UNIVERSIDAD DE CASTILLA-LA MANCHA

ESCUELA SUPERIOR DE INGENIERÍA INFORMÁTICA

MÁSTER UNIVERSITARIO EN TECNOLOGÍAS INFORMÁTICAS AVANZADAS

TRABAJO FIN DE MÁSTER

¿Aceleración de software de análisis de genoma mediante paralelismo?

Raúl Moreno Galdón

Julio de 2013

UNIVERSIDAD DE CASTILLA-LA MANCHA

ESCUELA SUPERIOR DE INGENIERÍA INFORMÁTICA

Departamento de Sistemas Informáticos

TRABAJO FIN DE MÁSTER

¿Aceleración de software de análisis de genoma mediante paralelismo?

Autor: Raúl Moreno Galdón

Director: Dr. Diego Cazorla Lopez

Codirector: Dr. Enrique Arias Antunez

Julio de 2013

A Antonio y Mª del Carmen

Resumen

El ADN y los genes moldean el organismo en el que residen, es la receta para construir el mismo. La diversidad biológica se debe a variaciones en el ADN, las cuales determinan las capacidades del organismo y la distinción entre especies. Los genes determinan las características del individuo que los posee, tanto físicas como mentales, determinando también sus enfermedades.

En bioinformática, el *descubrimiento de variantes* consiste en encontrar mutaciones y diferencias en una cadena de ADN respecto a otra cadena de ADN de referencia. El objetivo de realizar este descubrimiento es poder entender el ADN de un individuo. Esto tiene distintos ámbitos de aplicación, en medicina se podría utilizar para detectar algún tipo de enfermedad en un paciente y tratarla, tales como cáncer, trastornos, etc, determinando un estado detallado sobre su salud. También tiene aplicaciones en el ámbito de la agricultura y ganadería, de la ecología y la antropología.

Actualmente la bioinformática ofrece herramientas para realizar este tipo de análisis pero no utilizan tecnologías que permitan acelerar sus análisis usando paralelismo por ejemplo. Aunque realizan el descubrimiento de variantes, lo hacen en un tiempo que no es viable para su aplicación cotidiana, por ejemplo, en el ámbito de la medicina. Lo que se pretende por tanto es desarrollar nuevas herramientas que incorporen la funcionalidad de las que ya existen, pero utilizando tecnologías paralelas y estructuras de datos óptimas para la arquitectura en que se ejecuten. Se pretende llevar estas aplicaciones al ámbito de los sistemas *cluster*, que nos permitan realizar los análisis más rápido cuantos más recursos dispongan, utilizando eficientemente sus recursos.

El presente trabajo fin de máster aborda el estado del arte actual de las tecnologías paralelas disponibles. Ofreciendo además una visión sobre lo que se ha conseguido en el primer año de Tesis, desarrollando una de las herramientas del descubrimiento de variantes (recalibrador) utilizando estructuras de datos óptimas con el sistema de memoria.

Agradecimientos

A riesgo de no recordar a todos, agradezco a:
A mis padres Antonio y Mª del Carmen y a mi hermana Laura, por apoyarme incondicionalmente en la realización de mi Tesis, que ahora comienza, y durante todos estos años atrás.
A mis compañeros del Retics, en el I3A, José Antonio, Roberto y Nuria por su agradable acogida y ofrecimiento de ayuda desde el primer día.
A mis directores de Tesis, Diego Cazorla y Enrique Arias, por su apoyo y confianza. A José Luis Sánchez por su ayuda y disposición a colaborar cada vez que lo necesitaba.
A mis compañeros de máster Pedro, Jacinto, Miguel y Miguel Ángel por hacer del máster una experiencia más agradable.
A la Universidad de Castilla-La Mancha y al Instituto de Investigación en Informática de Albacete por el facilitamiento de sus instalaciones para la realización de este trabajo.

Raúl Moreno Galdón. Albacete, España, Junio 2013.

Índice general

1. Introducción **1**
1.1. Motivación 1
1.2. Estructura de la memoria 3

2. Asignaturas Cursadas **5**
2.1. Generación de documentos científicos 5
2.1.1. Descripción 5
2.1.2. Trabajo realizado 6
2.1.3. Relación con el tema de investigación 6
2.2. Arquitecturas de altas prestaciones 7
2.2.1. Descripción 7
2.2.2. Trabajo realizado 7
2.2.3. Relación con el tema de investigación 9
2.3. Tecnologías de red de altas prestaciones 9
2.3.1. Descripción 9
2.3.2. Trabajo realizado 10
2.3.3. Relación con el tema de investigación 10
2.4. Modelado y evaluación de sistemas 11
2.4.1. Descripción 11
2.4.2. Trabajo realizado 11
2.4.3. Relación con el tema de investigación 12
2.5. Análisis y diseño de sistemas concurrentes 12
2.5.1. Descripción 12
2.5.2. Trabajo realizado 12
2.5.3. Relación con el tema de investigación 13
2.6. Computación en clusters 13
2.6.1. Descripción 13
2.6.2. Trabajo realizado 14
2.6.3. Relación con el tema de investigación 14

3. Estado del Arte: Entornos paralelos **15**
3.1. Taxonomía de Flynn 16
3.2. Modelos de programación paralela 17
3.2.1. Espacio de memoria compartido 17
3.2.2. Paso de mensajes 20
3.2.3. Partitioned Global Address Space - PGAS 22
3.3. Conclusiones 23

4. Estado del Arte: Tratamiento de datos **25**
4.1. MapReduce 26
4.2. Hadoop 26
4.3. Hadoop Distributed File System 27
4.4. S3 28
4.5. Conclusiones 28

5. Estado del Arte: Bioinformática **29**
5.1. Bioconductor 29
5.2. Bioperl 30
5.3. Genome Analysis ToolKit 30
5.4. Bowtie 31
5.5. OpenCB 32
5.6. Conclusiones 32

6. Anteproyecto de Tesis **35**
6.1. Introducción al descubrimiento de variantes 35
6.1.1. ADN 35
6.1.2. Descripción del proceso de descubrimiento de variantes 36
6.2. Punto de partida 38
6.3. Objetivos 39
6.3.1. Objetivo general 39
6.3.2. Objetivos específicos 39
6.4. Planificación de tareas 40
6.4.1. Descripción de tareas 40
6.4.2. Situación actual 46

7. Trabajo de investigación: Recalibrador **47**
7.1. Descripción de un recalibrador 47
7.1.1. Formato *BAM/SAM* 48
7.2. Tareas completadas o en progreso 48
7.2.1. Estado del arte 48
7.2.2. Elección de tecnologías 49

7.2.3. Elección de estructuras de datos para el recalibrador . 50
7.2.4. Implementación del recalibrador 50
7.2.5. Pruebas del recalibrador 50
7.2.6. Implementación de SSE en el recalibrador 51
7.2.7. Pruebas SSE en el recalibrador 51
7.3. Herramientas utilizadas 51
7.3.1. Editor de código y compilación 51
7.3.2. Control de versiones 52
7.3.3. Depuración 52
7.4. Algoritmo de recalibrado 53
7.4.1. Proceso formal de recalibrado 53
7.4.2. Fase 1: Recogida de datos 56
7.4.3. Fase 2: Recalibrado 57
7.4.4. Estructura de datos utilizadas 58
7.4.5. Parámetros de ejecución del recalibrador 59
7.5. Evaluación del recalibrador obtenido 60
7.5.1. Evaluación del recalibrador sin SSE2 61
7.5.2. Evaluación del recalibrador con SSE2 automáticas . . . 62
7.5.3. Evaluación del recalibrador con SSE2 automáticas y manual 62
7.6. Conclusiones 63

