucleare

La tabella della 1.2 mostra le macchine utilizzate per i test.

(a) *Cluster Xeon D-1540 e Atom C2750.*

(b) *Cluster contenente bio8.*

Figura 1.2

I dettagli sui nodi dei cluster utilizzati per le computazioni sono esposti nelle tabelle sottostanti 1.1 e 1.2, che includono sia i tre nodi low power che un nodo di un server tradizionale (bio8). In particolare, tutti i nodi montano processori con architettura X86_64 su sistema operativo Linux Debian 9.

| *Nodo* | *CPU* | *Memory* | *Storage* | *Costo** | *Consumo** |
|---|---|---|---|---|---|
| *xeond* | 1x Xeon D-1540 | $16\,GB$ | $8\,TB(HDD)$ | €1000 | $60\,W$ |
| *avoton* | 1x Atom C2750 | $16\,GB$ | $5\,TB(HDD)$ | €600 | $30\,W$ |
| *n*3700 | 1x Pentium N3700 | $8\,GB$ | $0.5\,TB(SSD)$ | €130 | $8\,W$ |
| *bio*8 | 2x Xeon E5-2620v4 | $128\,GB$ | $2\,TB(HDD)$ | €10000 | $180\,W$ |

* I valori di costo e consumo energetico sono stimati.

Tabella 1.1: Caratteristiche dei nodi.

| $CPU$ | $Microarchitecture(Platform)/litho$ | $Freq(GHz)$ | $Cores$ | $Cache$ | $TDP$ |
|---|---|---|---|---|---|
| Xeon D-1540 | $Broadwell/14nm$ | 2.0(2.60) | 8(16) | $12\,MB$ | $45\,W$ |
| Atom C2750 | $Silvermont(Avoton)/22nm$ | 2.40(2.60) | 8 | $4\,MB$ | $25\,W$ |
| Pentium N3700 | $Airmont(Braswell)/14nm$ | 1.60(2.40) | 4 | $2\,MB$ | $6\,W$ |
| Xeon E5-2620v4 | $Broadwell - EP/14nm$ | 2.10(3.00) | 8(16) | $20\,MB$ | $85\,W$ |

Tabella 1.2: Caratteristiche delle CPU.

Si sottolinea che, in realtà, il nodo xeond ha una cpu con TDP pari a 45 W e quindi è considerato al limite come macchina low power. Inoltre il tipo di processore è classificato come tipologia server, esattamente come quello di bio8.

# Capitolo 2

# Materiali e Metodi

Questo capitolo ha lo scopo di presentare il procedimento seguito per valutare l'ottimizzazione di una pipeline per l'analisi computazionale biofisica su nodi di cluster a basso consumo energetico.

In primo luogo sarà spiegato la componente principale creata per l'elaborazione, che implementa una parte del metodo GATK-LOD$_{\text{n}}$, e ne saranno approfonditi i singoli passaggi.

A seguire, saranno specificate quali computazioni sono state svolte, spiegando i motivi che hanno spinto a considerare talune, e il tipo di operazioni statistiche effettuate per modellare il prodotto delle analisi.

## 2.1 Struttura delle simulazioni

Il procedimento seguito dal sistema è suddivisibile in tre fasi: installazione, esecuzione e configurazione.

La prima di queste, la fase di installazione, evidenzia quali sono i fattori che permettono all'esecuzione di avvenire senza problemi.

In seguito, con la fase di esecuzione, è chiarita la struttura fondamentale del procedimento, ovvero la parte relativa all'algoritmo di analisi genetica, ed è definita la successione dei diversi passaggi: dalle estrazioni dei subset di DNA, fino alla produzione e semplicazione dei dati generati dal metodo.

Infine, sono esposte, nella fase di configurazione, le modalità di impostazione del programma, così da garantire una corretta gestione dei parametri e delle istruzioni da trasmettere ad esso.

### 2.1.1 Installazione

Il passaggio iniziale consiste nel predisporre l'ambiente di lavoro soddisfacendo i requisiti indispensabili per una valida esecuzione.

Nella preparazione di questa tesi, è stato scritto uno script apposito per questa fase preliminare, denominato *installer.sh*, in modo da rendere questo procedimento compatto e più rapido. Infatti, aver a disposizione un unico file da eseguire consente di automatizzare l'installazione e di garantire una gestione semplificato nel caso si parallelizzasse l'esecuzione su diversi nodi.

Questo eseguibile, in più, può risultare utile per coloro che sono già in possesso dei requisiti ma che preferiscono evitare di mischiare diversi ambienti di lavoro. Ciò perchè l'installer aggiunge al bash script *.bashrc* il percorso della directory Miniconda prodotta nell'installazione, così da indurre l'utilizzo di quei tool, tra cui *Conda* e *Snakemake*, inclusi nella medesima cartella.

Le due prerogative più importanti coinvolgono l'installazione di *Conda*, che è inevitabile per l'attivazione degli ambienti, e ovviamente quella di *Snakemake* per l'avvio del programma.

In seguito, sono richieste alcune librerie per Python, Pandas [11] e Matplotlib [12], che sono necessarie per una fase di sintesi dei file di benchmark e per le analisi statistiche. In più, pur non avendo uno spessore di primo piano, sono essenziali i tool PyYAML e psutil, che colmano alcune lacune dei contenuti principali installati.

Dopo aver superato tali punti, è fondamentale equipaggiarsi del genoma umano di riferimento e dei dati genetici che si desiderano sequenziare.

In relazione al progetto ivi presentato, il genoma umano di riferimento è stato ottenuto via web dal sito ufficiale del IGSR(International Genome Sample Resource); mentre il campione di DNA esaminato è stato scaricato dal database pubblico DDBJ(DNA Data Bank of Japan).

Conclusi gli interventi preparatori, può essere avviata l'esecuzione del procedimento senza dover curare altri aspetti.

### 2.1.2 Esecuzione

La fase esecutiva comprende una serie di processi che si possono raggruppare in tre macro sezioni ben definite. La prima di esse descrive il processo di estrazione dei sottogruppi, o subset, di sequenze di DNA che si sceglie analizzare. La seconda presenta la pipeline per l'analisi del genoma e dichiara quali sono i prodotti attesi. La terza racchiude un'operazione di riordinamento e di semplificazione di tali ultimi prodotti per agevolare le successive indagini statistiche.

### Estrazione

Gli oggetti delle analisi sono i subset genetici che si sceglie di estrarre dall'intero genoma del soggetto e per questo è lecito accennare a come avviene l'estrazione.

Il genoma del soggetto è solitamente contenuto in un file in formato di testo *fastq* che contiene le sequenze di nucleotidi rilevate durante le analisi in laboratorio. In questo progetto però, la rilevazione non è singola ma duplice e quindi i dati del paziente sono suddivisi in due file fastq accoppiati. Ogni singola lettura contenuta nel *fastq* è descritta da quattro linee che rappresentano: la marcatura della sequenza, la sequenza, l'identificazione del punteggio di qualità e il punteggio di qualità.

Nel lavoro inerente a questa tesi, è stato scritto uno script in Python(*split.py*) che svolge l'operazione di estrazione e per il quale è sufficiente trasmettere da linea di comando gli estremi della sezione che si vuole ricavare; come indicato di seguito.

```
$ python split.py {inizio} {fine}
```

Grazie all'inserimento degli estremi, il programma calcola il numero di reads desiderate, seleziona le linee corrispondenti(il quadruplo del numero di letture) e le trascrive in un nuovo file fastq.

Ai fini dello studio finale, la disposizione di sottogruppi sia con diversi numero di sequenze che in diverse regioni consente di analizzare nei particolari le proprietà di scalabilità delle computazioni.

Siccome il numero di letture contenute è enorme, ed essendo che ad ognuna corrispondono quattro linee, la dimensione del file fastq è di conseguenza molto grande; per questo risulta funzionale dotarsi di un meccanismo per l'estrazione di sottogruppi.

La creazione di un unico programma anche per l'estrazione dei subsets conferma l'impegno nel dotarsi di un sistema automatizzato che possa elaborare pressochè senza interventi esterni.

