Indice

Prefazione

Questa tesi presenta un progetto dedito allo studio dell'efficienza computazionale di cluster a basso consumo energetico adoperati per l'analisi biofisica. In particolare, la ricerca esamina come l'utilizzo di macchine a minor dispendio energetico possano essere più convenienti e potenzialemente più potenti rispetto alle macchine tradizionali sfruttate nel ramo della ricerca biomedica e sanitaria.

Il sistema su cui sono stati concentrati gli sforzi è uno tra i più recenti sviluppati nella ricerca sulle mutazioni genetiche da cui derivano vari tipi di tumori: il sistema GATK-LOD$_{\text{n}}$. L'implementazione di una componente di questo metodo nel programma Snakemake, e quindi la creazione di un nuovo eseguibile, ha permesso una gestione più specifica dei vari step implicati, nel tentativo di ottimizzare l'esecuzione complessiva. Infatti, questo programma è in grado di parallelizzare i diversi compiti del metodo sia sullo stessa macchina che su diverse macchine, conservando l'esatta sequenzialità del metodo. Una maggior padronanza di questa caratteristica consente di verificare la fruibilità di insiemi di computer che collaborano come un solo apparato: i cluster.

La formazione di gruppi di computer nasce dalla necessità dei moderni organi di ricerca di potenziare le capacità computazionali, a cui deve essere anche alternato un contenimento dei costi sia economici che energetici. In questi tempi difatti, gli studi nel campo biomedico prevedono la collaborazione di professionisti in statistica dotati di computer ad alta potenza, che hanno il compito di fornire i risultati proficuamente ed in tempi sufficientemente ristretti. Lo sviluppo e il progresso in questi vari settori ha dipeso,e dipende tuttora, da un incremento della potenza di computazione e ciò comporta alcuni risvolti problematici. La crescita delle prestazioni dei computer ha come conseguenza di base un'inevitabile innalzamento dei costi di tali servizi e, quindi, una minor accessibilità alla maggioranza dei gruppi di ricerca. Questo però, non è l'unico effetto negativo poichè all'aumento della potenza segue un aumento del consumo energico, che è un fattore altamente penalizzante.

Negli ultimi anni, tali ragioni hanno portato alcuni studiosi ad interessarsi a metodi alternativi per la computazione e, in questa tesi, è introdotto il tema riguardante le simulazioni su cluster a basso consumo energetico. L'idea di fondo è il poter garantire ai ricercatori un risultato, in qualità di tempi, almeno pari a quello ottenuto con la metodologia tradizionale. Accompagna-

to però, dal vantaggio di consumare minor energia e spendere una quantità di fondi inferiore.

Nel primo capitolo saranno introdotti e approfonditi gli elementi cardine del progetto, a partire dall'esposizione del metodo GATK-LOD$_n$ sia nel funzionamento che nei risultati. Successivamente sarà evidenziata la componente del metodo che è implementata nell'ambiente di sviluppo del tool Snakemake e, quindi, saranno approfondite le capacità di questo strumento. Infine, sarà specificato il significato di low power e saranno mostrate le macchine adoperate nell'analisi, per poi concludere con un accenno al funzionamento dei cluster.

Nel secondo capitolo sarà spiegato dettagliatamente il funzionamento del programma con alcuni approfondimenti sui parametri utilizzati e verranno evidenziati i passaggi per un corretto uso del metodo, dall'installazione ai dati ottenuti. In più, sarà specificato quale tipo di analisi è stata compiuta su tali valori a disposizione.

Nel terzo capitolo verrano illustrati i risultati finali più rilevanti con l'aggiunta di tabelle e grafici utili ad impreziosire le analisi e ad accompagnare l'esposizione. Inoltre sarà presente anche un breve accenno ai dati non considerati proficui e a coloro che sono stati trascurati.

Per terminare saranno mostrate le conclusioni, le considerazioni finali e gli sviluppi futuri del progetto, in base agli esiti più interessanti tratti dalle indagini svolte.

Capitolo 1

Introduzione

Questo capitolo introduce le componenti principali del progetto ed ha il compito di spiegare in maniera approfondita le operazioni svolte da esse. Tali componenti sono state raggruppate nei tre campi su cui è stato condotto il lavoro: bioinformatica, sviluppo informatico e i dispositivi low power.

Il primo di questi è il campo bioinformatico e, quindi, un approfondimento del metodo GATK-LOD$_{\text{n}}$ riguardante la ricerca sull'origine dei tumori attraverso procedure di natura bioinformatica. Inoltre, sarà sottolineata l'importanza della fisica nell'ambiente di lavoro e nella creazione degli algoritmi.

Il secondo è il campo di sviluppo informatico, ovvero la spiegazione del programma Snakemake, le cui funzionalità concendono ampie possibilità sulla gestione delle risorse offerte dalle macchine.

Il terzo è il campo dei dispositivi low power, accompagnati dalla descrizione di coloro che sono stati adoperati per la computazione.

La combinazione tra tali differenti ambiti ha quindi fornito i sufficienti elementi per proseguire con la costruzione e la seguente attuazione del programma; i quali saranno affrontati nel prossimo capitolo.

1.1 La ricerca delle mutazioni genetiche

Questa sezione si occuperà di dare un'idea generale su un ramo degli studi riguardanti le mutazioni genetiche che sviluppano 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 ora 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, come ad esempio la resistenza alla chemioterapia. 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 o sono proprio differenti. Questi algoritmi elaborano i dati sperimentali grezzi attraverso vari processi che, generalmente, sfruttano 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 gruppi di ricerca 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 moderne tecniche con cui è analizzato attualmente il materiale genetico e le caratteristiche generali di questi procedimenti. Inoltre, sarà sottolineato l'impatto della fisica nei campi della biologia e della medicina.

1.1.1 L'analisi del DNA

Lo studio sull'eredità genetica ha inizio con gli studi di Gregory Mendel nella seconda metà del diciannovesimo secolo e, nella prima metà del secolo seguente, un serie di contributi, tra cui la fotografia della doppia elica da parte di Roselind Franklin, pongono le basi alla moderna ricerca genetica. Il contributo essenziale è 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 hanno subito 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 hanno avviato 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 è la decodifica del DNA umano che comprende principalemente la determinazioni 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 ha cercato di analizzare anche i genomi di esseri viventi valutati più semplici, come ad esempio le mosche.

Molte tecniche per la trattazione del materiale genetico hanno partecipato 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 è concluso con successo, essendo stato decodificato il 99% del genoma umano e visto che gli studi erano stati 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 di un esaminazione estremamente precisa, ha ottenuto l'intero genoma umano in circa tredici anni sostenuto da un investimento che si aggira attorno ai $ 2.7 miliardi. Questa nuova generazione di tecnologie, che sfrutta la parallelizzazione di processi su scale microscopiche([7]), 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, la quale è una branca della fisica che ha subito nell'ultimo ventennio un notevole progresso e che ha acquistato un significato oramai fondamentale nella ricerca medica e biologica.

Un fattore importante che supporta lo sviluppo di tecnologie più veloci è l'esistenza del genoma di riferimento prodotto dal HGP, che permette il confronto di sole certe regioni senza quindi dover estrarre l'intero codice genetico. Un ulteriore contributo è dato anche dalla contemporanea crescita delle aree che cooperano con la biologia, come ad esempio le tecniche di computazione.

Questi strumenti sono più conveniente del metodo Sanger, il quale rimane comunque artefice di un eccellente connubio tra qualità e quantità di letture(read-lenght), ma devono trovare l'idoneo compromesso tra l'accuratezza dei risultati e la diminuzione delle spese temporali ed economiche. Infatti, nello studio di sequenze a piccola scala la tecnologia di Sanger resta tuttora una tra le più affidabili, mentre su letture ad ordini superiori è atteso che le applicazioni del NGS possano vantare in futuro la miglior proficuità.