8. Conclusiones generales **65**

Bibliografía **67**

Currículum vítae **73**

Índice de figuras

3.1. Modelo paralelo de memoria compartida. 17
3.2. Modelo paralelo *Fork-Join* usado por *OpenMP*[1]. 19
3.3. Modelo paralelo de paso de mensajes. 20
3.4. Modelo paralelo PGAS. 22

4.1. Modelo paralelo *MapReduce* para contar palabras [2]. 27

5.1. Modelo paralelo utilizado por *GATK* para el procesamiento en *cluster* [3]. 31

6.1. ¿Dónde se encuentra el ADN? [4]. 36
6.2. Representación química del ADN [4]. 37
6.3. Proceso de descubrimiento de variantes en el ADN [5]. 38
6.4. Diagrama Gant: desarrollo de la tesis de doctorado. 41

Índice de tablas

6.1. Planificación de tareas y duración. 43

7.1. Parámetros configurables del recalibrador desarrollado. 59
7.2. Detalles del sistema utilizado y las pruebas del nuevo recalibrador. 61

Capítulo 1

Introducción

Este capítulo trata de ofrecer una visión general sobre la línea de investigación que se quiere seguir durante la realización de la tesis doctoral y cuáles son las motivaciones que la impulsan. Además, se presenta un breve resumen sobre la estructura que sigue esta memoria para orientar al lector sobre sus contenidos.

1.1. Motivación

El ADN [6, 7] (ácido desoxirribonucleico) de todos los organismos vivos conocidos contiene las instrucciones genéticas usadas en su desarrollo y funcionamiento, es como una plantilla de la que saldrá el organismo final. El papel de la molécula de ADN es el de almacenar a largo plazo información sobre esa plantilla, plano o receta, como quiera llamarse, que al fin y al cabo es un *código*. Este código contiene las instrucciones necesarias para construir los componentes de las células (proteínas y moléculas de ARN [6, 7] o ácido ribonucleico) y además se transmite de generación en generación.

Un gen [6, 7] es una región de ADN que influye en una característica particular de un organismo (color de los ojos por ejemplo). Los genes son el plano para producir las proteínas, que regulan las funciones del organismo. Si un gen resultase cambiado se podría producir una falta de proteínas dando lugar a enfermedades. En otros casos podría producir efectos beneficiosos que generarían individuos más adaptados a su entorno (en esto se basa la evolución).

La búsqueda del estudio de la estructura del ADN permite por tanto

el desarrollo de multitud de herramientas tecnológicas que explotan sus propiedades para analizar su implicación en problemas concretos. Por ejemplo la caracterización de la variabilidad individual de un paciente en respuesta a un determinado fármaco, o la obtención de una característica común de un grupo de individuos de interés. Los beneficios son visibles en medicina, pero también en agricultura y ganadería (obtener carnes y cultivos de más calidad). Es posible incluso determinar la evolución que ha seguido una especie analizando las mutaciones [7] que ha sufrido su ADN y que quedan grabadas en el mismo, avanzando en el campo de la ecología y la antropología.

En cuanto a la *bioinformática*, son muchos los avances que se han realizado en los últimos años en el campo de la genómica. La tecnología de secuenciación [8] cada vez produce más datos a una escala sin precedentes, lo cual hace surgir dificultades a la hora de almacenar y procesar esos datos. Estos datos necesitan de recursos computacionales y técnicas de procesamiento que permitan obtener resultados rápidamente, permitiendo avanzar a los campos científicos que se basan en los mismos.

Se han desarrollado muchas soluciones en los últimos años destinadas al análisis del ADN, pero ninguna ha conseguido una alta calidad y completitud. Los principales problemas encontrados son:

- Normalmente solo resuelven una parte específica del análisis y fallan en otras funciones, lo que provoca el uso de diferentes programas a modo *pipeline*.
- Implementaciones pobres, con *bugs* y rendimiento muy bajo en términos de tiempo de ejecución y necesidades de memoria.
- No están preparadas para dar el salto a distintos escenarios de computación paralela como puede ser un *cluster* con varios elementos de procesamiento interconectados entre sí.
- Falta de documentación.

Estos inconvenientes se convierten en un cuello de botella cuando los precios de los recursos de cómputo bajan mientras que los datos a procesar se incrementan.

Lo que se busca por tanto en esta *tesis* es la elaboración de un conjunto de programas que permitan analizar estos datos masivos obtenidos por

los secuenciadores de ADN, que exploten al máximo las plataformas de computación de las que se puedan disponer y por tanto acelerar no solo la caracterización de un ADN, sino las ciencias que se apoyan en el mismo. Para acelerar estos programas se puede hacer uso de la programación paralela, la cual permite utilizar varios elementos de computación simultáneamente para obtener resultados. Además será necesario contar con mecanismos de gestión y tratamiento de grandes cantidades de datos para poder manejar los datos que producen los secuenciadores.

1.2. Estructura de la memoria

Este documento se puede dividir en tres grandes partes, estas a su vez, conformadas por capítulos. El capítulo 2 forma la parte de las asignaturas cursadas. En este capítulo se explican qué asignaturas se han cursado durante la realización del máster, su contenido, qué se ha realizado y qué relación tienen con el tema de investigación.

La segunda parte del documento está compuesta por los capítulos 3, 4 y 5, en los cuales se detalla el estado del arte que influye en el tema de investigación. El capítulo 3 habla sobre los entornos y sistemas paralelos que se utilizan en la actualidad y que podrían ser de utilidad a la hora de acelerar las herramientas que se quieren desarrollar. El siguiente capítulo se centra en el tratamiento de grandes cantidades de datos, cómo se manejan y procesan los grandes bancos de datos que no pueden ser cargados completamente en memoria. El capítulo 5 es el último de la parte de estado del arte, en el se comentan algunas de las herramientas actuales que existen para realizar análisis de bioinformática. Se verán las características de estas herramientas y cómo podrían ser mejoradas.

En la tercera parte del documento se encuentran los capítulos 6 y 7. El capítulo 6 muestra una descripción detallada del anteproyecto de tesis que se seguirá para la realización de la tesis doctoral. En este capítulo se describirá desde donde se ha partido, cuales son los objetivos a alcanzar durante los próximos años y que planificación de tareas se seguirá. El capítulo 7 describe el trabajo realizado hasta el momento, el cual incluye las primeras tareas de la planificación del anteproyecto y sus resultados. Se describirá también la herramienta que se ha desarrollado y el algoritmo que implementa, haciendo una pequeña evaluación de lo que se ha conseguido.

Al final del documento se incluye la bibliografía utilizada para su elaboración y un breve currículum vítae del autor.