### Algoritmo di ricostruzione genetica

Una volta creati i subsets, può essere applicata la pipeline di ricostruzione genetica, che è interamente contenuta nell'apposito Snakefile. É importante notare che la procedura seguita è ripresa dal metodo GATK-LOD$_n$ solamente in alcuni passaggi. Tali passaggi, in particolare, sono concretizzati nello Snakefile da regole che si susseguono l'un l'altra in base alle dipendenze di ciascuna(sezione 1.2).

L'algoritmo GATK-LOD$_n$ è stato integrato solo fino al riallineamento di GATK, escludendo quindi i passaggi di riallineamento Indel e di ricalibrazione

dei punteggi di qualità. Allo stesso tempo non sono state considerate le fasi fondamentali di chiamata alle varianti e l'implementazione del classificatore $LOD_n$ di MuTect, vera novità del metodo GATK-$LOD_n$.

Questa forte selezione degli step è stata voluta appositamente per non appesantire lo studio sull'ottimizzazione computazionale e, quindi, è stato scelto di concentrare le analisi solo su quelle operazioni che formano le basi dell'algoritmo per l'indagine del DNA. In tale maniera, è facilitata sia l'indagine sulla scalabilità che il confronto tra i diversi nodi adoperati, così da poter trarre rapidamente le prime conclusioni.

Le prime regole consistono nell'indicizzazione del genoma umano di riferimento per i software che partecipano al sequenziamento. Queste coinvolgono il software di allineamento BWA, il tool di manipolazione Picard, il gestore di strumenti Samtools e i programmi affini a GATK. Per le prime tre applicazioni, questa operazione è portata avanti da una specifica opzione delle stesse, mentre per l'ultimo essa è inclusa già nell'uso di Samtools.

Terminate le indicizzazioni, è sfruttato BWA MEM per la mappatura del DNA del soggetto sul genoma di riferimento che, inoltre, produce un file di etichetta, in formato *SAM*, che è sottoposto ad un riordinamento da parte Picard con l'assistenza della modalità SortSam.

Dopo essersi procurati i file `sorted_bam`, è eseguito il comando MarkDuplicates di Picard per identificare le letture doppie ed è generato un nuovo file `dedup_bam`. Elaborando quest'ultimo, la regola successiva crea un indice(in un file *bai*) per il file *bam* in modo da velocizzare l'analisi dei dati nel *bam*; ciò è realizzato da un'altra funzionalità di Picard chiamata BuildBamIndex.

Per concludere, supportati dalla direttiva RealignerTargetCreator, propria del pacchetto GATK, il contenuto del file *bam* è riallineato localmente in modo da diminuire ed evidenzire il numero di variazioni presenti.

Per riassumere l'intero processo è possibile osservare la raffigurazione seguente 2.1.

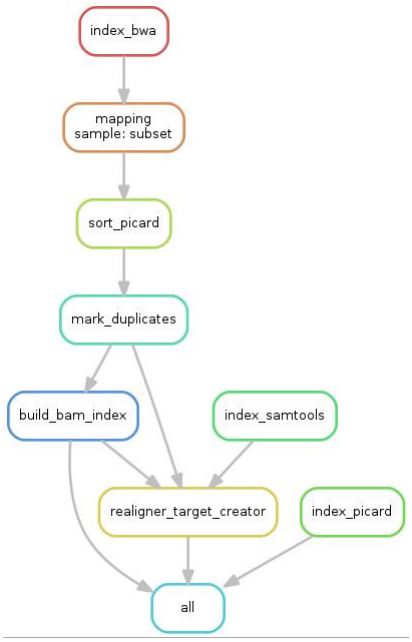



Figura 2.1: Schematizzazione del procedimento implementato nella pipeline.

### Organizzazione dei prodotti

Il materiale su cui sono condotte le analisi statistiche sono i particolari tecnici e tempistici di ognuna delle regole completate. Tali dati sono procurati, come già spiegato nella sezione 1.2, dalla direttiva benchmark in ogni passaggio. Ciò significa che al termine dell'algoritmo sono prodotti tanti singoli file di benchmarking quante regole sono state completate, contando pure il caso in cui esse venissero ripetute ricorsivamente. Per controllare la quantità di file e agevolare il futuro studio statistico, è stato scritto uno script in Python, denominato `script_benchmark`.

Ogni file benchmark è dotato di un nome con funzione di etichetta; in particolare, la forma è la seguente.

```
benchmark_{name}_subset_{sample}_n_sim_{n_sim}_cputype_{cpu_type}
    _thrs_{thrs}_ncpu{n_cpu}.txt
```

Gli attributi contenuti nel nome indicano ordinatamente: il nome della regola completata, il tipo di campione analizzato, il numero della simulazione, il tipo di cpu utilizzata e il numero di threads e di cpu adoperati. Mentre i primi due sono ricavati in automatico, gli altri fanno riferimento al file di configurazione dello Snakefile, come sarà spiegato nel paragrafo seguente.

Il lavoro svolto dallo script consiste quindi nel leggere ognuno dei file benchmark generati, considerare le etichette delle simulazioni e trasferire i dati

incolonnati in un'unica tabella. La tabella risultante non è dotata, quindi, solo delle colonne predefinite da *Snakemake* nell'operazione di benchmarking(1.2), bensì è arricchita dai dettagli contenuti nell'etichettatura. In particolare, i dati sono organizzati nella tabella in ordine crescente rispetto al numero di simulazione.

All'avvio di questo script è controllato se è presente una tabella con lo stesso nome: in caso affermativo, i nuovi dati sono accodati ai precedenti; al contrario, ne è inizializzata un'altra. In particolare, è necessario accedere al codice sorgente dello script se si vuole modificare il nome della tabella.

Attraverso questo sistema, il prodotto finale della fase di esecuzione, ed un unico oggetto dell'analisi statistica, è una tabella che racchiude le proprietà relative alle simulazioni ultimate.

Prima di procedere con l'esposizione dello studio effettuato su tali tabelle, è indispensabile perfezionare la descrizione del procedimento, specificando le opzioni di configurazione su cui essa si struttura.

### 2.1.3 Configurazione

Le ultime componenti del procedimento da delineare sono le modalità che definiscono le condizioni sotto cui deve essere portata avanti la computazione. Queste componenti sono riportate nei file di configurazione e, per questa tesi, sono stati scritti due file di questo tipo che corrispondono rispettivamente a *Snakemake* e a *Conda*.

Le informazioni necessarie allo Snakefile sono procurate dal file *config.yaml*, il quale comunica a *Snakemake* alcuni dettagli della simulazione e le indicazioni su certe dipendenze.

Gli attributi della simulazione completano semplicemente l'etichettatura dei file di benchmarking e, pur rivestendo uno ruolo marginale, tali etichette permettono allo script di organizzazione dei dati una lettura rapida e quindi un'istanza immediata della tabella. Come già citato precedentemente, i dettagli trascritti nel file di configurazione sono il tipo di cpu sfruttata, il numero della simulazione, il numero di threads adoperati e il numero di cpu coinvolte.

É stato scelto di gestire queste informazioni come parametri per non irrigidire il programma, visto che le macchine e i meccanismi usati possono essere innumerevoli. Nello specifico, per modificare questi dettagli nelle etichette è sufficiente agire da linea di comando, come indicato nel paragrafo 1.2.1.

Sempre nello stesso file di configurazione sono presenti alcune chiavi che rappresentato le dipendenze non implementabili automaticamente negli ambienti di *Conda* per *Snakemake*. Innanzitutto, sono presenti due indicazioni

necessarie per il meccanismo di mapping, che sono l'utilizzo di *Illumina* come piattaforma e di *WES-Nextera-Rapid-Capture* come libreria. A seguire, è indicato l'indirizzo in cui si può trovare il Genome Analysis ToolKit, da applicare nella procedura di riallineamento.

Altre istruzioni sulla configurazione del procedimento sono richieste da *Conda* per la creazione, attivazione e disattivazione degli ambienti di lavoro durante l'esecuzione di *Snakemake*(paragrafo 1.2.1).