Pur adoperando tipi di tecnologia diversi, il funzionamento di un generale algoritmo del NGS prevede sempre alcuni passaggi, tra cui l'allineamento delle letture con un riferimento e l'investigazione sulle possibili variazioni. La difficoltà nello scegliere quale tra le diverse opzioni disponibili usare può è superata dal confronto tra i diversi risultati finali dei procedimenti, ovviamente in relazione agli scopi preposti. Essi includono la qualità dei risultati, 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 nella ricerca presentatata 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([3]). 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 nei cromosomi determinanti il sesso dell'individuo e che si trasmettono di generazione in generazione. Le seconde, invece, rappresentano le variazioni negli altri cromosomi contenenti le caratteristiche del singolo soggetto. 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. E' 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. Un'osservazione da sottolineare è che i due metodi scovano pure 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 ovviamente 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 range 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 scovare varianti non trovate da MuTect senza dover rinunciare alla specificità e sensibilità.

Visti i risultati positivi ricavati dall'algoritmo, la ricerca sostiene con un fiducia 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.

In modo da 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.

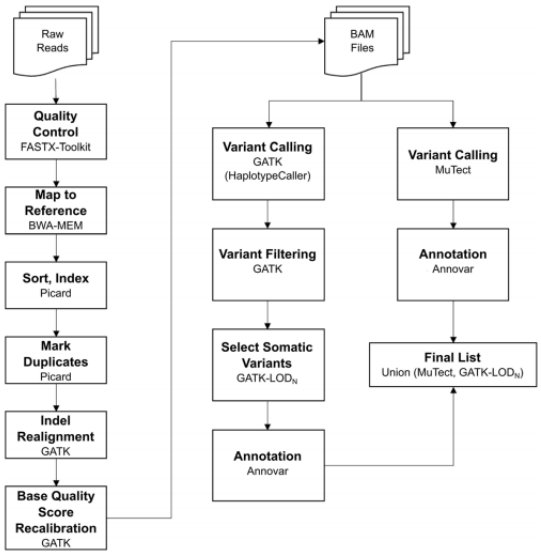


Figura 1.1: GATK-LOD$_n$ Workflow

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([5]). 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 poichè non è stato ritenuto necessario per la comprensione di questo scritto.

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 del multikilobase. 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 ottimizzazione della memoria disposte da GATK. E' 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_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([2]). Infatti, quest'ultimi fornivano livelli di sensibilità e specificità considerati insoddisfacenti per una sufficiente comprensione delle anomalie.

Il sistema focalizza gli sforzi 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 pure 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 non perfettamente. 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 $\mathrm{LOD_T}$, 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 $\mathrm{LOD_n}$, implementato nel metodo GATK-$\mathrm{LOD_n}$, 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 relativià 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([8]). 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([6]).

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([9]). 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 sui network

La teoria dei network è materia prevalentemente dei fisici, dove per network si intende uno schema a base matematica 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. Sono proprio i meccanismi dei network infatti la causa per cui non è più indispensabile ricostruire il codice genetico puntualmente, dato che risulta conveniente ricombinare tante

stringhe casuali dello stesso filamento di DNA(shotgun sequencing). Questa novità è ciò che contraddistingue il metodo Sanger dalle nuove tecniche del NGS e che permette a quest'ultime di ricavare i genomi di un individuo in tempi decisamente più ristretti.

La ricostruzione del genoma avviene, in particolare, a partire dal 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 la teoria dei network da perseguire. Il 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 dispositivi 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 del flusso di lavoro che semplifica l'esecuzione di algoritmi particolarmente complessi grazie all'utilizzo di un ambiente di sviluppo nitido ed intuitivo([4]). 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 e la sua formazione è fortemente influenzata dal noto tool Make del sistema operativo Linux. Visto il largo uso di Python e l'affermata conoscenza di Make nella comunità di sviluppatori, queste due caratteristiche rendono questo strumento ancora più affidabile e prossimo agli interessi dei ricercatori.

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 dove i lavori sono rappresentati da nodi e dove le dipendenze sono semplicemente descritte da linee congiungenti. Dalla forma assunta da tale struttura è facile osservare che il lavoro degli sviluppatori del DAG è stato basato sulla teoria dei network e che l'applicazione di questa consente una descrizione semplice e compatta del sistema. L'utilità della schematizzazione consente anche di studiare quali dei lavori da eseguire 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 desiderati. 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 speci-

ficare singolarmente per ogni regola anche le risorse disponibili, aggiungendo la voce *resources* nel dominio.

Altre due notevoli 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.

Una peculiarietà particolarmente utile di Snakemake è la capacità di riprendere l'esecuzione di una pipeline interrotta, esattamente dalla regola in cui si trovava al momento dell'interruzione. Ciò è garantito dal fatto che il sistema attiva una regola solo se è assente il relativo output e, quindi, solo coloro che non sono state ultimate , o rimaste intoccate, prendono parte alla nuova esecuzione senza ripartire dall'inizio. Chiaramente, questo approccio è modificabile richiendo esplicitamente che alla richiesta di una nuova istanza vengano sovrascritti i file di output e quindi acconsentita una diversa realizzazione.

Nella pipeline creata nell'ambito di questa ricerca, sono stati implementati 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, che è l'eccezionale capacità di gestire l'esecuzione della pipeline sulle risorse tecniche dei dispositivi 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 dall'apparecchio 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 scambiata, 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 I dispositivi low power

I gruppi di ricerca che insistono nel proporre metodi avanzati per l'analisi dei big data in campo biomedico, non si limitano ad ottimizzare la sola componente software, bensì affrontano pure la scelta del tipo di componente hardware da adoperare. Lo sviluppo di applicazioni sempre più raffinate ha richiesto infatti, nel corso del tempo, un progresso tecnologico che procedesse di pari passo e che permettesse di soddisfare in maniera esaustiva le richieste dei ricercatori. Normalmente, i macchinari prediletti sono quelli più potenti e performanti che, pur essendo abili nel terminare i lavori previsti, hanno il difetto principale di consumare massicce quantità di energia.

Il dispendio di energia di uno strumento elettrico avviene, in particolare, alla conversione da corrente analogica (AC) a corrente digitale (DC): questo passaggio implica un'innalzamento della temperatura che causa lo sprigionamento di calore nell'ambiente circostante, da cui si ha una perdita di lavoro. Tale effetto equivale ad uno spreco di energia ed esso si riproduce in ogni macchinario elettrico. Il contenimento del calore, e quindi in generale una diminuzione dello spreco energetico, consente non solo una diminuzione dei costi per il sostentamento dell'apparecchio, che può penalizzare le grandi aziende, ma anche il miglioramento della resa del dispositivo. Difatti, il contenimento del calore implica la presenza di sistemi di raffreddamento che consumano grandi quantità di energia e, nel complesso, ciò ricade su ulteriore spesa degli esercenti. Un dato significativo relativo alle spese di raffreddamento è ad esempio sostenuto dal CNAF di Bologna(National Center for Frame Analysis) dove il costo annuale per il dispendio elettrico è dell'ordine del milione di euro.

L'altro difetto predomaninate è il costo che le macchine tradizionali richiedono, dato che spesso equivalgono a decine di migliaia di euro, come dimostrato ad esempio dall'Università Alma Mater Studiorum di Bologna(UNIBO) che per gli apparecchi a disposizione ha investito un cifra prossima ai €25 000 per singola macchina. In aggiunta, un altro fattore penalizzante è dato dalla mancanza di scalabilità di aggiornare questi dispositivi, dato che ciò consisterebbe nell'acquisto di un'altra macchina più evoluta.

Tali aspetti negativi hanno spinto vari team di studiosi ad interessarsi a differenti risorse computazionali, tra cui quella relativa ai dispositivi low power. I macchinari low power sono strumenti che mirano ad abbassare il consumo energetico e soprattutto ad incrementare l'efficienza.

Questi strumenti, però, non sono una vera novità nel campo tecnologico, dato che alcuni oggetti, come gli orologi, possono essere già considerati tali. La vera innovazione è l'impiego di strumenti di questo calibro nei processi computazionali sui big data. Ovviamente, la differenza di performance tra

$Hostname$	$Hostname(10Gb/s)$	ISA	$Microarchitecture(Platform)/litho$
$xeond$	$xeond$	$x86_64$	$Broadwell/14nm$
$avoton$	$avoton$	$x86_64$	$Silvermont(Avoton)/22nm$
$n3700-00,2$	–	$x86_64$	$Airmont(Braswell)/14nm$

Tabella 1.1

CPU	$Freq(GHz)$	$Cores$	$Cache(MB)$
Xeon D-1540	2.0(2.60)	8(16)	12
Atom C2750	2.40(2.60)	8	4
Pentium N3700	1.60(2.40)	4	2

Tabella 1.2

gli apparecchi low power e i potenti macchinari utilizzati giornalmente è abissale. Nonostante ciò, la cooperazione fra diversi dispositivi di tale genere, il cluster, potrebbe ambire a sostituire gli apparecchi moderni. Questa tesi descrive proprio una ricerca introduttiva ad una questione così rilevante, che, difatti, considera la computazione non sul cluster ma sulle singole macchine: i nodi low power.