Capítulo 2

Asignaturas Cursadas

En este capítulo se resumen las asignaturas, sus objetivos y contenidos, elegidas por el alumno para su realización durante el máster.
En general, las asignaturas tienen como objetivo general la formación del personal investigador en un amplio espectro, desde la formación en términos técnicos de informática a aspectos que lo relacionan con el ámbito investigador.
Se presentará el trabajo realizado en cada una de la asignaturas cursadas, su metodología. y cuál es su relación con el tema de investigación de tesis.

2.1. Generación de documentos científicos en informática

2.1.1. Descripción

Esta asignatura engloba varios aspectos del mundo de la investigación, como son la elaboración de publicaciones, de documentos científicos y de análisis de propiedades de los entornos a investigar.

La primera parte del curso trata de introducir al alumno en el mundo de la investigación, presentándole a qué se va a enfrentar y cómo se evalúan los trabajos científicos mediante publicaciones y en qué lugares, tanto a nivel de investigador como a nivel de organización.

La segunda parte se centra en la elaboración de documentos científicos, cómo deben organizarse y escribirse para que sean aceptados en el ámbito investigador. Además, se presentan algunas herramientas, entre ellas LaTeX,

que permiten escribir documentos de forma limpia y adecuada.

En la última parte del curso se enseña al alumno a organizar sus experimentos respetando ciertas condiciones. Esto permite realizar posteriormente contrastes sobre los resultados de los mismos para obtener finalmente conclusiones fiables y sólidas.

2.1.2. Trabajo realizado

En la primera parte de la asignatura se realizan trabajos de búsqueda de información, a modo de entrenamiento, sobre plataformas dedicadas a este tipo de búsquedas, entre las cuales se encuentran Google Scholar, Scopus y Web of knowledge. Uno de los objetivos de este trabajo es realizar un análisis de la calidad de los profesores del Departamento de Sistemas Informáticos de la ESII en Albacete, utilizando como métrica los índices H.

Para la segunda parte de la asignatura se realizan diversos ejercicios de entrenamiento en el uso de LaTeX. El objetivo de estos ejercicios es la preparación para poder realizar finalmente la parte escrita y la presentación del trabajo final de la asignatura mediante esta herramienta.

En la última parte de la asignatura se enseña al alumno a realizar contrastes de hipótesis, sobre datos obtenidos de experimentos mediante la realización de ejercicios de entrenamiento. Esto permite obtener propiedades importantes acerca de esos datos.

Para el trabajo final de asignatura he realizado un contraste de hipótesis sobre el comportamiento del programa de análisis de genoma *GATK* en entornos de programación paralela. Básicamente el contraste consiste en determinar si hay diferencia, en cuanto a rendimiento, al usar el programa con distinto número de hilos de ejecución, concretamente entre 7 y 8 hilos. Los resultados del contraste indicaban que el rendimiento con 7 hilos era el mismo que con 8, por lo que podrías ahorrar costes utilizando sólo 7 *cores* del procesador en lugar de 8.

2.1.3. Relación con el tema de investigación

Esta asignatura está relacionada con todos los temas de investigación, puesto que para cualquiera se necesita obtener información, elaborar

documentos sobre el tema, realizar experimentos y contrastarlos, etc.

Por tanto, para esta investigación será útil a la hora de realizar las tareas descritas en la asignatura, incluso en la propia elaboración de este documento he utilizado los conceptos aprendidos en esta asignatura. En cuanto al análisis de resultados también me será útil puesto que tendré que determinar y contrastar los rendimientos y tiempos obtenidos durante mi investigación en la búsqueda de los algoritmos y métodos mas óptimos.

2.2. Introducción a la programación de arquitecturas de altas prestaciones

2.2.1. Descripción

Esta asignatura engloba el ámbito de la programación secuencial, de forma eficiente, para aprovechar al máximo entornos de altas prestaciones. También aborda la programación paralela en este tipo de entornos, ofreciendo una metodología de diseño y evaluación de algoritmos paralelos.

Las técnicas de programación de arquitecturas de altas prestaciones tienen como objetivo establecer una metodología que permita obtener códigos capaces de resolver problemas de la forma más rápida y eficiente posible. Para ello se consideran dos tipos de técnicas distintas: optimizar el código secuencial y paralelizar el código secuencial.

Además de las técnicas de programación, se presentan conceptos importantes que mejoran el rendimiento de la máquina si son tenidos en cuenta, como puede ser la localidad temporal y la localidad espacial. Es en estos conceptos en los que se apoyan además las técnicas vistas en la asignatura.

2.2.2. Trabajo realizado

En primer lugar, en la asignatura se ha estudiado como se puede optimizar un código secuencial para que aproveche eficientemente los recursos de cómputo de los que se dispone. Para ello se han presentado dos conceptos importantes a explotar: la localidad temporal y la localidad espacial. Estos conceptos nos dicen que si un programa utiliza un dato, es muy probable que se vuelva a utilizar en un futuro cercano (temporal), también nos dice

que hay una alta probabilidad de utilizar seguidamente el dato contiguo (espacial).

También se ha estudiado la importancia de detectar y minimizar el efecto de los cuellos de botella, que suelen ser las operaciones I/O y el intercambio de datos con la memoria central. Lo que se pretende es intentar realizar estas operaciones sin que el procesador quede desocupado esperando que se completen.

La técnica que se ha estudiado y que intenta minimizar los problemas anteriores y obtener más eficiencia es la denominada programación orientada a bloques. Es una técnica que funciona bien para operaciones de álgebra lineal, como puede ser la multiplicación de matrices. Consiste en dividir los datos de entrada en bloques y procesarlos a ese nivel.

En las prácticas de la asignatura se realizan programas de multiplicación de matrices utilizando programación en bloques, observándose cómo efectivamente se conseguían unos resultados mucho más rápidos y de manera más eficiente. Además, se programa también utilizando la librería de cálculo *BLAS* para comparar y ver hasta qué punto se puede optimizar un algoritmo de multiplicación de matrices (hasta 12 veces más rápido en las pruebas).

El siguiente paso en la asignatura fue la utilización de la programación paralela utilizando la librería de paso de mensajes *MPI*, y utilizando también una implementación paralela de la librería *BLAS* llamada *PBLAS*. Para ello se considera de nuevo el problema de la multiplicación de matrices y se realizan soluciones paralelas. Aunque solo se programa mediante el paradigma de programación en entornos de memoria distribuida, se estudia también el paradigma de memoria compartida.

Una vez obtenidas las soluciones paralelas, se obtienen medidas de rendimiento como son el *SpeedUp* y la eficiencia, las cuales nos indican cuánto más rápidas son estas soluciones respecto a la secuencial y cómo de eficientes son en cuanto al uso de los recursos disponibles. Esto nos permitía comparar las soluciones entre sí y evaluar cuales son las más adecuadas para resolver el problema inicial.

2.2.3. Relación con el tema de investigación

La programación paralela puede ser muy importante a la hora de acelerar un programa que requiere de un alto rendimiento, como son los programas de análisis de genoma. Además, si se cumplen ciertas condiciones favorables al paralelismo, la velocidad podría aumentar cuantos más elementos de procesamiento utilizáramos.

El concepto de computación por bloques es totalmente aplicable a nuestro problema, los ficheros que hay que procesar son tan grandes que necesariamente hay que procesarlos por bloques ya que, de otro modo, colapsaríamos la memoria de la máquina. Esto nos permite también procesarlos de manera paralela, puesto que cada bloque es independiente de los demás y por tanto podemos procesarlo en procesadores distintos.