Sempre nell'ambito di questa tesi, è stato scritto solo un documento di configurazione relativo a *Conda*, dato che la maggior parte dei processi necessita di un ambiente con le stesse caratteristiche. Nello specifico, quest'unico file di riferimento è contenuto nella directory *envs* ed è chiamato `config_conda.yaml`.

I requisiti richiesti per l'ambiente sono soddisfatti istruendo espressamente *Conda* all'utilizzo del canale *bioconda* per procedere con l'installazione di tre software coinvolti nel sequenziamento: *BWA*, *Picard* e *Samtools*.

Ora che la natura del sistema è stata argomentata e sono stati approfonditi pure le procedure configurative, è possibile illustrare il percorso di studio statistico effettuato, accompagnato da una presentazione del genere di simulazioni conseguite.

## 2.2 Analisi statistiche

Questa sezione si occupa di descrivere quale tipologia di studio statistico è stato compiuto e quali sono state le simulazioni sostenute. In particolare, saranno indicate quali caratteristiche sono state considerate e quali relazioni sono state oggetto di studio.

L'analisi dei dati è stata condotta utilizzando il linguaggio di Python, sulla piattaforma iPython [13], e dotandosi delle librerie: Pandas, Matplotlib, Seaborn [14], NumPy [15] e SciPy [16].

Il trattamento dei dati ha coinvolto inizialmente le funzioni sui dataframe di Pandas per l'estrazione e la modifica delle tabelle finali ricavate al termine del procedimento. Tali tabelle sono state ridotte ad un unico dataframe a cui sono state aggiunte quattro colonne per facilitare l'analisi. Le prime due contengono gli estremi del subset, ovvero la posizione della lettura iniziale e la posizione di quella finale, la terza contiene il numero di letture di ogni subset e la quarta il logaritmo del tempo di esecuzione. Da questo unico contenitore sono state estratte progressivamente le tabelle utili alle varie analisi.

La sezione è divisa in tre sottosezioni, dove le prime due corrispondono alle caratteristiche principali su cui è stata condotta l'indagine statistica: il

tempo di esecuzione e la memoria impiegata. L'ultima sezione include una semplice presentazione sul tipo di simulazioni che sono state completate.

### 2.2.1 Analisi sul tempo di esecuzione

Lo studio sul tempo di esecuzione è stato eseguito con lo scopo di delineare un andamento preciso di esso per ogni regola e per ognuna dei nodi adoperati.

Inizialmente è stata derivata dal dataframe generale una tabella in cui ogni regola svolta corrispondesse a un subset e che quindi, non fossero presenti quelle regole indipendenti dal tipo di dati analizzati. I lavori che dipendono dai dati sono, in ordine di esecuzione: la mappatura(mapping), l'ordinamento tramite picard(sort picard), la rilevazione dei duplicati(mark duplicates), la creazione dei file bam(build bam) e il riallineamento(realigner). Al contrario, i processi indipendenti sono le indicizzazioni dello human reference per bwa, per picard e per samtools(index bwa, index picard e index samtools).

In seguito è stata rappresentata su di un grafico la dipendenza del tempo di esecuzione di ogni regola rispetto alla grandezza del subset, identificando sullo questo stesso grafico un andamento per ogni nodo. In questa maniera è stato possibile verificare le differenze di prestazione tra le macchine server, adattando a ciascuna di esse una regressione lineare tra il tempo e il numero di sequenze del subset.

Una volta osservato l'andamento del tempo per ogni singola regola, l'attenzione è stata rivolta al tempo totale impiegato dalle regole dipendenti dal subset. Per calcolare il tempo totale è stata creata una nuova tabella che contenesse tali regole, il tipo di cpu, il numero della simulazione e il tempo totale di esecuzione.

Un tale grafico consente di osservare come cresce il tempo di esecuzione complessivo all'aumentare del numero di letture e come ciò è influenzato dal tipo di macchina adoperata.

Terminato lo studio sulle regole dipendenti dalla grandezza dei subset, sono state brevemente considerate anche le regole indipendenti. Anche in questo caso è stata tracciata su un grafico la dipendenza del tempo di esecuzione rispetto ad ogni regola, distinguendo il comportamento tra i differenti nodi.

Infine, sono stati estratti subset di ugual grandezza ma con punti di partenza differenti, che hanno permesso di stimare la variabilità dei tempi di esecuzione in base alla posizione di partenza.

Il tempo impiegato per concludere ogni regola non è però l'unico fattore da prendere in considerazione, visto che per promuovere l'uso della parallelizzazione è indispensabile analizzare le modalità di utilizzo della memoria.

### 2.2.2 Analisi della memoria occupata

La memoria fisica studiata è ottenuta grazie a Snakemake dal tool psutil sottoforma di una memoria detta RSS o *Resident Set Size*(paragrafo 1.2).

I dettagli sulla memoria che viene impiegata nei singoli processi è di fondamentale importanza per un corretto sviluppo di un futuro sistema parallelizzato. Infatti, lo studio sull'andamento di tale memoria consente di verificare fino a che livello di scalabilità la cpu è proficua e quando il suo uso comincia a saturare. Questi comportamenti indicano fino a che punto è possibile ottimizzare l'utilizzo dei core della cpu e identificano le memorie richieste dalle varie regole. Distinguere quali sono i processi che necessitano maggiormente di memoria e quelli che ne utilizzano una scarsa percentuale, permette di conoscere quali sono i nodi per le varie regole che ottimizzano l'esecuzione.

Lo studio statistico condotto è stato interessato, come per le tempistiche, ad osservare il comportamento della memoria in base alle regole, ai subsets e ai diversi nodi. Perciò, è stata inizialmente posta in relazione la memoria occupata da ogni regola con il numero di sequenze del subset, distinguendo le macchine. Successivamente è stata eseguita la stessa operazione per le regole indipendenti dai dati ed infine, è stato studiato il comportamento della memoria lasciando invariato il numero di letture ma considerando diversi punti di partenza.

### 2.2.3 Le simulazioni completate

Le simulazioni che sono state eseguite ai fini di questa tesi sono state scelte per valutare le performance sui nodi dei cluster a disposizione, in modo da evidenziarne la potenziale scalabilità.

I nodi low power adoperati, che sono stati illustrati nel paragrafo 1.3, sono stati impiegati in base alle loro diverse potenze computazionali e sono stati confrontati con un nodo del cluster bio8, utilizzato come riferimento per le macchine tradizionali. È stato utilizzato un solo core per ognuno dei nodi in tutto l'arco del procedimento, rimandando ad un futuro sviluppo l'utilizzo di core multipli.

Tutti i nodi hanno processato gli stessi subset di dati con un numero di letture iniziale pari a centomila fino a 3 milioni, passando per i multipli di centomila e di un milione. In questo modo è stato possibile studiare le potenzialità delle macchine a processare dati identici con larghezza crescente e come questo incremento condiziona l'uso della cpu.

Solamente sui nodi xeond e bio8 sono state portate avanti le simulazioni fino a subset con un numero di letture pari a 9 milioni, continuando da 3 milioni a scalare progressivamente di 1 milione.

Non sono stati fatti proseguire anche gli altri nodi perchè lo scopo della tesi è valutare il comportamento dei nodi su piccoli subset di dati e quali indicazioni questo possa dare per sviluppare un'esecuzione parallelizzata efficace. Visto che per tale valutazione sono stati sufficienti i dati citati, solo le due macchine più performanti sono andati oltre per confermare le stime ottenute.