I dispositivi coinvolti in questi studio sono stati scelti con caratteristiche tecniche differenti, in modo da evidenziare fra taluni una crescita delle prestazioni. In accordo con il centro nazionale per la ricerca e lo sviluppo sulle tecnologie per l'informazione e la comunicazione (CNAF) dell'Istituto Nazionale di Fisica Nucleare, avente sede a Bologna, sono state utilizzate alcuni server appartenti al bastion dell'INFN. I dettagli sui nodi dei cluster utilizzati per la computazioni sono esposti nelle tabelle sottostante.

$Memory(GB)$	$Ubuntu$	TDP
16	16.04.2	$45W$
16	16.04.1	$20W$
8	16.04.2	$6W$

Tabella 1.3

Capitolo 2

Procedimento

Questo capitolo ha lo scopo di presentare il procedimento seguito per valutare l'ottimizzazione di una pipeline per l'analisi computazionale biofisica su apparecchi 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 Il corpo delle analisi

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 dispositivi.

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 e matplotlib, che sono necessarie per una fase di semplificazione dei benchmark e per le future 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 condiviso dall'Azienda Ospedaliero-Universitaria Sant'Orsola-Malpighi(Bologna, Italia), in accordo con i canoni dettati dalla dichiarazione di Helsinki.

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 contiene il fulcro del procedimento, quindi illustra il tipo di operazioni svolte dal metodo per lo studio 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 analisi 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. Spesso però, la rilevazione non è singola ma, come nel caso di questo progetto, duplice e, quindi, i dati del paziente sono suddivisi in due file fastq accoppiati.

Brevemente, ogni singola lettura contenuta nel fastq è descritta da quattro linee aventi i seguenti ruoli: la prima linea marca la sequenza, la seconda mostra la sequenza, la terza identifica il punteggio di qualità e la quarta riporta tale punteggio.

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, senza entrare in dettagli tecnici, 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 range 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 avere inizio l'applicazione dell'algoritmo di ricostruzione genetica, il quale è interamento contenuto nell'apposito Snakefile. E' 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 Snakemake Cap1**).

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 facilitato sia l'indagine sulla scalabilità che il confronto tra i diversi apparecchi adoperati, così da poter trarre rapidamente le prime conclusioni.

I primi step consistono nell'indicizzazione del genoma umano di riferimento per i software che partecipano al sequenziamento. Questi 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.

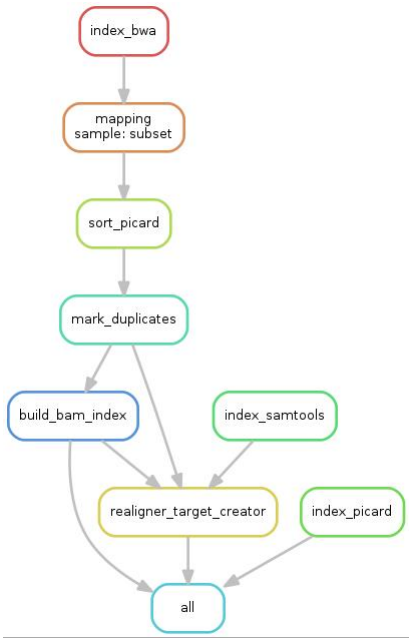


Figura 2.1: Workflow

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 nel **capitolo Snakemake**, 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 avente funzione di etichetta; in particolare, la forma è la seguente.

benchmark_{name}_subset_{sample}_n_sim_{n_sim}_cputype_{cpu_type}_thrs

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(presentate nella sezione **benchmark** della sezione 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.

E' stato scelto di gestire queste informazioni come parametri per non irrigidire il programma, visto che gli apparecchi 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* co-

me 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 Studio statistico

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, e dotandosi di specifiche librerie per la statistica, tra cui: pandas, matplotlib e seaborn.

Il trattamento dei dati ha coinvolto inizialmente le funzioni sui dataframe, procurate da 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 range di ogni subset e la quarta il logaritmo del tempo di esecuzione. Da questo unico contenitore sono state estratte progressivamente le tabelle utili ai vari studi.

Il capitolo è suddiviso in quattro sezione, dove le prime tre corrispondono alle caratteristiche principali su cui è stata condotta l'indagine statistica: il tempo di esecuzione, il max_rss e i processi di input ed output. 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 delle cpu adoperata.

Inizialmente è stata derivata dal dataframe generale una tabella in cui ogni lavoro svolto 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 sequenza, la mappatura, l'ordinamento tramite picard, la rilevazione dei duplicati, la creazione dei file bam e il riallineamento. Al contrario, i processi indipendenti sono le indicizzazioni dello human reference per bwa, per picard e per samtools.

In seguito è stata graficata la dipendenza del tempo di esecuzione di ogni regola rispetto al range del subset, portando sullo stesso grafico anche l'andamento relativo al tipo di dispositivo impiegato. In questa maniera è stato possibile verificare le differenze di prestazione tra i dispositivi e adattare la relazione matematica più idonea tra il tempo e il range per ciascuno di essi.

La funzione usata per i grafici è stata lmplot di seaborn, i cui attributi hanno fornito le opzioni migliori sia per la visualizzazione del grafico che per la ricerca della curva di fitting più adeguata.

Una volta osservato l'andamento del tempo per ogni singola regola, l'attenzione è stata rivolta al tempo totale impiegato sempre dalle stesse regole. Per calcolare il tempo totale sono state sfruttate le proprietà della funzione groupby proveniente dalla libreria numpy, la quale ha consentito la creazione di una nuova tabella avente la colonna per il tempo complessivo.

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

Terminato lo studio sui lavori dipendenti dall'intervallo dei dati, sono state brevemente considerate anche le regole indipendenti. L'unica elaborazione effettuata è stata tracciare su un grafico i tempi impiegati per il completamento di ogni regola e ciò è stato descritto, sulla stessa figura, per ogni tipo di cpu.

L'ultima analisi conseguita, relativa alle tempistiche, è stata esaminare il comportamento dei tempi di esecuzione tra diversi intervalli di lettura con diversi estremi ma con lo stesso range. Ciò è stato fatto per ognuna delle regole e per ciascuna delle macchine. Così operando, è possibile osservare se

la durata di completamento dei lavori è influenzata dal contenuto del subset processato e con quale entità ciò influisce.

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

2.2.2 Analisi sul max rss

La rss o *Resident Set Size* è la memoria fisica, non estesa, che un singolo processo sfrutta e in Snakemake questa grandezza è espressa, nel file di benchmarking, in ordine di megabytes(MB). In realtà, Snakemake non fa altro che ottenere le informazioni sulla memoria dal tool psutil e successivamente converte talune in MB, dato che psutil le dispone in bytes.

I dettagli su rss è di fondamentale importantanza per un corretto sviluppo di un 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 in modo da realizzare una parallelizzazione efficiente.

Lo studio statistico condotto è stato interessato, come per le tempistiche, ad osservare il comportamento della memoria rss in base alle regole, ai subsets e ai diversi apparecchi. Perciò sono stati elaborati i grafici per ogni regola in funzione del range del subset, quelli relativi alle regole indipendenti dai dati e il comportamento per lo stesso range ma diverso intervallo, sempre considerando tutti i dispositivi coinvolti. Non è stato descritto l'utilizzo complessivo della cpu per ogni simulazione, ovvero la somma tra gli rss, perchè le informazioni essenziali per la parallelizzazione provengono dagli specifici utilizzi per i singoli lavori.

2.2.3 Analisi sui processi di input ed output

Due ulteriori dati che Snakemake trascrive nel benchmark sono ottenuti da psutil, specificatamente dalla funzione `disk_io_counters`, e sono il numero di bytes letti e quelli scritti dal processo. In particolare, questi sono rispettivamente nominati `io_in` e `io_out` e, come per la memoria rss, sono convertiti in megabytes da Snakemake.