Es necesaria la utilización de un mecanismo de paso de mensajes (como puede ser *MPI*) para realizar un procesamiento paralelo en varías máquinas, por lo que la aplicación, con lo visto en esta asignatura, al programa de genómica es prácticamente directa. Esto permitirá partir el genoma en trozos y procesarlos de forma distribuida y paralela.

2.3. Tecnologías de red de altas prestaciones

2.3.1. Descripción

Esta asignatura pretende presentar el papel que las redes de interconexión tienen hoy día en la arquitectura de diversos sistemas distribuidos, desde los supercomputadores (miles de nodos de cálculo unidos por una red de altas prestaciones), hasta los entornos Grid y Cloud, donde la interconexión es la propia Internet.

El objetivo de esta asignatura es la descripción de los aspectos más relevantes de una red de altas prestaciones. También se analizan las alternativas de diseño para los distintos elementos de las redes de interconexión, además de comprender y distinguir los distintos tipos de arquitecturas de computación distribuida de la actualidad.

2.3.2. Trabajo realizado

Durante la primera parte de esta asignatura se realizan diversos trabajos para obtener información sobre las redes que se utilizan en los mejores supercomputadores del mundo, obtenidos de las listas del Top 500. El objetivo es caracterizar estas redes y obtener sus principales propiedades, para determinar así su adecuación al entorno en que se estaban utilizando.

Otra parte de la asignatura consiste en la lectura, análisis, resumen y presentación de diversos artículos científicos relacionados con los contenidos que se presentan en la misma. Esto permite adquirir práctica en la lectura de artículos, además de obtener algunos conocimientos extra sobre redes de interconexión.

Para el trabajo final de asignatura he realizado una recopilación de información sobre software de análisis de genoma que utilizaba sistemas distribuidos cloud para acelerar estos análisis. En el trabajo se describen detalles sobre cómo utilizan los programas el cloud, y qué elementos de éste les daba la ventaja.

2.3.3. Relación con el tema de investigación

Para la computación distribuida hay que utilizar de un modo u otro una red de interconexión que una todos los nodos de procesamiento, sea o no de altas prestaciones, por lo que los conceptos introducidos por esta asignatura para estas redes son directamente aplicables a mi investigación. La red de interconexión es de los elementos más importantes a la hora de obtener beneficio en la computación distribuida.

Además, dependiendo del entorno de aplicación, unas redes serán más adecuadas que otras, ya que no es lo mismo acelerar un programa para un *cluster* con una red de interconexión determinada, que acelerarlo para una red *on-chip*.

2.4. Modelado y evaluación de sistemas

2.4.1. Descripción

Esta asignatura se centra en presentar al alumno una visión sobre el modelado de sistemas que le permita además evaluar sus características, centrándose en el modelado para simulación. El objetivo es el análisis de un sistema real, tanto estático como dinámico, para obtener un modelo que lo describa lo mejor posible.

Además ese modelo es utilizable para simulación, un proceso que lo utiliza para analizar y evaluar el rendimiento de un sistema antes de construirlo, evaluar las consecuencias de un suceso antes de que ocurra, comparar varios sistemas entre sí, conocer lo que ocurre realmente en el sistema...

2.4.2. Trabajo realizado

En la primera parte de la asignatura se presentan algunos métodos para representar un sistema real en un modelo que sea analizable computacionalmente para determinar su comportamiento. Entre los modelos que se presentan se encuentran principalmente los que se utilizan para el campo de la simulación, que permite determinar la evolución de un sistema en el tiempo. Para ello hay que caracterizar el sistema que se quiere modelar, siendo de varios tipos: estático, dinámico, continuo, discreto...

En la segunda parte de la asignatura se presenta teoría de colas como herramienta para evaluar un sistema del tipo servidor que utilice colas donde depositar los elementos a los que va a dar servicio. Mediante las métricas que ofrece se pueden evaluar tiempos de respuesta, dispersión de peticiones que pueden ser atendidas, ritmo máximo de llegada de esas peticiones para que puedan ser atendidas, etc.

En la ultima parte se inicia al alumno en el uso de dos simuladores de redes de interconexión: NS2[9] y OPNET[10]. Estos dos son los simuladores más extendidos para este tipo de simulación, permitiéndote modelar una red tanto a nivel de topología como a nivel hardware de conmutador.

También se realiza un trabajo final que tenga que ver con los contenidos presentados en la asignatura, en mi caso el tema es el modelado del ADN

para que sea analizable computacionalmente y una pequeña evaluación de rendimiento del recalibrador que he diseñado.

2.4.3. Relación con el tema de investigación

La relación de esta asignatura con el análisis del ADN es directo, puesto que para analizarlo primero hay que modelarlo en la máquina y a partir de ahí se podrán comparar ADNs entre sí, simular las consecuencias de determinados cambios en la estructura del mismo, etc.

Por otra parte también es útil para evaluar el rendimiento del sistema, cuya rapidez de respuesta debe ser lo más alta posible, evaluando cuellos de botella y código que tiene que ser acelerado.

2.5. Modelos para el análisis y diseño de sistemas concurrentes

2.5.1. Descripción

En esta asignatura se describen los principales modelos para la descripción de sistemas concurrentes, como son las álgebras de procesos, redes de Petri y autómatas de estados finitos. También se abordan extensiones de estos modelos que incrementan su capacidad como son los modelos temporizados y con probabilidades.

Adicionalmente se presentan las principales herramientas que dan soporte a dichos modelos como por ejemplo *Uppaal*[11] y cuales son las técnicas de análisis de propiedades que utilizan.

2.5.2. Trabajo realizado

En esta asignatura no han habido partes temporalmente diferenciadas, todos los contenidos se han dado simultáneamente lo cual permite comparar los distintos modelos entre sí y analizar sus ventajas e inconvenientes respecto a otros.

Entre los modelos que se presentan están las redes de Petri y las álgebras de procesos, estas últimas pueden dar lugar a los modelos de autómatas de estados finitos. Durante el curso el alumno realiza el modelado de varios casos de estudio, usando esos 3 modelos distintos y analizando las propiedades de los mismos.

Finalmente se realiza un ejercicio de modelado final utilizando las tres herramientas que se presentan en la asignatura (*Tina*, *Uppaal* y *CWB*), que utilizan uno de los tres modelos presentados en el curso.

2.5.3. Relación con el tema de investigación

Lo visto en esta asignatura me sirve para modelar un sistema y determinar sus propiedades más importantes. Estas propiedades pueden ser la aparición de bloqueos por ejemplo, o que un determinado elemento del programa llegue a ejecutarse alguna vez. Además, mediante la validación del modelo podemos determinar si realmente un programa se comporta como queremos.

Puedo por tanto modelar el comportamiento de cualquier software que diseñe durante la tesis. Esto me permitiría evaluar su comportamiento teniendo en cuenta la mayoría de casos posibles a los que puede llegar y que además sea fiable.

2.6. Computación en clusters

2.6.1. Descripción

En esta asignatura se estudian y analizan los diferentes aspectos de un *cluster* de computadores. Se describen las tendencias en cuanto a los nuevos sistemas de interconexión y I/O, como puede ser Infiniband. Además se presentan las posibilidades y los problemas a resolver en cuanto a la configuración de plataformas *cluster*, mostrando ejemplos de aplicaciones que permiten aprovechar estas arquitecturas. Como último punto se introduce al alumno en la programación paralela usando *MPI*.