Ciò significa che tutte le simulazioni sono state svolte solamente su una piccola frazione delle letture in possesso. Infatti il numero di letture complessivo del paziente è 45 milioni e la massima grandezza di un subset condivisa tra tutti i dispositivi è stata di 3 milioni, che equivale circa al 6.7%. Per xeond e bio8 invece, i subset sono stati allargati a 9 milioni, ovvero circa il 20%. La tabella 2.1 mostra la dimensione occupata su disco dal numero di letture contenute nei subset.

| numero di letture | dimensione su disco |
|---|---|
| $1 \times 10^5$ | 2x 28.4 MB |
| $1 \times 10^6$ | 2x 284.9 MB |
| $3 \times 10^6$ | 2x 854.9 MB |
| $9 \times 10^6$ | 2x 2.6 GB |
| $4.5 \times 10^7$ | 2x 12.8 GB |

Tabella 2.1: Stima della dimensione dei subset in relazione al numero di letture. L'ultimo valore si riferisce all'intero paziente.

Per ogni numero di letture la dimensione occupata su disco è il doppio perchè, come citato nel paragrafo 2.1.2, sono generati in fase di estrazione due file *.fastq* per ogni subset.

In dettaglio, queste simulazioni sono state ripetute dieci volte per i nodi xeond e bio8 mentre sono stata ripetute cinque volte per gli altri nodi.

Infine utilizzando lo script `split.py` descritto nella sezione 2.1.2, sono stati estratti subsets contenenti letture diverse, ma dello stesso numero, che sono stati studiati in una fase delle analisi. L'unico numero di letture considerato è stato centomila e ciò è stato eseguito solo per i nodi dei cluster low power, e quindi non da bio8.

Nel capitolo seguente saranno esposti gli esiti più rilevanti, selezionati dai prodotti finali dell'analisi statistica effettuata sui dati delle simulazioni.

# Capitolo 3

# Risultati

I risultati finali sono suddivisi nelle due caratteristiche su cui è stata sviluppata l'analisi dei dati. Sono quindi presenti una sezione relativa ai tempi di esecuzione e una sulla memoria fisica sfruttata. La descrizione delle relazioni più interessanti è, inoltre, accompagnata dall'utilizzo di alcuni tra i grafici elaborati durante lo studio statistico.

In sede di analisi sono stati usati subset con dimensione minima di centomila.

## 3.1 Tempi di esecuzione

L'indagine sui tempi di esecuzione, come spiegato nel paragrafo 2.2.1, è stata condotta approfondendo le seguenti relazioni: i tempi di esecuzione per le regole, la durata complessiva del processo e i tempi impiegati per intervalli differenti con lo stesso numero di sequenze. Questa sezione si occuperà di presentare gli esiti finali più rilevanti di ognuno di tali aspetti.

### 3.1.1 Tempi e regole

Le regole previste dal procedimento hanno concentrato l'analisi su due campi, una relativo a quei processi che non dipendono dal tipo di dati considerato e uno relativo a quelli che invece dipendono. Considerando le prime, si può notare dalla figura 3.1 come l'unico processo significativo sia l'indicizzazione dello human reference per BWA, mentre le rimanenti non influiscano in alcun modo. In più, la differenza tra le potenze computazionali è netta già da questa figura, dato che tra il nodo migliore e il peggiore vi è uno scarto di più di un'ora.

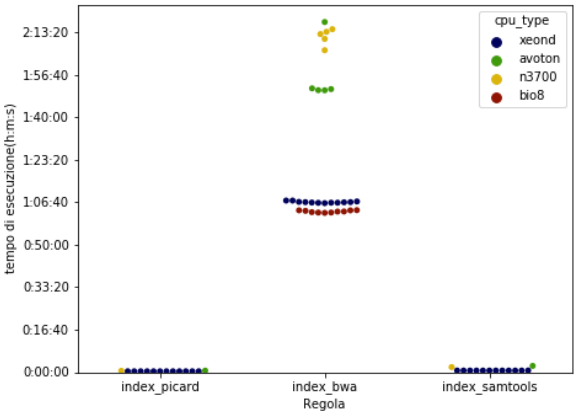



Figura 3.1: Tempi di esecuzione per le regole indipendenti dal subset.

Nonostante la durata per l'indicizzazione sia notevole, questa fase è riprodotta una sola volta per tutte le successive simulazioni, visto che è riferita solamente al genoma umano di riferimento.

Passando alle regole dipendenti, i grafici riportati nelle figure 3.2, 3.3, 3.4, 3.5 e 3.6 mostrano i vari andamenti per la grandezza dei subset.

Per ciascuna figura è anche riportata una tabella con i dettagli delle rigressioni lineari effetuate per descrivere il tempo di esecuzione in funzione del numero di sequenze lette. La regressione è stata effettuata con il metodo di Theil-Sen. Questo metodo implica una regressione lineare robusta, ovvero che non risente della presenza di *outlier* nei dati.

Nelle tabelle è riportato l'intervallo di confidenza della pendenza della retta. Visto che nessuno di questi intervalli di confidenza contiene lo zero, per tutte le regole vi è una dipendenza significativa fra il numero di sequenze ed il tempo impiegato.

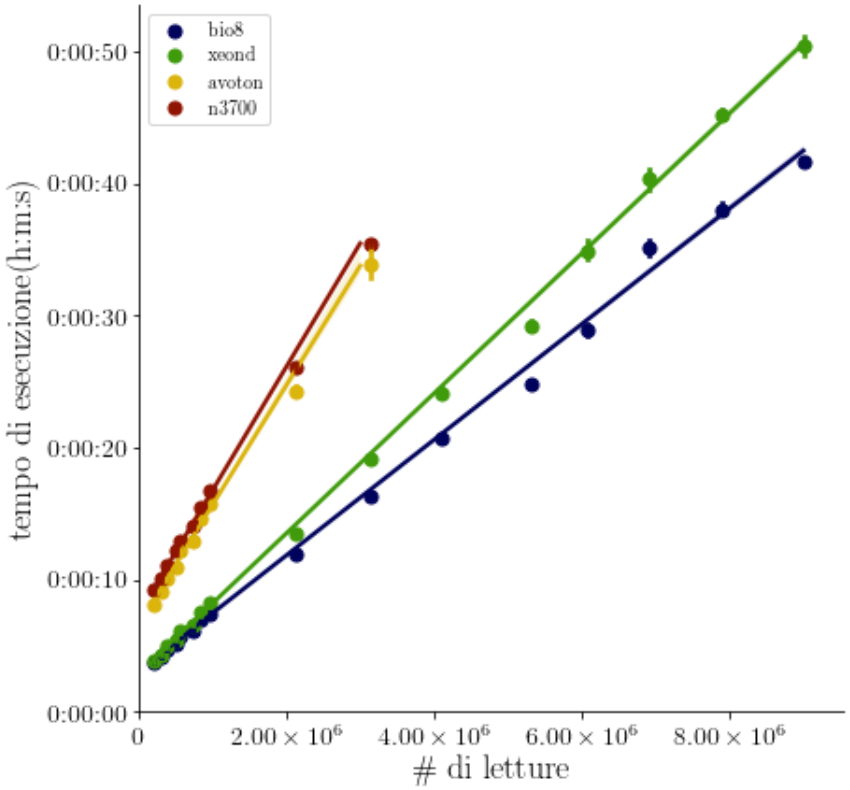



Figura 3.2: Tempi per Build BAM.

| tipo di cpu | pendenza $\frac{secondi}{10^6 \text{ letture}}$ | intercetta *secondi* | min pendenza $\frac{secondi}{10^6 \text{ letture}}$ | max pendenza $\frac{secondi}{10^6 \text{ letture}}$ |
|---|---|---|---|---|
| avoton | 9.2 | 6.4 | 9 | 9.5 |
| bio8 | 4.4 | 3.1 | 4.4 | 4.4 |
| n3700 | 9.4 | 7.4 | 9.3 | 9.5 |
| xeond | 5.3 | 3 | 5.3 | 5.3 |

Tabella 3.1: Dettagli retta di fit per Build BAM.