Queste due proprietà garantiscono una conoscenza più approfondita di come la cpu partecipa alla computazione, evidenziando con quale intensità, espressa in MB, essa è coinvolta nelle distinte operazioni. Così come l'analisi di rss, la statistica sulle fasi di input ed output delinea gli effetti che i lavori

causano sull'apparecchio. Da queste informazioni si può trarre una valutazione su quanto al massimo è possibile servirsi della cpu e fino a che punto l'utilizzo di essa è efficace.

Le analisi realizzate sono state le stesse che per la memoria rss e quindi sono stati studiati gli andamenti dell'input e dell'output per le regole dipendenti dai subset e pure quelle indipendenti. Ugualmente ai casi precedenti sono state valutate anche le dipendenze per il tipo di cpu e sono stati confrontati intervalli diversi aventi identico range. Non è stato presa in esame il valore complessivo per l'input e output nell'intero arco dell'esecuzione perchè da tale dato totale non è possibile ricavare alcuna informazione proficua.

2.2.4 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 dispositivi low power adoperati, che sono stati mostrati nel paragrafo 1.3 , sono stati impiegati in base alle loro diverse potenze computazionali e di seguito sono sono esposti i tipi di subset su cui hanno operato.

Tutti le macchine hanno processato gli stessi subset di dati con un range di iniziale 5000 fino a 3 milioni, passando per i multipli di diecimila, di centomila e di un milione. In questo modo è stato possibile studiare l'attitudine delle macchine a processare dati identici con larghezza crescente e come questo incremento condiziona l'uso della cpu. In dettaglio, queste simulazioni sono state ripetute dieci volte per il dispositivo Xeon D-1540, che è il più performante, e cinque volte per gli altri nodi, dato che i tempi di esaurimento del processo coprivano già per tali piccoli gruppi di dati un arco di tempo considerevole.

Solo sulla macchina montane Xeon sono stati portati avanti i subset fino ad un range di 9 milioni, sempre con un intervallo di 1 milioni tra due subset consecutivi, e ciò per confermare gli andamenti tratti dalle analisi precedenti. Non sono stati fatti proseguire anche gli altri apparecchi perchè, oltre i prolungati tempi di esecuzione, 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 sufficienti i dati fino a 3 milioni, solo la macchina più performante è andata oltre per affermare le stime ottenute.

Infine, sono stati estratti vari intervalli con lo stesso range per verificare se il contenuto dei dati influenza significativamente le computazioni. I due range trattati sono stati diecimila e centomila, e per ricavare intervalli diversi sono

stati estratti i relativi subsets in posizioni differenti nel materiale genetico grezzo a disposizione.

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 tre caratteristiche su cui è stata sviluppata l'analisi dei dati. Saranno presentate quindi una sezione relativa ai tempi di esecuzione, una sulla memoria rss e una riguardo ai processi di lettura e scrittura. La descrizione delle relazioni più interessanti sarà, inoltre, accompagnata dall'utilizzo di alcuni tra i grafici formati durante l'indagine statistica.

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 range. 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 diviso questo ramo dell'analisi in due parti, una relativo a quei processi che non dipendono dal tipo di dati considerato e uno relativo a quelli che invece dipendono. Considerando le prime, è evidente dalla figura 3.1 come l'unico processo significativo è l'indicizzazione dello human reference per BWA, mentre le rimanenti non influiscono in alcun modo. In più, la differenza tra le potenze computazionali è netta già da questa figura, dato che tra l'apparecchio migliore e il peggiore vi è uno scarto di più di un'ora.

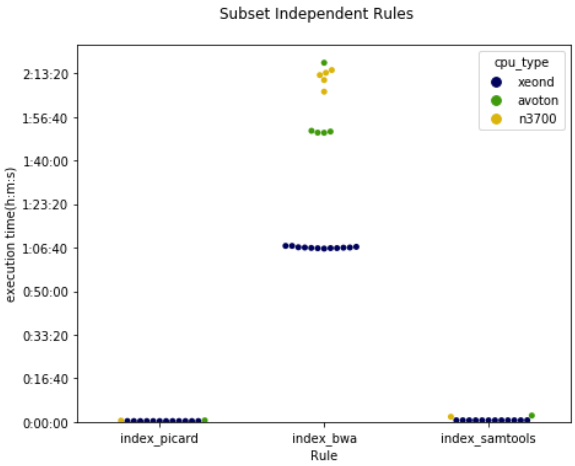
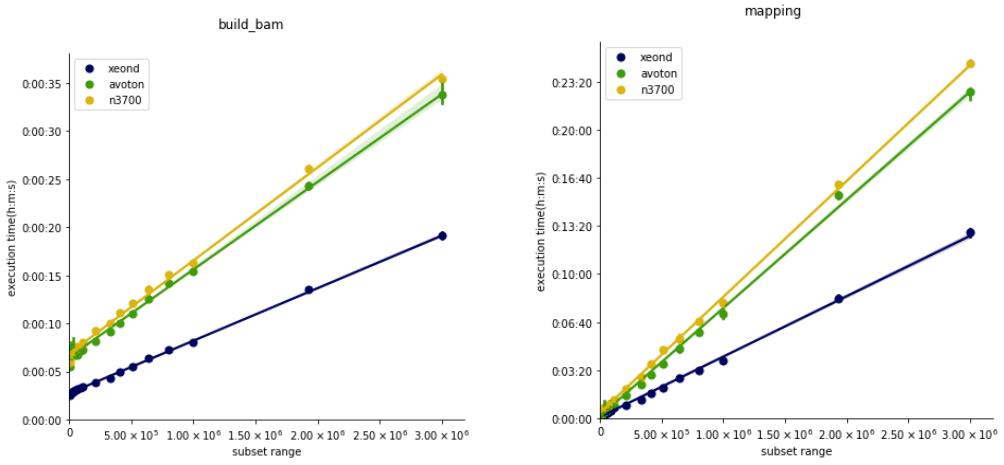


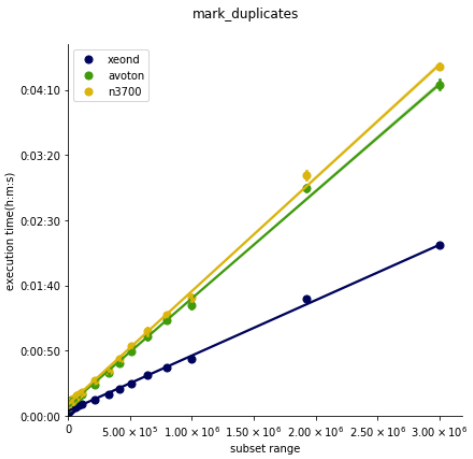
Figura 3.1: Tempi di esecuzione per le regole indipendenti dal subset

Nonostante la durata per l'indicizzazione sia già notevole, il vantaggio di questa fase è che può essere riprodotta una sola volta per tutti i successivi lavori, a causa della sua indipendeza.

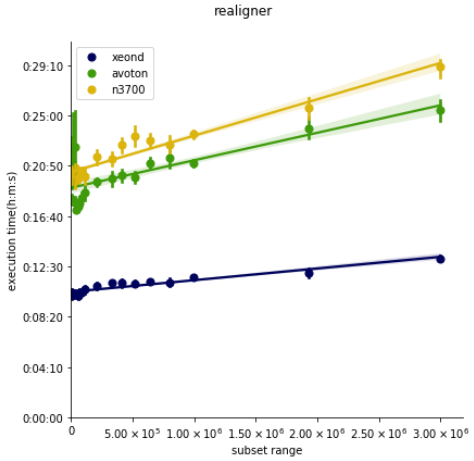
Passando alle regole dipendenti, i grafici riportati nella figura 3.0 mostrano i vari andamenti per quei subset citati in precedenza con letture di range massimo 3 milioni. Le fasi del procedimento che esauriscono più tempo sono la mappatura, che incrementa velocemente, e il riallineamento, che al contrario cresce lentamente; mentre le altre regole aumentano ma impiegano tempi sempre inferiori a 5 minuti.