2.6.2. Trabajo realizado

En la primera parte de la asignatura se presenta al alumno un estado del arte sobre qué son los *cluster* y su impacto en el mercado, analizando la lista del *Top 500* de computadores. Seguidamente se tratan los entornos software necesarios para que una arquitectura de tipo *cluster* de la sensación al usuario de estar trabajando con un único recurso computacional, lo cual se denomina imagen de sistema único.

En la parte final se presenta al alumno la librería *MPI* y una pequeña introducción a su programación orientada a *cluster*. Para ello se le anima a programar un algoritmo basado en el recorrido de matrices usando esta librería y ejecutándolo en un *cluster* con distinto número de nodos. Finalmente se hace uso de un software de monitorización llamado "*Paraver*" analizando trazas de una aplicación *MPI* y permitiendo determinar cual ha sido el comportamiento durante su ejecución.

Como trabajo final de asignatura, he realizado una introducción sobre la tecnología *SSE*[12] (*Streaming SIMD Extensions*) como uso del paralelismo a nivel de procesador en un *cluster*, explicando en qué consiste y pequeños ejemplos sobre su uso.

2.6.3. Relación con el tema de investigación

El uso de un *cluster* es crucial para acelerar las aplicaciones de análisis de genoma, los grandes requisitos computacionales de estas requieren muchos recursos a su disposición, siendo un *cluster* el que puede ofrecerlos. Por lo tanto se puede orientar el desarrollo del software al uso de estas plataformas que ofrecen un alto grado de paralelismo y además distintos recursos, adecuados para distintas situaciones (procesadores, GPUs...).

Además se puede utilizar *MPI* para estos fines, ya que permite aprovechar los recursos disponibles y además está ampliamente documentado y soportado por la comunidad científica.

Capítulo 3

Estado del Arte: Entornos paralelos

La computación paralela [13] ha tenido un tremendo impacto en una gran cantidad de áreas de investigación, desde simulaciones para la ciencia, hasta su aplicación en aplicaciones comerciales para tareas de minería de datos y procesamiento de transacciones.

Se está produciendo una revolución en entornos *HPC* (*High Performance Computing*) para aplicaciones científicas, concretamente el International Human Genome Sequencing Consortium ha abierto nuevas fronteras en el campo de la bioinformática. El objetivo es el de caracterizar los genes, funcional y estructuralmente, para entender la influencia de los procesos biológicos, analizar las secuencias de genoma con vistas al desarrollo de nuevos medicamentos y desarrollar curas para las enfermedades. Todos estos objetivos están siendo posibles gracias a la computación paralela. La computación paralela no sólo se limita a este campo, también tenemos avances en física, química, astrofísica...

La bioinformática presenta uno de los problemas más difíciles de abordar, el tratamiento de grandes bancos de datos o *datasets*. Estos bancos de datos pueden llegar a ser de los más grandes en el ámbito científico, por lo que analizarlos puede requerir una gran capacidad de computo que actualmente solo puede ser abordado por métodos paralelos.

En esta sección se discutirán los principales modelos de programación paralela que se utilizan actualmente, acompañados de algunas de sus implementaciones prácticas. Para ello se introducirá la clasificación de las diferentes plataformas paralelas según la Taxonomía de Flynn, puesto que

nos permite clasificar tanto los modelos de programación paralela como los sistemas de cómputo.

3.1. Taxonomía de Flynn

La taxonomía de Flynn [14, 15] establece una clasificación general de los sistemas paralelos basándose en las interacciones entre los flujos de datos y los flujos de instrucciones. Es importante caracterizar los sistemas paralelos que se quieren abordar, puesto que dependiendo de sus características unas soluciones serán más viables que otras.

En primer lugar tenemos los sistemas *Single Instruction Single Data* (*SISD*), los cuales tienen un único flujo de instrucciones sobre un único flujo de datos. Actualmente los monoprocesadores (con un solo flujo de instrucción, hilo o *thread*) son el ejemplo más claro de este grupo. En general este grupo engloba cualquier máquina secuencial.

Después tenemos los sistemas *Single Instruction Multiple Data* (*SIMD*), los cuales tienen un único flujo de instrucciones sobre múltiples flujos de datos. En la actualidad este tipo de sistemas están en auge, ya que hace unos años no se concebían para computación paralela en ámbitos científicos o de altas prestaciones. El ejemplo más directo son las unidades de procesamiento de gráficos (*GPU*), las cuales están siendo ampliamente utilizadas en computación científica. Debido a esto, las GPU se están combinando con CPU de propósito general para dar lugar a las denominadas *GPGPU* [16] (*General Purpose GPU*). Las GPGPU tratan de aprovechar la capacidad de cómputo de las GPU en ámbitos de propósito general, ya que una GPU es un sistema SIMD y por tanto sus aplicaciones muy concretas.

Los sistemas *Multiple Instruction Single Data* (*MISD*) tienen varios flujos de instrucciones sobre un único flujo de datos. Para encontrar actualmente un ejemplo sobre este tipo de sistema hay que irse, por ejemplo, al interior de un procesador (sistemas segmentados). La decodificación segmentada que realiza sobre las instrucciones que tiene que realizar es un sistema MISD, puesto que para cada instrucción que tiene que realizar, le aplica siempre el mismo proceso (búsqueda, decodificación, ejecución...).

Por último tenemos los sistemas *Multiple Instruction Multiple Data* (*MIMD*), que son los más extendidos actualmente. Estos sistemas aplican

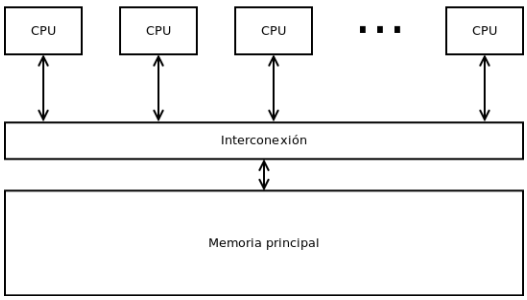

Figura 3.1: Modelo paralelo de memoria compartida.

múltiples flujos de instrucciones sobre múltiples flujos de datos. El ejemplo más claro lo tenemos en los multiprocesadores, donde podemos ejecutar varios procesos simultáneamente sobre varios conjuntos de datos. A mayor escala tendríamos los *cluster*, que consisten en grupos de procesadores interconectados para realizar varios procesos de forma simultánea.

Aunque esta clasificación se formuló hace unos 40 años, hoy día sigue siendo válida y nos permite caracterizar prácticamente todos los sistemas paralelos.

3.2. Modelos de programación paralela

En este apartado se presentarán los tres modelos principales de programación paralela que se utilizan, los cuales se distinguen en la forma que tienen los procesadores para comunicarse.

3.2.1. Espacio de memoria compartido

Este modelo de programación paralela se caracteriza por tener un espacio de memoria físicamente unido, común a todos los procesadores que intervienen en el sistema. La interacción se realiza mediante la modificación de los datos que residen en la memoria compartida. En la Figura 3.1 se muestra gráficamente el modelo de memoria compartida.