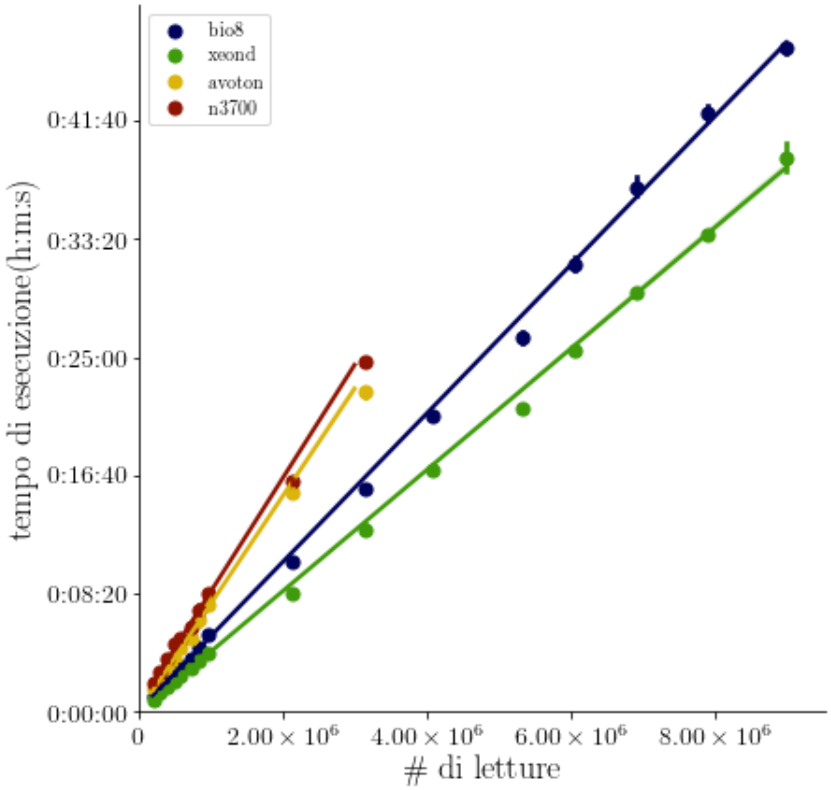



Figura 3.3: Tempi per Mapping.

| tipo di cpu | pendenza $\frac{secondi}{10^6 \text{ letture}}$ | intercetta *secondi* | min pendenza $\frac{secondi}{10^6 \text{ letture}}$ | max pendenza $\frac{secondi}{10^6 \text{ letture}}$ |
|---|---|---|---|---|
| avoton | 440 | 6.8 | 430 | 45 |
| bio8 | 310 | 7.9 | 310 | 310 |
| n3700 | 470 | 24 | 470 | 480 |
| xeond | 250 | -3.3 | 250 | 250 |

Tabella 3.2: Dettagli retta di fit per Mapping.

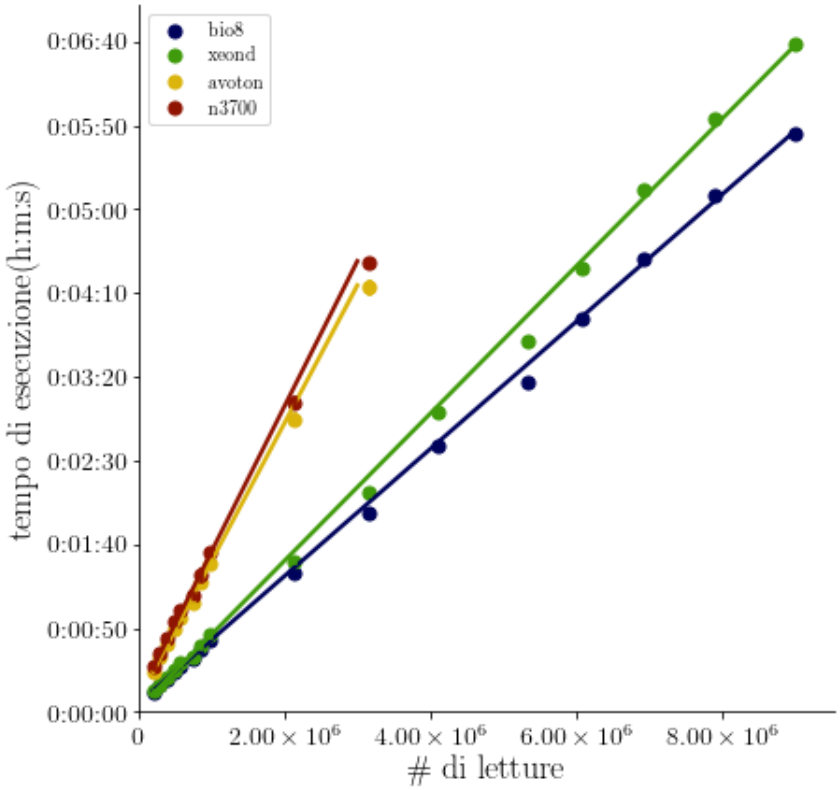



Figura 3.4: Tempi per Mark Duplicates.

| tipo di cpu | pendenza $\frac{secondi}{10^6 \text{ letture}}$ | intercetta *secondi* | min pendenza $\frac{secondi}{10^6 \text{ letture}}$ | max pendenza $\frac{secondi}{10^6 \text{ letture}}$ |
|---|---|---|---|---|
| avoton | 81 | 7.9 | 81 | 82 |
| bio8 | 38 | 4.7 | 38 | 38 |
| n3700 | 86 | 8.9 | 85 | 87 |
| xeond | 44 | 2.4 | 43 | 44 |

Tabella 3.3: Dettagli retta di fit per Mark Duplicates.

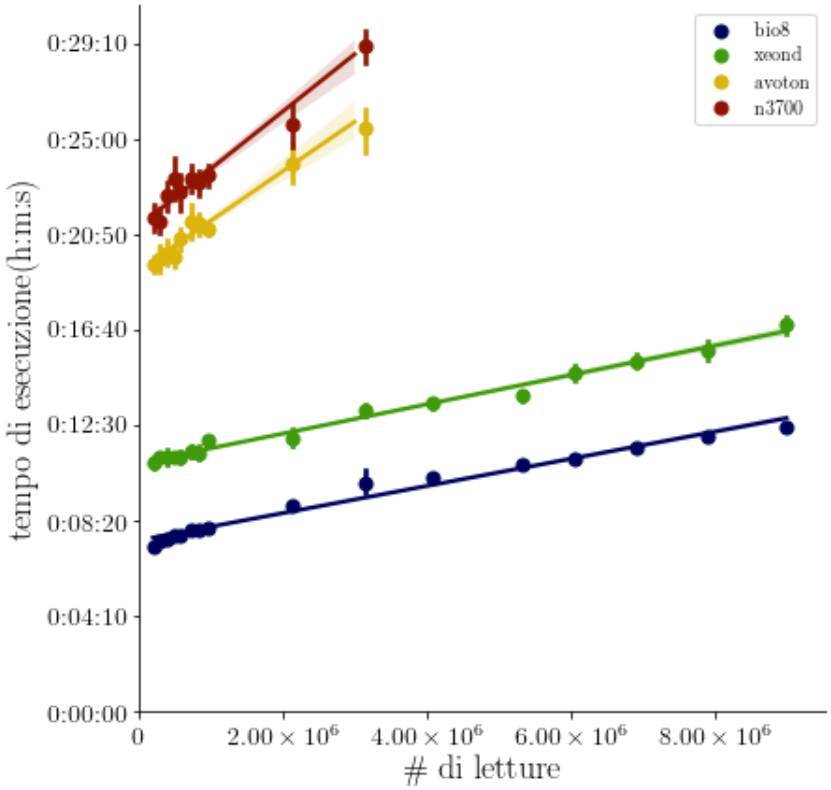



Figura 3.5: Tempi per i tempi di Realigner.

| tipo di cpu | pendenza $\frac{secondi}{10^6 \text{ letture}}$ | intercetta *secondi* | min pendenza $\frac{secondi}{10^6 \text{ letture}}$ | max pendenza $\frac{secondi}{10^6 \text{ letture}}$ |
|---|---|---|---|---|
| avoton | 130 | $1.2 \times 10^3$ | 110 | 160 |
| bio8 | 38 | $4.5 \times 10^2$ | 36 | 39 |
| n3700 | 150 | $1.3 \times 10^3$ | 120 | 170 |
| xeond | 38 | $6.8 \times 10^2$ | 36 | 40 |

Tabella 3.4: Dettagli retta di fit per Realigner.