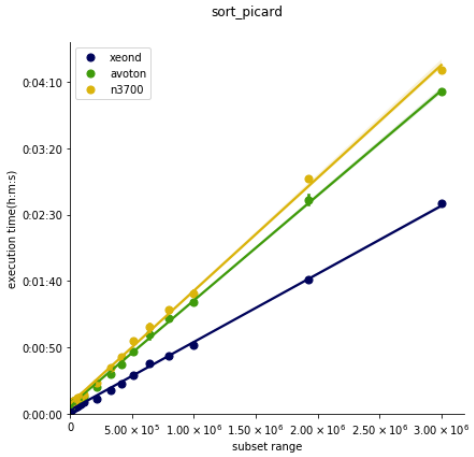
(a) *Build BAM* (b) *Mapping*



(c) *Mark Duplicates*



(d) *Realigner*



(e) *Sort Picard*

Figura 3.0

Ciascun gruppo di dati è stato sottoposto a regressione lineare e sono stati pure calcolati i coefficienti di correlazione, i quali hanno evidenziato come il tempo di esecuzione di ogni regola cresce linearmente rispetto al range del subset. L'unica regola che mostra una particolarità è la fase di riallineamento(Figura3.1d), dove soprattutto per la cpu Avoton i tempi oscillano quando i subset sono piccoli. In dettaglio, i coefficienti di correlazione in questa regola sono per le cpu Xeon, Atom(avoton) e Pentium n3700, rispettivamente: 0.812, 0.656 e 0.923. Tale conseguenza è data da due fattori, dove il primo è la già citata oscillazione dei tempi per piccole variazione del range

degli intervalli e il secondo è una caduta di performance dovute ad agenti scionosciuti, presumibilmente di natura tecnica. Quest'ultimo effetto è rilevabile visibilmente per avoton(linea verde nel grafico 3.1d), dove il fenomeno di oscillazione iniziale è ben nitido. Nonostante ciò, visto che il numero di dati che è abbondantemente superiore al centinaio, il coefficiente di Pearson determina comunque che i dati siano disposti linearmente.

E' possibile visualizzare grazie al grafico () l'andamento della macchina più performante, nominata xeond, rispetto ad un esempio di macchina tradizionale, classical, sull'intero range da 0 a 9 milioni.

3.1.2 Durata complessiva

Un interessante aspetto rilevato è stato il tempo complessivo per concludere l'intero procedimento dei passaggi per il sequenziamento che dipendono dal subset, quindi escludendo soprattuto l'indexing per bwa. Il grafico 3.1 è riferito ai subset con massimo range di 3 milioni e, anche in questo caso, la regressione lineare è confermata dai coefficienti di correlazione che superano abbondantemente lo 0.95, per ciascuna macchina.

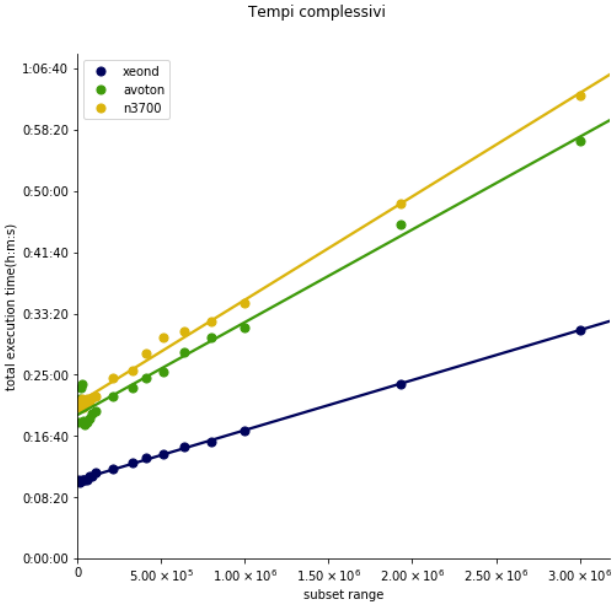
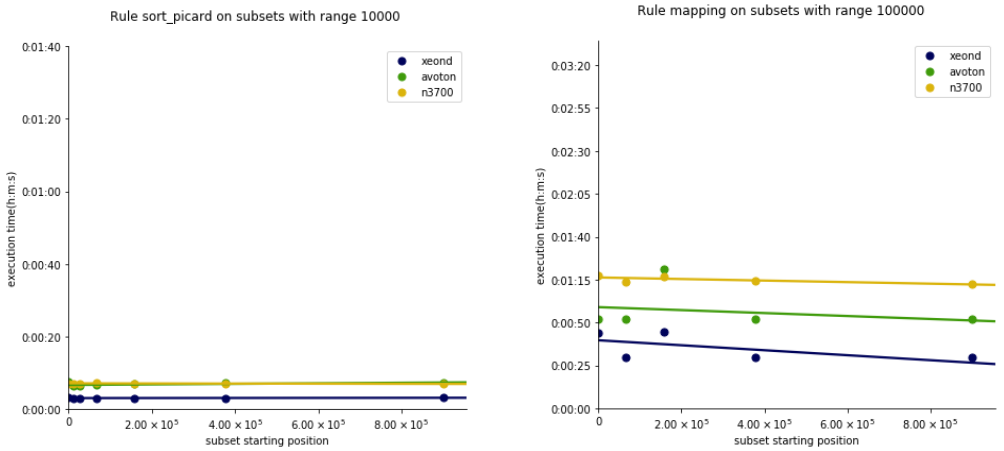


Figura 3.1: Tempi di esecuzione per le regole indipendenti dal subset

3.1.3 Tempi per stessi range

Un approfondimento è richiesto per valutare quanto il contenuto dei dati influenza la tenuta temporale e, in conseguenza, sono state considerate varie posizioni iniziali di estrazioni dal materiale genetico grezzo, lasciando invariato la lunghezza dei dati. Questa operazione è stata fatta, come già indicato nel paragrafo 2.2.4, su intervalli da diecimila e centomila letture.

Sono riportati nella figura 3.2 due grafici rappresentativi dei due andamenti che predominano questa analisi.



(a) *Sorting per subset da diecimila* (b) *Mapping per subset da centomila*

Figura 3.2

Il primo caso è quello di un comportamento pressochè costante, che è il più intuitivo e che si manifesta per la maggior parte dei grafici. Il secondo rappresenta, invece, un andamento che si allontana da una costanza ma che assume più i contorni di un fenomeno stocastico rispetto al range del subset. Dalla seconda figura si vede proprio l'effetto di allontanamento dalla retta di fit e già si vede come questa sia comunque leggermente inclinata senza un parvente motivo.

E' possibile ipotizzare che per le regole che elaborano nello specifico le misure nei subset (la mappatura e il riallineamento), i lavori si adattino al tipo di dati e che ciò comporti un'oscillazione del tempo non ben prestabilita.

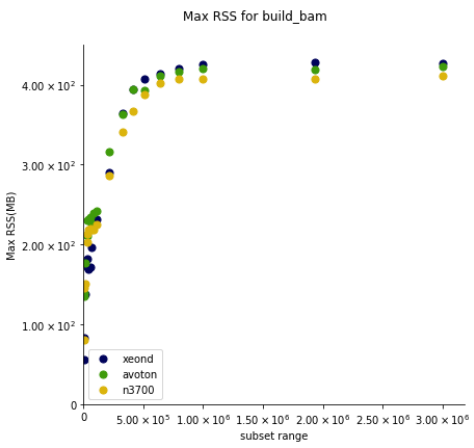
3.2 Memoria RSS

La memoria rss è stata studiata analogamente al tempo eccetto che per lo studio di una memoria complessiva dell'intero processo. Ciò implica che

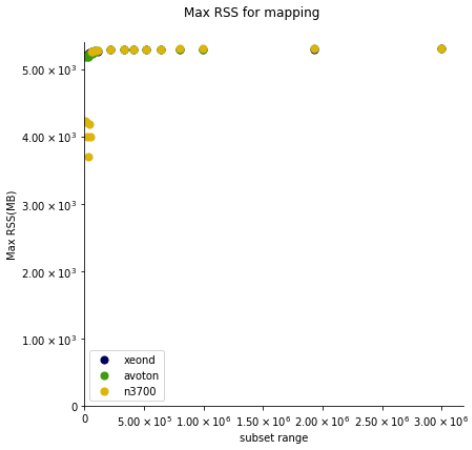
sono state analizzate in un primo momento le occupazioni della memoria per ognuna delle regole e che poi è stato tentato di definire l'andamento per diversi intervalli con stesso range.