Hay dos tipos de memoria en las plataformas que implementan este sistema, la memoria local a un procesador y la memoria global a todos los procesadores. Si el tiempo de acceso a ambas memorias es idéntico, estamos

antes un sistema de acceso a memoria uniforme (*UMA*); si el tiempo de acceso es distinto, estamos ante un sistema de acceso a memoria no uniforme (*NUMA*) puesto que dependiendo de la posición de memoria que se acceda y su lejanía del nodo que la accede, el tiempo de acceso varía. Un ejemplo de sistema *UMA* es un sistema multiprocesador simétrico o *SMP*, en cambio, un sistema multiprocesador escalable de memoria compartida pertenece al tipo *NUMA*.

Este modelo hace que la programación sea mucho más fácil, puesto que las lecturas de memoria son transparentes al programador, siendo necesario únicamente una mayor complejidad en las operaciones de lectura/escritura. Esto es debido a que se hacen necesarios mecanismos de bloqueos sobre variables o *locks* para evitar el acceso simultáneo de varios procesadores a los mismos datos (exclusión mutua).

Puesto que las arquitecturas actuales incorporan el uso de memorias caché en los procesadores, se hace necesaria la implementación de sistemas de coherencia de caché en estos sistemas, lo cual aumenta la complejidad hardware de estas arquitecturas. Los sistemas *NUMA* se dividen en dos tipos dependiendo del sistema de coherencia que implementan, los *CC-NUMA* mantienen la coherencia en todas las caches, mientras que los *NCC-NUMA* no.

Algunos paradigmas de programación utilizados en el modelo de espacio de memoria compartido son los threads (como por ejemplo *Pthreads*), directivas (OpenMP) y uso de GPUs (por ejemplo CUDA).

Los POSIX [17] (*Portable Operating System Interface uniX*) *threads* (a partir de ahora *pthreads*) se basan en el concepto de hilo o *thread*, que la mayoría de sistemas operativos actuales utiliza.

En un sistema UNIX, un proceso no es más que un único hilo de control. Este hilo podría replicarse para crear más hilos "hijos", permitiendo que cada uno de estos hilos se dedique a ejecutar una tarea distinta a los demás. Esto permite no solo ejecutar un programa con varios hilos ejecutándose en paralelo, sino que permite abstraer el programa en tareas definidas, independientemente del número de procesadores que tengamos. Estos hilos también comparten memoria, por lo que se enmarca dentro del modelo de memoria compartida.

Pthreads [17] es la interfaz de programación con hilos que ofrecen las

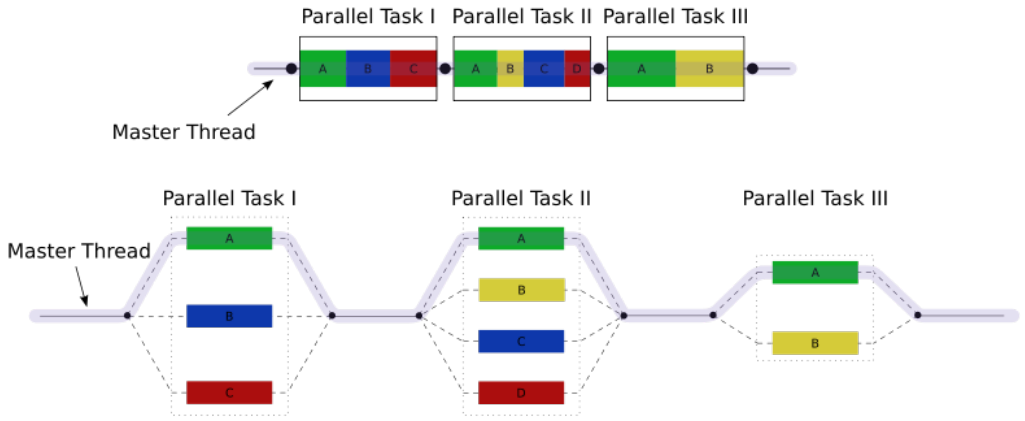

Figura 3.2: Modelo paralelo *Fork-Join* usado por *OpenMP*[1].

librerías POSIX. Esta interfaz nos ofrece funciones simples y potentes necesarias para crear cualquier código que utilice hilos.

Por otro lado, *OpenMP* [15, 18, 19] es un API de programación paralela para multiprocesadores de memoria compartida (basada en directivas del compilador, rutinas de librería y variables de entorno), en C/C++ y Fortran. Puede ejecutarse en sistemas Unix y Windows. Sus características más importantes residen en su simplicidad y flexibilidad, ofreciendo una interfaz que permite crear programas paralelos tanto para sistemas sobremesa como supercomputadores.

OpenMP basa su funcionamiento en el modelo *fork-join* (Figura 3.2) para llevar a cabo las tareas paralelas. Además se asegura una ejecución correcta tanto en sistemas paralelos como secuenciales (siempre y cuando sea correctamente utilizado). Por otra parte, utiliza el modelo de memoria compartida para ejecutar sus hilos. Permite que los hilos compartan espacio de memoria, pero también permite que tengan su propio espacio de memoria privada. Además permite a los hilos obtener una instantánea del estado de la memoria o *snapshot* en cualquier momento, y utilizar esa información mientras el resto de hilos pueden seguir modificando la memoria sin afectar al *snapshot*.

Otro API de programación utilizado para plataformas de memoria compartida basadas en GPU es *CUDA*[20]. Las GPUs son máquinas SIMD según la taxonomía de Flynn (véase Sección 3.1) que utilizan el modelo de memoria compartida. Una diferencia con los procesadores con varios núcleos

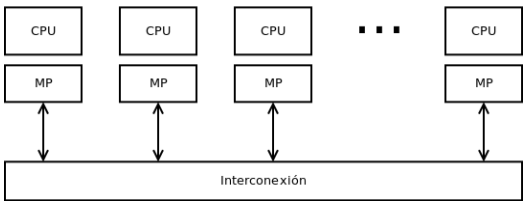

Figura 3.3: Modelo paralelo de paso de mensajes.

de procesamiento o *multicores*, que también utilizan este modelo, es que las GPUs pueden tener miles de hilos ejecutandose simultáneamente. Un programa en CUDA tiene dos partes: el código que se ejecuta en el *host* o procesador central (CPU), y el código, llamado *kernel*, que se ejecuta en el *device* (GPU). Debido a esto es necesaria una CPU que trabaje en conjunto con la GPU. Cualquier aplicación que pueda explotar el modelo SIMD, donde se apliquen las mismas instrucciones a un conjunto de datos, se verá muy beneficiada en el uso de una GPU. Cualquier otro tipo de aplicación no es factible en estas plataformas.

Otro ejemplo de sistema de memoria compartida son las instrucciones *SSE* [12] (*Streaming SIMD Extensions*), consisten en un conjunto de instrucciones máquina del procesador junto con determinado hardware asociado. Se basa en un modelo *SIMD* debido a que es un procesamiento vectorial donde se aplica una misma instrucción a varios datos de forma simultánea en el tiempo. Por tanto, con estas instrucciones podremos realizar en paralelo la misma operación sobre varios datos al mismo tiempo.