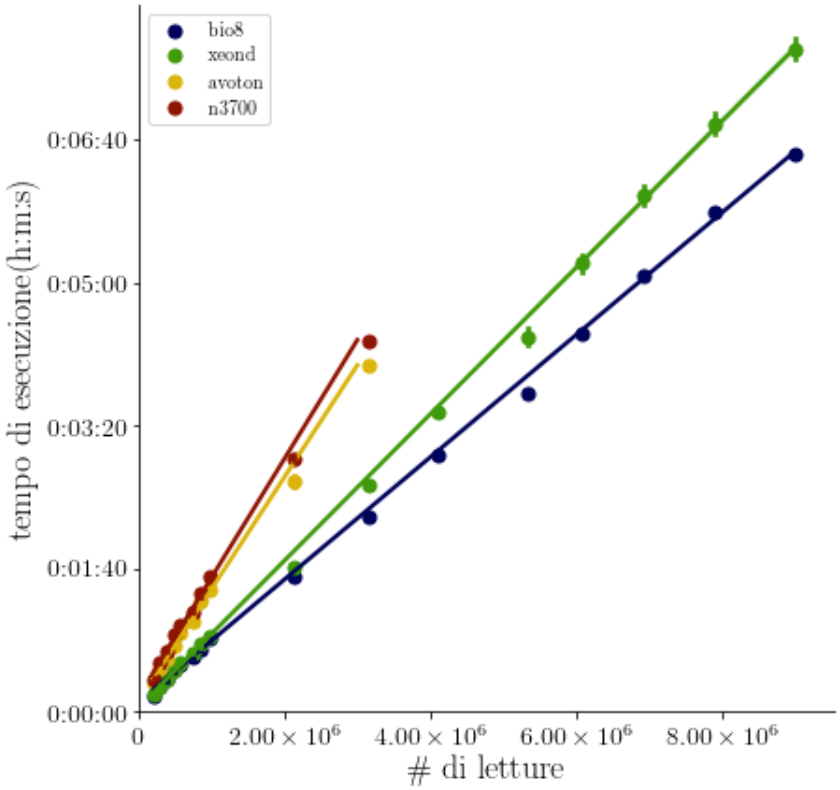



Figura 3.6: Tempi per Sort Picard.

| tipo di cpu | pendenza $\frac{secondi}{10^6 \text{ letture}}$ | intercetta *secondi* | min pendenza $\frac{secondi}{10^6 \text{ letture}}$ | max pendenza $\frac{secondi}{10^6 \text{ letture}}$ |
|---|---|---|---|---|
| avoton | 80 | 6.5 | 79 | 83 |
| bio8 | 43 | 7.9 | 43 | 43 |
| n3700 | 84 | 12 | 83 | 86 |
| xeond | 51 | 2.9 | 51 | 51 |

Tabella 3.5: Dettagli retta di fit per Sort Picard.

La fase di riallineamento ha un comportamento differente dalle altre(realigner figura 3.5), in quanto il tempo di avvio della regola, rappresentato dall'intercetta, costituisce una parte importante del tempo complessivo di esecuzione della regola stessa.

Nei grafici sono distinguibili due gruppi di nodi significativamente diversi: avoton e n3700 contro xeond e bio8. Questa distinzione è confermata dai valori delle pendenze riportate nelle tabelle 3.1, 3.3, 3.2, 3.4 e 3.5. In particolare,

il fatto che xeond e bio8 condividano lo stesso genere di architettura(tabella 1.1) indica come questa caratteristica possa influenzare l'esecuzione.

Infine, dai grafici si può notare che le regole di lunga durata sono il mapping e il realigner, mentre quella che impiega il minor tempo è il build bam. Riguardo a mark duplicates e sort picard, si nota che queste regole condividono la stessa durata e la stessa crescita(tabelle 3.3 e 3.5).

### 3.1.2 Durata complessiva

Un altro aspetto rilevato è stato il tempo complessivo richiesto per svolgere tutti i passaggi che esclusivamente dipendono dal numero di letture nel subset, escludendo tutti i passaggi non dipendenti (come l'indicizzazione del reference umano fatto da BWA). Questi tempi sono rappresentativi del tempo richiesto per la pipeline per ciascun paziente.

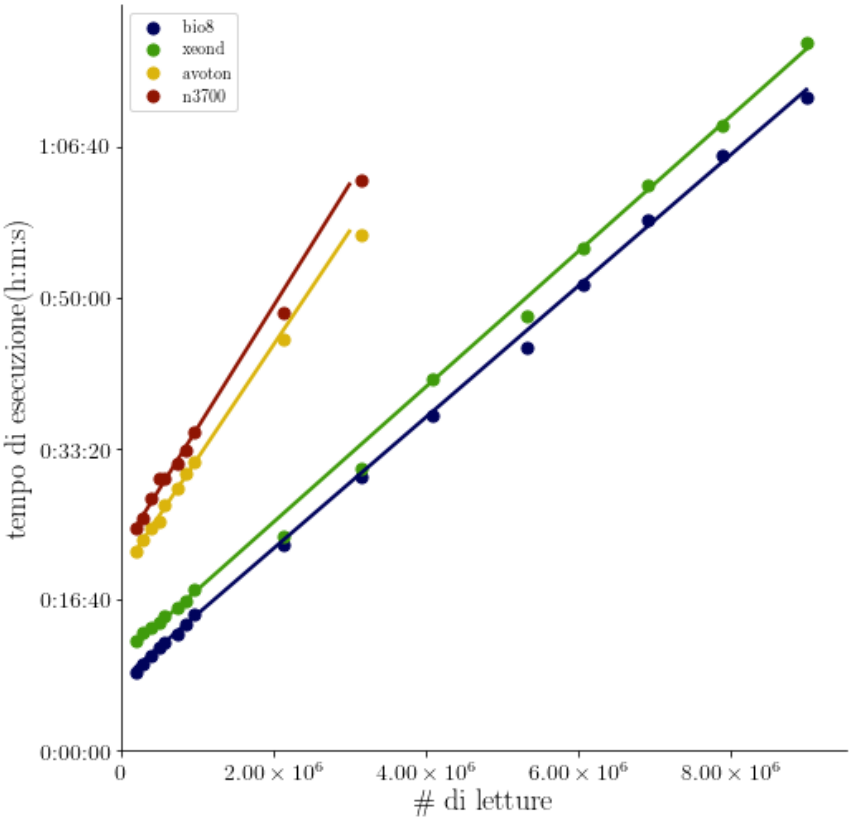



Figura 3.7: Tempi complessivo di esecuzione.

| tipo di cpu | pendenza $\frac{secondi}{10^6 \text{ letture}}$ | intercetta $secondi$ | min pendenza $\frac{secondi}{10^6 \text{ letture}}$ | max pendenza $\frac{secondi}{10^6 \text{ letture}}$ |
|---|---|---|---|---|
| avoton | 760 | $1.2 \times 10^3$ | 730 | 790 |
| bio8 | 440 | $4.7 \times 10^2$ | 440 | 440 |
| n3700 | 800 | $1.3 \times 10^3$ | 770 | 830 |
| xeond | 390 | $6.8 \times 10^2$ | 390 | 400 |

Tabella 3.6: Pendenze per i tempi complessivi.

I grafici in figura 3.7 e i valori in tabella 3.6 confermano che i nodi sono separati in due coppie comparabili: avoton e n3700, xeond e bio8.