3.2.1 RSS e regole

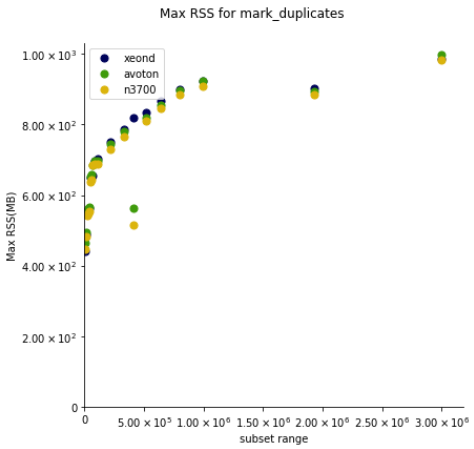
Le informazioni sui vari comportamenti sono tratte dai grafici in figura 3.1, per i quali sono necessarie delle descrizioni specifiche, dato che non si ha un andamento generale.



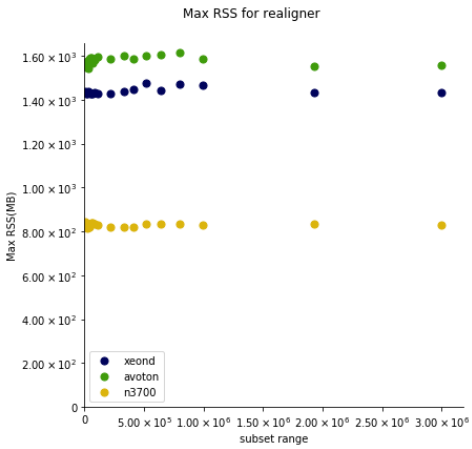
(a) *Build BAM*



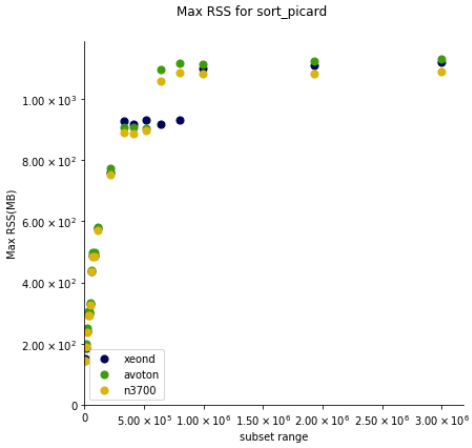
(b) *Mapping*



(c) *Mark Duplicates*



(d) *Realigner*



(e) *Sort Picard*

Figura 3.1

E' evidente che regole di marcamento dei duplicati, di riordimento per picard e di formazione del file BAM, sono pressochè adattabili ad un logaritmo. Ciò indica una saturazione della memoria per un certo range di subset raggiunto e questo sarà un fattore rilevante per la determinazione di un'uso efficace della parallelizzazione.

Le altre due regole che si soffermano specificatamente nei dati, la mappatura e il riallineamento, manifestano una costanza di uso della memoria ma con un notevole distinzione. Nel caso del mapping tutte e tre le macchine lavorano alla stessa intensità, indicando una saturazione generale dell'utilizzo della memoria, mentre il riallineamento distingue una costanza singolare per ognuna delle cpu.

Allo stesso modo che per i tempi, è stato ottenuto pure il grafico(3.2) per le regole indipendenti dal set di dati, dove a differenza dell'impiego temporale anche l'indicizzazione dello human reference per picard consuma una parte della memoria.

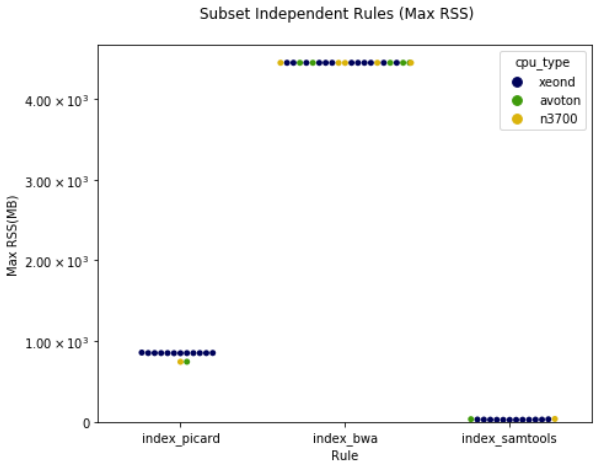
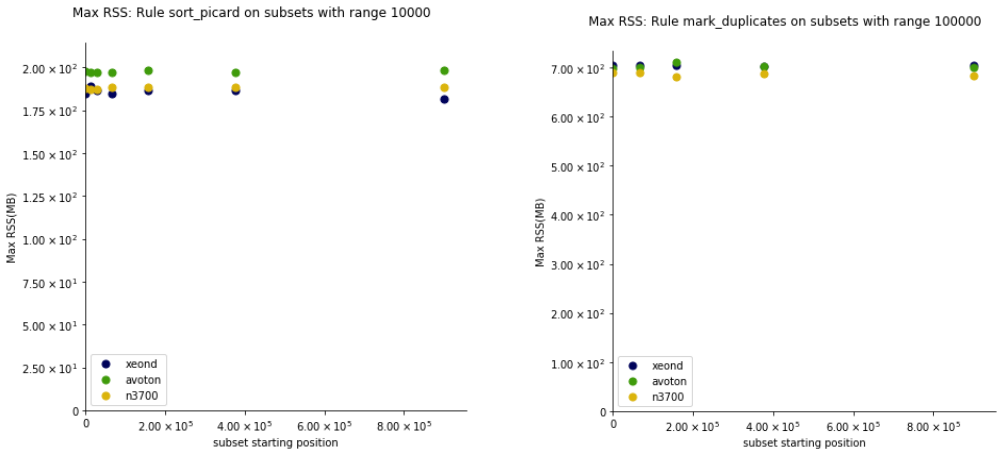


Figura 3.2: Max RSS per le regole indipendenti dal subset

3.2.2 RSS per stessi range

Lo studio sulla memoria per intervalli diversi su stessi range ha prodotto due comportamenti caratteristici, che sono stati posti in figura 3.4.



(a) *Sorting per subset da diecimila* (b) *Mark duplicates per subset da centomila*

Figura 3.3

Nella prima figura i dati oscillano leggermente rispetto ad un valore costante, ma con tali valori diversi per ciascuna cpu; mentre nella seconda, la linea su cui i dati sembrano adattarsi è la stessa per ogni apparecchio. Ciò indica come per alcuni lavori i dispositivi si comportino allo stesso modo, o saturano completamente, e per altri la distinzione è più netta. In generale

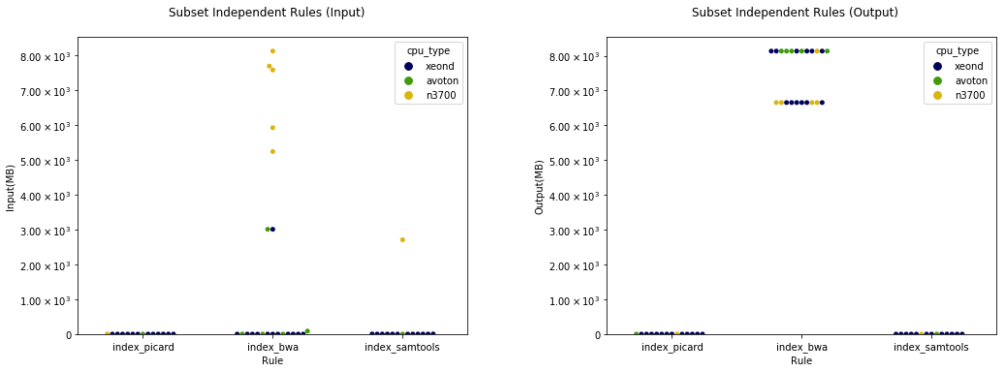
però, diversamente dai tempi di esecuzione, l'occupazione della memoria rss tende a non dipendere dal contenuto dei subset aventi la stessa grandezza, dato che tende a rimanere costante per ogni regola.