3.2.2. Paso de mensajes

En este modelo, la plataforma consiste en una serie de nodos de procesamiento interconectados, cada uno con su propio espacio de direcciones de memoria. Cada nodo puede ser un solo procesador o un multiprocesador con espacio de memoria compartido entre sus núcleos de procesamiento o *cores*. En este modelo, la interacción entre procesos ejecutándose en distintos nodos se realiza mediante el paso de mensajes. En la Figura 3.3 se puede ver una representación de un sistema que utilizaría paso de mensajes para la comunicación.

Los mensajes proporcionan un modo de transferir datos, tareas y sincronizar procesos. Puesto que la interacción se consigue mediante

envío/recepción de mensajes, las operaciones básicas de este paradigma de programación son *send* y *receive*. Es necesario además un mecanismo que asigne un identificador único (*id*) a cada proceso que permita identificarlo en un programa paralelo con múltiples procesos, puesto que los mensajes necesitan un *id* de destino para ser entregados. Un proceso podría saber pues su *id* introduciendo la operación *whoami*.

Otra función necesaria es *numprocs* que diga al proceso el número de procesos que intervienen en el programa paralelo. Con las cuatro operaciones introducidas anteriormente se puede implementar cualquier programa de paso de mensajes. Algunas de las implementaciones de paso de mensaje más utilizadas son *Message Passing Interface* [20, 15, 21] (*MPI*) y *Parallel Virtual Machine* [22] (*PVM*).

Message Passing Interface es un estándar de facto para el modelo de programación paralela mediante paso de mensajes. MPI, al igual que OpenMP, es una API usada generalmente en entornos C/C++ entre otros. Se dispone de implementaciones tanto en código libre, como propietarias. Entre las implementaciones más conocidas se encuentran *OpenMPI* [23], *LAM* [24] (*Local Area Multicomputer*) y *MPICH* [21] (*MPI over CHameleon*).

La forma de procesar un programa, en un entorno MPI, consiste en ejecutar copias de ese programa (procesos) en tantas máquinas como se quieran utilizar. Cada copia se encarga de procesar una parte de los datos de entrada intercambiando información, mediante paso de mensajes, por la red. MPI permite distintas clases de comunicación en los sistemas que la utilizan. En las comunicaciones colectivas interviene un nodo que interactúa con los demás, ya sea que todos le manden información al mismo o que éste envíe información a todos. También permite las comunicaciones punto a punto. Además, estas dos clases de comunicación tienen dos variantes: bloqueantes y no bloqueantes.

Adicionalmente, existe una API basada en MPI para operaciones I/O en sistemas paralelos de almacenamiento, se llama *MPI-IO* [21]. Esta API es la parte de MPI que maneja el I/O ofreciendo distintas operaciones de lectura y escritura. Permite además determinar cómo se van a distribuir los datos entre los procesos, y cómo se van a almacenar los datos en disco una vez procesados. Esto permite solucionar las limitaciones de memoria al procesar ficheros muy grandes (no hay que cargarlos en la memoria de una sola máquina) y obtener un único fichero de salida.

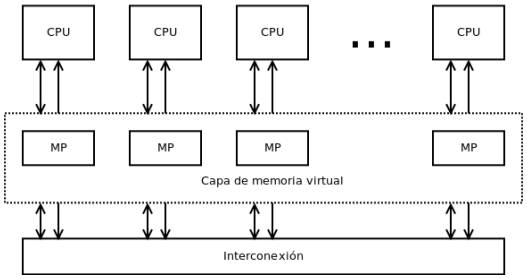

Figura 3.4: Modelo paralelo PGAS.

3.2.3. Partitioned Global Address Space - PGAS

El modelo de PGAS [25] se caracteriza por ofrecer al programador un espacio de memoria único y compartido en un sistema con memoria distribuida. Mediante una capa interfaz, se ofrece a los nodos (que tiene cada uno su memoria local) una forma de utilizar también las memorias locales de los otros nodos del sistema. Lo que esto ofrece al programador es una visión local de todo el espacio de memoria como uno único y global, transparencia en las comunicaciones (puesto que el compilador se encarga de las mismas) y soporte para datos distribuidos. En la Figura 3.4 se muestra una posible solución de un modelo PGAS.

A nivel de nodo, la variables de memoria pueden ser *shared* o *local*, permitiendo compartir únicamente las variables que interesan para la comunicación y no sobrecargando la comunicación con variables auxiliares y privadas que solo le sirven al nodo local. Algunos APIs de programación que utilizan este modelo son *Unified Parallel C* [26], *Co-Array Fortran* [27], *Titanium* [28], *X10* [29], *Chapel* [30] y *OmpSS* [31].

OmpSS es un modelo de programación que utiliza como base OpenMP (y puede considerarse una variante de PGAS), modificándolo para soportar paralelismo irregular, asíncrono y en plataformas heterogéneas. Utiliza un modelo de *pool* de hilos en lugar de *fork-join*(usado por OpenMP), donde un hilo maestro crea los trabajos y el resto de hilos los procesan. Otra diferencia es la distinción de varios espacios de memoria de cada nodo, al contrario del único espacio de memoria de OpenMP. Es el propio compilador el que se encarga de distribuir los datos y elegir en qué dispositivo procesarlos. Ofrece una visión de memoria compartida en un sistema distribuido al igual que los PGAS, con la diferencia de que cada nodo tiene adicionalmente su propio espacio de memoria.

3.3. Conclusiones

Es interesante la aplicación del modelo de memoria compartida junto al modelo de paso de mensajes en la elaboración de esta *tesis*, a distintos niveles. Podemos utilizar el modelo de memoria compartida a nivel de procesador, puesto que hoy día los nuevos procesadores son mayoritariamente un sistema *SMP* (*Sistema Multi-Procesador*) *on-chip* con varios núcleos de procesamiento y que comparten la misma memoria principal. Podemos usar este modelo de programación para acelerar los tiempos de ejecución dentro del procesador, usando eficientemente todos los núcleos de procesamiento del mismo.

Podemos aplicar igualmente el modelo de paso de mensajes para llevar la ejecución a un entorno multiprocesador o un *cluster*. Mediante este modelo podemos acelerar el tiempo de ejecución de los programas usando en paralelo los procesadores de los que se dispone en este tipo de sistemas. Con ello se busca mayor velocidad de cómputo cuantos más recursos (procesadores) disponga el sistema.

En cambio, el uso del modelo de *PGAS* no se considera factible en esta Tesis. Esto es debido a que la capa de virtualización, para dar la visión de memoria compartida, es costosa en prestaciones. Debido a la sobrecarga a la que somete al sistema y puesto que estamos en entornos de altas prestaciones, no se utilizará al no considerarse necesaria.

Otra técnica a tener en cuenta sería solapar las operaciones I/O con el procesamiento, así mientras un flujo de ejecución procesa un lote, otro flujo puede ir leyendo el siguiente lote a procesar. Esto permite eliminar o reducir el coste adicional de acceder a disco.

Podemos concluir por tanto en el uso del modelo de memoria compartida combinado con el modelo de paso de mensajes, cada uno acelerando el tiempo de ejecución a distintos niveles en un sistema multiprocesador. Estos dos modelos se dan simultáneamente en un *cluster* con procesadores modernos, por lo que es totalmente factible. Además, sería factible incluir el uso de GPUs, las cuales se pueden encontrar en cualquier *cluster*, combinándolas con los procesadores de propósito general y por tanto disponer de su alta capacidad de cómputo para determinadas operaciones.