### 3.1.3 Variabilità dei tempi di esecuzione

Tutti i subset utilizzati nella sezione precedente condividono la stessa posizione iniziale di estrazione. Per valutare la variabilità del tempo di esecuzione dalla posizione iniziale di estrazione, sono state ripetute le analisi con subset di stessa lunghezza ma con diversi punti di estrazione. Questi punti di estrazione sono stati scelti in modo da non aver sovrapposizione fra questi subset e il numero di letture stabilito è stato centomila

| regola<br>tipo di cpu | build bam | mapping | mark duplicates | realigner | sort picard |
|---|---|---|---|---|---|
| avoton | $7.3 \pm 0.2$ | $60 \pm 14$ | $16.4 \pm 0.3$ | $1110 \pm 50$ | $12.8 \pm 0.4$ |
| n3700 | $8.00 \pm 0.18$ | $74 \pm 3$ | $18.4 \pm 0.8$ | $1250 \pm 60$ | $14.7 \pm 0.4$ |
| xeond | $3.27 \pm 0.07$ | $30 \pm 11$ | $8.1 \pm 0.6$ | $610 \pm 30$ | $7.2 \pm 0.9$ |

Tabella 3.7: Media e deviazione standard, espresse in secondi(s), dei tempi di esecuzione delle regole su diversi subset da centomila letture.

I dati associati a ciascuna regola indicano che i nodi tendenzialmente si stabilizzano su valori fissati. Infatti, eccetto il mapping che mostra l'andamento meno stabile, i nodi impiegano la stessa quantità di tempo senza dipendere dal contenuto delle letture.

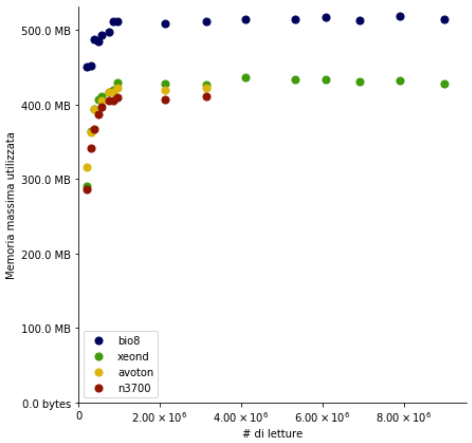
## 3.2 Memoria utilizzata

La memoria occupata dai singoli processi è stata studiata analogamente al tempo trascurando però lo studio di una memoria complessiva per tutti i

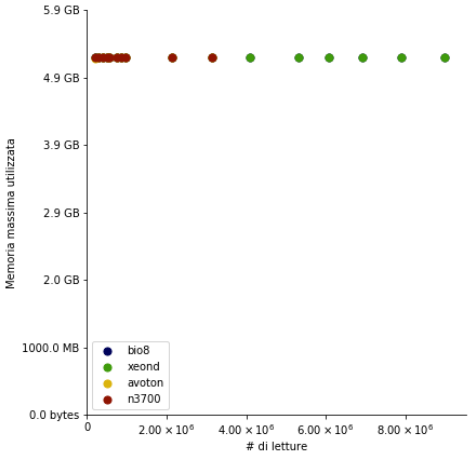
passaggi. È analizzato in un primo momento l' occupazione della memoria per ognuna delle regole e si è poi stimata la variabilità di queste misure a partire da diverse posizioni iniziali. La memoria intesa in questo capitolo si riferisce alla memoria "Resident Set Size" ricavata dal tool psutil e riportata da Snakemake, come già indicato nel paragrafo 2.2.2.

### 3.2.1 Consumo di memoria per le regole

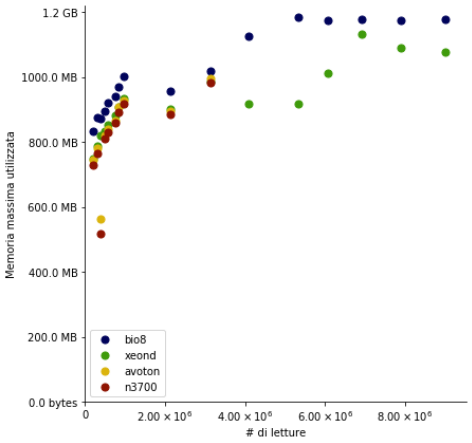
Le informazioni sui vari comportamenti sono tratte dai grafici in figura 3.6 e dalla tabella 3.8.



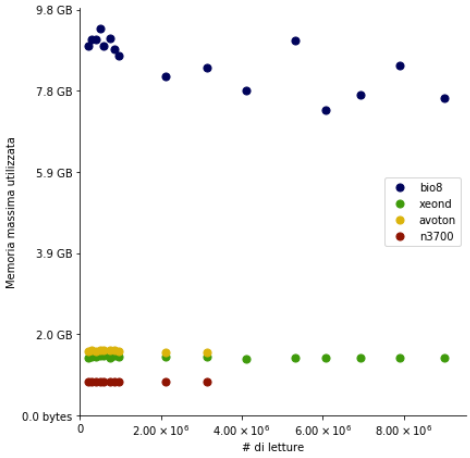

(a) *Memoria utilizzata da Build BAM.*



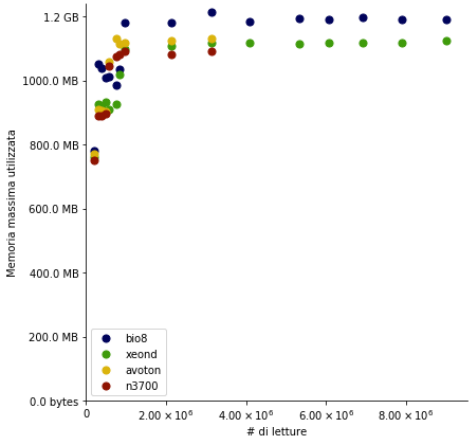

(b) *Memoria utilizzata da Mapping.*




(c) *Memoria utilizzata da Mark Duplicates.*




(d) *Memoria utilizzata da Realigner.*




(e) *Memoria utilizzata da Sort Picard.*

Figura 3.6

| regola<br>cpu type | build bam | mapping | mark duplicates | realigner | sort picard |
|---|---|---|---|---|---|
| avoton | 429 | 5305 | 1019 | 1626 | 1150 |
| bio8 | 527 | 5295 | 1042 | 10362 | 1232 |
| n3700 | 419 | 5306 | 988 | 859 | 1115 |
| xeond | 435 | 5305 | 1004 | 1582 | 1138 |

Tabella 3.8: Tabella dei massimi valori di memoria impiegati, espressi in MB.

Si può vedere che nelle regole di marcamento dei duplicati(figura 3.7c), di formazione del file BAM(figura 3.8a) e di riordimento per picard(figura 3.6e), la memoria satura una volta superato il milione di letture.

Le altre regole che analizzano specificatamente i dati, la mappatura(figura 3.8b) e il riallineamento(figura 3.7d), hanno entrambe un'occupazione fissa della memoria. Nel caso del mapping, infatti, tutte le macchine occupano la stessa quantità di memoria, indicando una saturazione generale, mentre nel riallineamento ciascuna cpu utilizza una quantità fissata di memoria che sembra essere una frazione della memoria totale disponibile.

Per le regole che non dipendono dal paziente, riportate in figura 3.7, la richiesta di memoria non sembra invece dipendere significativamente dal nodo.

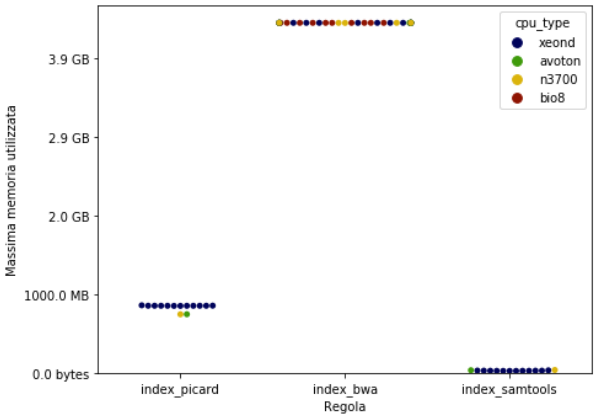



Figura 3.7: Memoria impiegata dalle regole indipendenti dai subsets.