3.3 Processi di I-O

Gli ultimi risultati sono stati ottenuti dall'indagine sui processi di input e output coinvolti nel completamento di ogni lavoro. Questa sezione è suddivisa in due parti che determinano come la lettura e la scrittura sono dipendenti dal set di dati e se il contenuto di tali dati incide sull'uso della macchina per l'input ed l'output.

3.3.1 I-O e regole

Il primo esito riportato è per le regole indipendenti dai subset ed eccetto l'indicizzazione per BWA l'impatto è praticamente nullo.



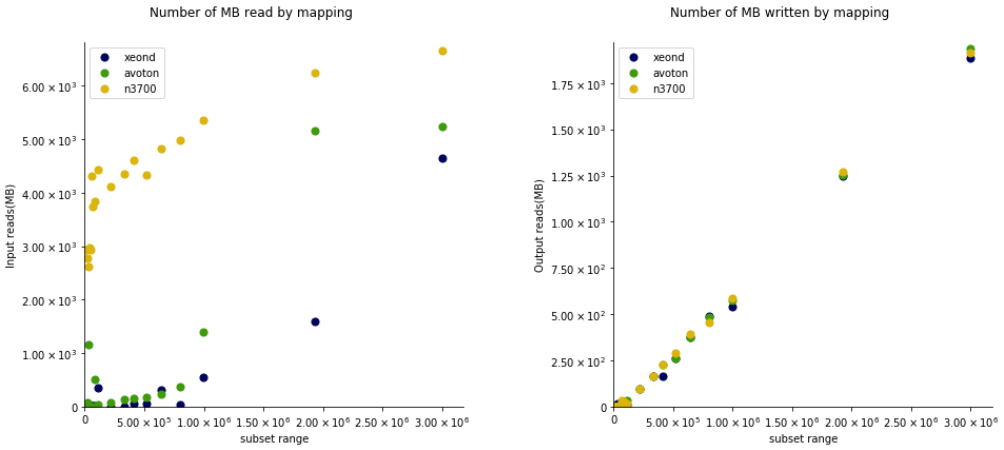
(a) *Scrittura per regole indipendenti dai subset*

(b) *Scrittura per regole indipendenti dai subset.*

Figura 3.4

Diversamente dai tempi e dalla memoria, il lavoro di indicizzazione non è sempre costante sia nel caso di input che per quello dell'output. Riguardo all'input, le macchine più performanti (Xeon e Pentium N3700) per la maggior parte delle volte non eseguono alcuna lettura mentre la rimanente varia imprevedibilmente per valori elevati. Per l'output invece, l'andamento è costante per Pentium, al contrario di Xeon e Atom che similmente occupano due valori ben distinti.

Considerando le regole che invece dipendono dal subset, sono elecanti di seguito gli andamenti dei processi per ognuna di esse.

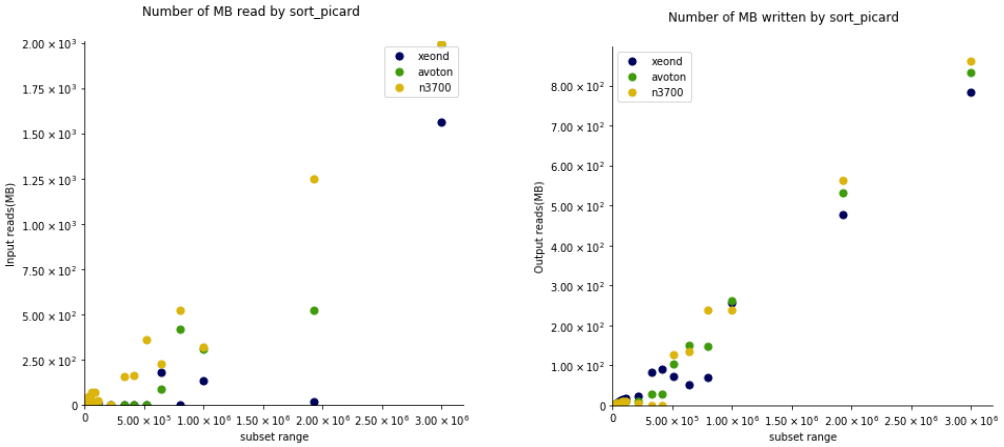


(a) *Lettura per la mappatura* (b) *Scrittura per la mappatura*

Figura 3.5

La fase di lettura per la mappatura sembra crescere esponenzialmente per le due macchine migliori mentre per Pentium N3700 la crescita ha un andamento che tende a saturare in maniera logaritmica. Gli elementi finali di Atom non seguono però un'esponenziale, suggerendo che avvenga una saturazione anche per le altre due macchine per range dei subset superiori.

La fase di scrittura, invece, mostra una andamento lineare praticamente omogeneo per tutte le macchine.

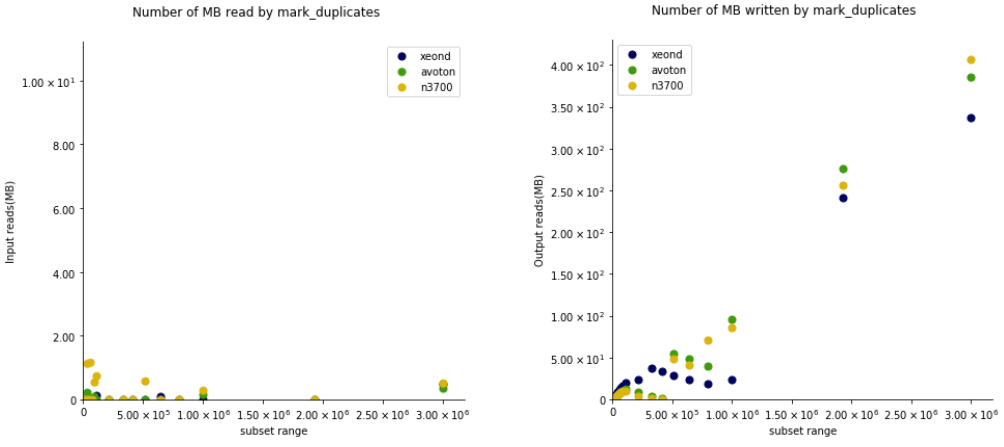


(a) *Lettura per il sorting di picard* (b) *Scrittura per il sorting di picard*

Figura 3.6

Il lavoro di riordinamento per picard è ambiguo nella lettura perchè le macchine incrementano con traiettorie non ben definibili. Ad esempio, per Pentium la traiettoria potrebbe essere adattata ad una retta ma un certo sotto gruppo di dati descrive una direzione diversa, vanificando l'ipotesi.

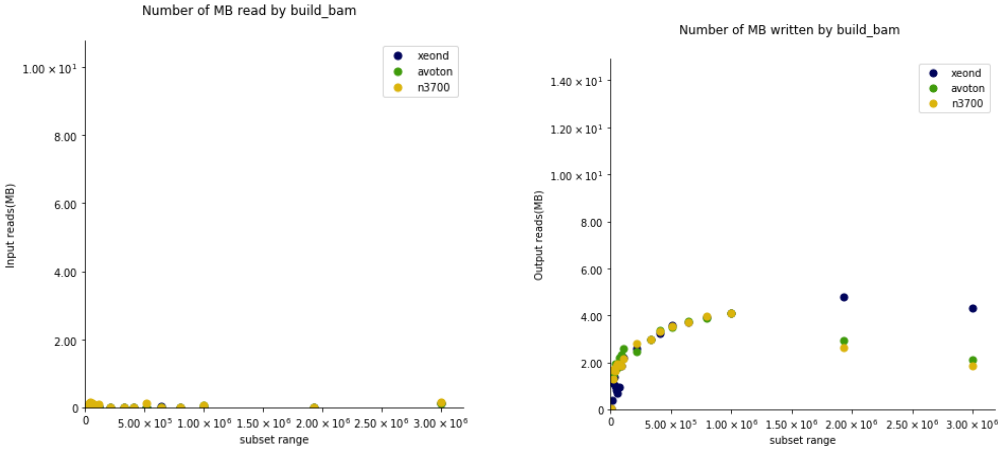
La scrittura invece inizia ad assumere contorni più chiari dopo una certa grandezza del subset, oltre il quale tende a crescere linearmente. Prima di raggiungere tale grandezza ogni macchina, per range diversi, mostra un avvallamento non motivabile direttamente da questo tipo di studio statistico.



(a) *Lettura per la marcatura dei duplicati* (b) *Scrittura per la marcatura dei duplicati*

Figura 3.7

La descrizione della marcatura dei duplicati determina che la fase di lettura è coinvolta solo marginalmente, dato che esaurisce in generale meno di $2\,MB$, mentre quella di scrittura segue la stessa attitudine che il sorting per picard. Infatti ciò è evidente soprattutto in Xeon, per cui prima di avanzare con linearità è ben delineato un ventre di una curva.

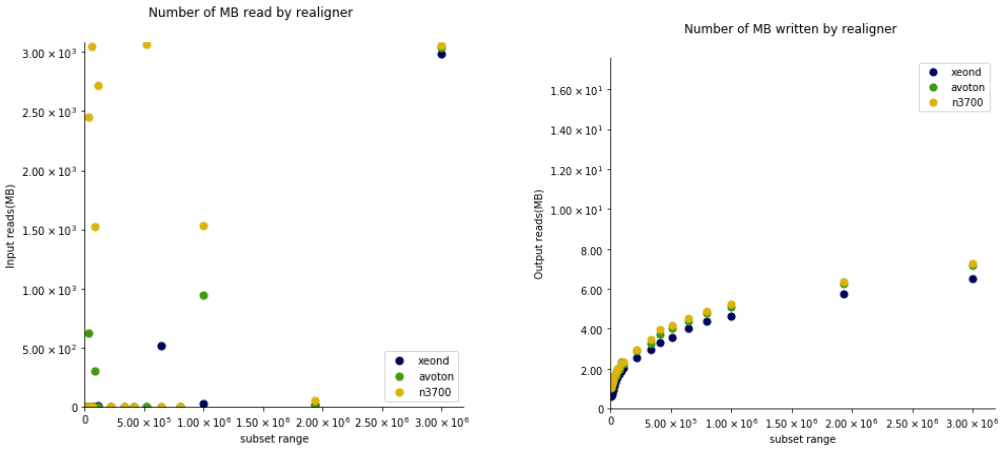


(a) *Lettura per la formazione del BAM* (b) *Scrittura per la formazione del BAM*

Figura 3.8

Riguardo al passaggio per la costruzione dei file BAM, il numbero dei megabytes letti è trascurabile, al contrario che per la scrittura che segue una crescita logaritmica fino a circa 1 milione di subset, per poi cominciare a calare vistosamente.

Le ragioni di questo calo non sono spiegabili, come nei casi precedenti, semplicemente osservando tale relazione dato che sarebbe più esauriente un approfindimento sia sulle prestazione dei macchinari che sul funzionamento dell'algoritmo di formazione del BAM.



(a) *Lettura per il riallineamento* (b) *Scrittura per il riallineamento*

Figura 3.9

In coda, il riallineamento mostra contemporaneamente il carattere meno comprensibile per la fase di lettura e quello più nitido per la fase di scrittura. I valori riportati nel grafico di input sono difficili da decifrare; non c'è ne un andamento univoco tra le cpu che tra i dati associati ad ognuna di esse. Sono presenti un numero considerevole di processi che consumano una percentuale infinitesima in input che, però, sono alternati a salti elevati che tra loro non suggeriscono alcun andamento ben fissato. Evidentemente la natura del metodo di riallineamento influenza pesantemente questa fase dei bytes, riproducendo una lettura dei bytes fortemente discontinua.

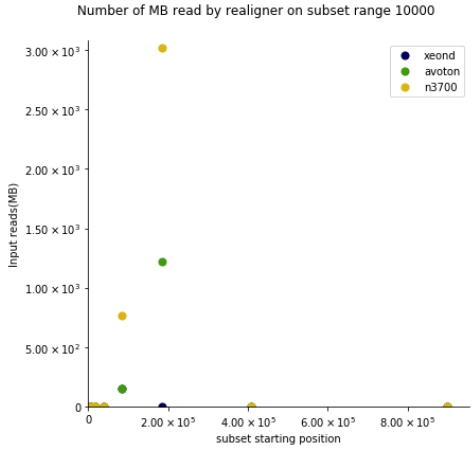
Nettamente diverso è il caso della scrittura, dove i dati tracciano una curva simile ad un logaritmo senza essere dotati di valori estranei ad essa. In più, i vari andamenti sono ordinati rispetto alla potenza computazionale delle macchine anche se le discrepanze in questo frangente sono sottili.

Prima di passare al caso degli intervalli diversi con stesso intervallo, è utile sottolineare quali sono i lavori che generalemente consumano più bytes in fase di lettura e scrittura. In entrambi i casi è il mapping ha vantare l'utilizzo maggior e all'opposto è la formazione dei BAM che necessita del minor uso delle operazioni. L'unico passaggio che presenta una netta inversione nell'uso di lettura e scrittura è, come si può controllare in figura 3.9, il riallineamento.

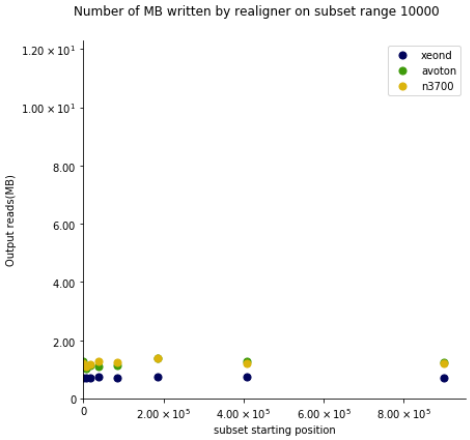
3.3.2 I-O per stessi range

L'ultima valutazione è stata eseguita, allo stesso modo che per i tempi e la memoria, su intervalli diversi con lo stesso numero di letture. Sono stati

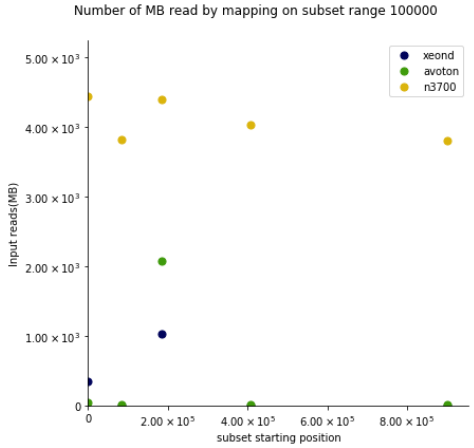
scelti i grafici sottostanti per illustrare gli andamenti più interessanti ricavati durante le analisi.



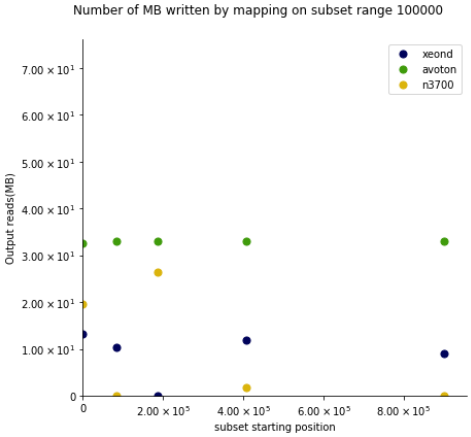
(a) *Input per realigner su range diecimila*



(b) *Output per realigner su range diecimila*



(c) *Input per mapping su range centomila*



(d) *Output per mapping su range centomila*

Figura 3.9

I passaggi di scrittura mostrano un andamento più omogeneo per tipo di subset, pressochè costante nel caso della figura 3.10b e per Atom(avoton) in 3.10b. E' interessante notare come le macchine computino con modalità diverse e che ciò causi la perdita dell'ordine delle cpu, dalla migliore alla peggiore, che caratterizzava le analisi per i tempi e la memoria.

Diverso è il caso della lettura che è rappresanto sia in 3.10a che in 3.9c. Le varie tracce non seguono andamenti ben definiti e sono spesso accompagnati

da netti salti tra le misure di lettura, i quali inducono a pensare che sia il contenuto dei subset a determinare il tipo di lettura da svolgere.

Capitolo 4

Conclusioni

I risultati esposti nel capitolo **??** permettono di delineare l'organizzazione per il futuro sviluppo di una tecnica per la parallelizzazione dei processi.

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] 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.

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

[5] 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.

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

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

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

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