### 3.2.2 Variabilità della memoria occupata

Lo studio sulla memoria impiegata per intervalli diversi con ugual numero di sequenze è stato compiuto solo sui nodi dei cluster low power. La tabella 3.9 riporta la media e la deviazione standard per ciascuna regola, su ogni nodo. In particolare il numero di letture considerato è stato di centomila.

| regola<br>tipo di cpu | build bam | mapping | mark duplicates | realigner | sort picard |
|---|---|---|---|---|---|
| avoton | $240 \pm 8$ | $5279.0 \pm 0.7$ | $702 \pm 7$ | $1600 \pm 15$ | $585 \pm 3$ |
| n3700 | $222 \pm 4$ | $5280 \pm 6$ | $686 \pm 7$ | $830 \pm 11$ | $572 \pm 3$ |
| xeond | $230 \pm 12$ | $5280 \pm 16$ | $703 \pm 6$ | $1430 \pm 30$ | $576 \pm 3$ |

Tabella 3.9: Media e deviazione standard, espresse in MB, della memoria occupata dalle regole su diversi subset da centomila letture.

I valori ottenuti evidenziano che la memoria è fortemente stabile in ogni regola, per ogni macchina. Ciò conferma che le operazioni sfruttano la stessa quantità di memoria indipendentemente dal contenuto dei subset e, eccetto il caso del realigner, ne occupano pressochè lo stesso valore.

# Capitolo 4

# Conclusioni

Il lavoro effettuato durante lo svolgimento di questa tesi ha lo scopo di esplorare la possibilità di sostituire i nodi di calcolo tradizionalmente usati per calcoli ad alta performance, con nodi a basso consumo energetico. Questo è motivato dalla possibilità di avere una maggior potenza di calcolo a parità di spesa, grazie al minor costo per unità dei server low power, a fronte di una capacità di calcolo comparabile. Per testare questi nodi di calcolo è stata sfruttata una pipeline di calcolo bioinformatico, che ha requisiti molto elevati in termini di potenza di calcolo, di occupazione di memoria e di spazio d'archiviazione.

Questa pipeline è stata testata su quattro nodi dai requisiti di potenza progressivamente più elevati, partendo da una macchina a consumi molto ridotti (6W per n3700) fino ad un nodo di calcolo tradizionale (180W per bio8). Le analisi delle simulazioni effettuate, riportate nel capitolo 3, hanno approfondito il tempo di esecuzione e la quantità di memoria occupata per ciascuna operazione svolta nella pipeline. Questi due parametri sono fondamentali per la progettazione dei cluster di calcolo in quanto determinano la dimensione minima necessaria dei nodi.

L'analisi dei tempi di esecuzione mostra come vi siano due gruppi di nodi di calcolo distinguibili in base all'architettura del processore. I due nodi a potenza più ridotta (n3700 e avoton) hanno dei tempi d'esecuzione generalmente doppi rispetto a quelli a potenza più elevata. Il terzo nodo(xeond), sempre low power, ha dei tempi di esecuzione confrontabili con il nodo di calcolo tradizionale(bio8) a fronte di un consumo energetico di circa un terzo e di un costo dieci volte inferiore.

L'analisi della memoria occupata dalle varie regole mostra due diversi comportamenti. In un caso, si ha una saturazione della una quantità di memoria occupata ad un valore fisso, che varia poco tra i diversi nodi. Nell'altro caso le regole si adattano dinamicamente alla quantità di memoria libera di-

sponibile. In entrambi i casi la memoria occupata dalle singole regole è sempre risultata inferiore alla memoria disponibile in tutti i nodi, occupando al massimo 5 GB a fronte di una memoria complessiva di almeno 8 GB.

In base a questi risultati questa pipeline di calcolo bioinformatico sembra essere realisticamente eseguibile anche su nodi a bassa potenza senza una perdita considerevole di prestazioni. Questo, combinato con la possibilità di suddividere il calcolo in maniera *embarassingly parallel* rende i nodi di calcolo low power un'alternativa concreta a quelli tradizionali.

Il percorso di studio futuro avrà come primo step il completamento della pipeline di GATK-LOD$_{\text{n}}$. Una volta completata la struttura della pipeline, saranno svolte nuove simulazioni per confermare i risultati ottenuti in precedenza. In seguito saranno ripetute le simulazioni sui nodi sfruttando più core e successivamente interi cluster, sia low power che tradizionali che cloud.

L'esito positivo della ricerca consentirebbe agli enti accademici e ospedialieri di adottare una tecnologia più accessibile, meno costosa e soprattutto più veloce, che sostenga la ricerca in medicina e in biologia.

# Bibliografia


[1] Sam Behjati and Patrick S Tarpey. What is next generation sequencing? *Archives of disease in childhood - Education & practice edition*, 98(6):236–238, 2013.

[2] Jay Shendure and Hanlee Ji. Next-generation DNA sequencing. *Nature Biotechnology*, 26(10):1135–1145, 2008.

[3] Ítalo Faria do Valle, Enrico Giampieri, Giorgia Simonetti, Antonella Padella, Marco Manfrini, Anna Ferrari, Cristina Papayannidis, Isabella Zironi, Marianna Garonzi, Simona Bernardi, Massimo Delledonne, Giovanni Martinelli, Daniel Remondini, and Gastone Castellani. Optimized pipeline of MuTect and GATK tools to improve the detection of somatic single nucleotide polymorphisms in whole-exome sequencing data. *BMC Bioinformatics*, 17(S12):341, 2016.

[4] Aaron McKenna, Matthew Hanna, Eric Banks, Andrey Sivachenko, Kristian Cibulskis, Andrew Kernytsky, Kiran Garimella, David Altshuler, Stacey Gabriel, Mark Daly, and Mark A. DePristo. The genome analysis toolkit: A MapReduce framework for analyzing next-generation DNA sequencing data. *Genome Research*, 20(9):1297–1303, 2010.

[5] Kristian Cibulskis, Michael S Lawrence, Scott L Carter, Andrey Sivachenko, David Jaffe, Carrie Sougnez, Stacey Gabriel, Matthew Meyerson, Eric S Lander, and Gad Getz. Sensitive detection of somatic point mutations in impure and heterogeneous cancer samples. *Nature Biotechnology*, 31(3):213–219, 2013.

[6] Harold Varmus. The Impact of Physics on Biology and Medicine. *Physics World*, 12(9):27, 1999.

[7] Richard Pooley. Bridging the culture gap, 2005.

[8] Michael Zwolak and Massimiliano Di Ventra. Colloquium: Physical approaches to DNA sequencing and detection. *Reviews of Modern Physics*, 80(1):141–165, 2008.

[9] Johannes Köster and Sven Rahmann. Snakemake-a scalable bioinformatics workflow engine. *Bioinformatics*, 28(19):2520–2522, 2012.

[10] T. E. Oliphant. Python for scientific computing. *Computing in Science Engineering*, 9(3):10–20, May 2007.

[11] Wes McKinney. Data structures for statistical computing in python. In Stéfan van der Walt and Jarrod Millman, editors, *Proceedings of the 9th Python in Science Conference*, pages 51 – 56, 2010.

[12] J. D. Hunter. Matplotlib: A 2d graphics environment. *Computing in Science Engineering*, 9(3):90–95, May 2007.

[13] F. Perez and B. E. Granger. Ipython: A system for interactive scientific computing. *Computing in Science Engineering*, 9(3):21–29, May 2007.

[14] Michael Waskom, Olga Botvinnik, Paul Hobson, John B. Cole, Yaroslav Halchenko, Stephan Hoyer, Alistair Miles, Tom Augspurger, Tal Yarkoni, Tobias Megies, Luis Pedro Coelho, Daniel Wehner, cynddl, Erik Ziegler, diego0020, Yury V. Zaytsev, Travis Hoppe, Skipper Seabold, Phillip Cloud, Miikka Koskinen, Kyle Meyer, Adel Qalieh, and Dan Allan. seaborn: v0.5.0 (november 2014), November 2014.

[15] S. van der Walt, S. C. Colbert, and G. Varoquaux. The numpy array: A structure for efficient numerical computation. *Computing in Science Engineering*, 13(2):22–30, March 2011.

[16] Eric Jones, Travis Oliphant, Pearu Peterson, et al. SciPy: Open source scientific tools for Python, 2001–.