Capítulo 4

Estado del Arte: Tratamiento de grandes cantidades de datos

Una de las características más significativas en el tratamiento de genoma, es la gran cantidad de bancos de datos o *datasets* que se generan. Estos datos hay que manejarlos de forma eficiente y adecuada, para preservar su integridad y permitir a la vez que su acceso sea lo más rápido posible. Esta sección está dedicada a distintos modelos que se utilizan hoy día para el tratamiento de una gran cantidad de datos.

Conocidos habitualmente como *"Big data"*, el tratamiento de una gran cantidad de datos busca abarcar varias dimensiones (según *IBM*[32]):

- **Volumen**: Big data solo conoce un tamaño, grande. Actualmente nos encontramos inundados de datos, hablando en términos de terabytes (10^{12} bytes) e incluso petabytes (10^{15} bytes) de información.

- **Velocidad**: Este aspecto tiene suma importancia en el acceso y manejo de los datos y es crítico en sistemas de tiempo real.

- **Variedad**: No solo se manejan muchos datos, sino que además estos pueden tener distinta estructura: texto, audio, vídeo, etc.

Todas estas dimensiones son las que se pretenden abordar con las soluciones presentadas en esta sección. Unas se centran en el procesamiento de los datos como *MapReduce* y *Hadoop*, mientras que *S3* y *HDFS* se centran en el almacenamiento de los mismos.

4.1. MapReduce

MapReduce [33, 34] es un modelo de programación con una implementación asociada para procesar y generar grandes *datasets*. En los últimos años se han venido implementando cientos de códigos de propósito especial (Google implementó la mayoría) para procesar grandes cantidades de datos en bruto, como documentos o peticiones web, para obtener distintos tipos de información derivada (índices, grafos de webs, resúmenes...). La mayoría de estos procesos son triviales y no requieren de un cómputo intensivo, pero los datos de entrada son demasiado grandes y tienen que ser distribuidos entre miles de máquinas para procesarlos en un tiempo razonable.

Todo esto generaba cada vez más complejidad, por lo que la reacción fue el diseño de un nuevo tipo de abstracción que permitiese realizar simples cómputos ocultando detalles del paralelismo, la distribución de los datos y la carga, ofreciendo además tolerancia a fallos. Esto dio lugar al desarrolló de una librería llamada *MapReduce* (de mano de Google).

El cómputo en este modelo se basa en un conjunto de pares clave/valor de entrada, que generan un conjunto de pares clave/valor de salida. Esto se genera mediante dos funciones que el programador debe definir: *Map* y *Reduce*. *Map* obtiene un par de entrada y genera un conjunto de pares intermedios. Estos pares intermedios son recopilados internamente por la librería y los agrupa por su clave intermedia, pasándolos seguidamente a la función *Reduce*. La función *Reduce* acepta una de las claves intermedias y el conjunto de valores para esa clave, combinándolos para generar el conjunto de valores de salida. La Figura 4.1 muestra gráficamente un ejemplo para contar palabras usando el modelo *MapReduce*.

Actualmente hay mucho software que implementa este modelo como puede ser el conocido *Hadoop* [33, 2], *Hive* [35], *Cascalog* [36], *mrjob* [37]. A continuación se comentará *Hadoop*.

4.2. Hadoop

Hadoop es un proyecto *open-source* diseñado para el procesamiento de grandes cantidades datos. Es software fiable y escalable que se basa en computación distribuida. *Hadoop* se encarga de ejecutar las aplicaciones en un entorno distribuido como puede ser un *cluster*.

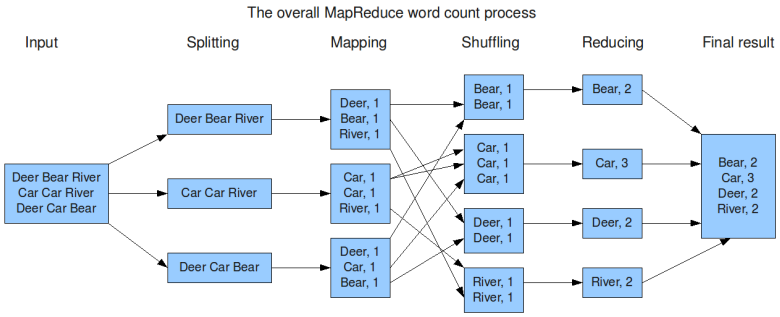

Figura 4.1: Modelo paralelo *MapReduce* para contar palabras [2].

Entre sus funciones se encuentra la de particionar la gran cantidad de datos de entrada y repartirlos entra las máquinas disponibles. Una vez ejecutada la aplicación, se recogen los resultados y se almacenan en el lugar correspondiente junto con información sobre el progreso de la ejecución.

Puesto que *Hadoop* utiliza el modelo *MapReduce* únicamente se puede utilizar para aplicaciones de paralelismo trivial, operaciones sencillas pero masivas que sean independientes las unas de las otras y por tanto no haya comunicación entre las mismas. Por este motivo *Hadoop* no funciona con aplicaciones que requieren de altas prestaciones.

4.3. Hadoop Distributed File System

Hadoop Distributed File System[33, 2] (*HDFS*) es un sistema de ficheros virtual diseñado para soportar aplicaciones que utilicen el modelo *MapReduce* que necesiten leer y escribir una gran cantidad de datos en lotes. El sistema de ficheros se construye sobre un sistema de almacenamiento distribuido.

Como ventajas en cuanto a almacenamiento, es la capacidad de renombrar y mover los ficheros, además de tener un árbol de directorios, pero no permite la modificación de los mismos para facilitar la coherencia. La escritura en los ficheros solo se permite por tanto para su creación. Este es el sistema de ficheros utilizado por *Hadoop*.

4.4. S3

S3 [33, 38] difiere de *MapReduce* en que no se centra en el procesamiento de datos, sino en su almacenamiento. Este servicio de *Amazon* permite almacenar grandes bloques de datos en un servicio *online*, con una interfaz que facilita el acceso a los mismos (mediante *HTTP*).

La visión más acertada es la de una base de datos de pares clave/valor, donde cada clave almacena un gran bloque de datos que corresponde al valor. Esto limita las operaciones, no permitiendo por ejemplo añadir más datos a un bloque, renombrarlos, reescribirlos o incluso tener un árbol de directorios.

La ventajas de este servicio residen en su bajo coste, su buena documentación, fiabilidad, rapidez y su fácil acceso desde cualquier entorno. Aún así normalmente se utiliza como una base de datos en bruto, muy simple.

4.5. Conclusiones

En esta Tesis se tendrá en cuenta, como base, el modelo *MapReduce*. Lo que se pretende es utilizar un modelo similar, parecido al *scatter-gather*, que nos permita leer por partes o lotes el *dataset* de entrada. Estos lotes serán enviados a procesar a distintos procesadores por orden de lectura y obteniendo los resultados de los mismos de forma parcial. Cuando un lote es procesado se lee el siguiente y se procesa. La cantidad de lotes leídos simultáneamente podría basarse en el factor de memoria disponible en todos los nodos, que depende del número de nodos del que se disponga.

Capítulo 5

Estado del Arte: Bioinformática

En los últimos años se han producido avances en las tecnologías de secuenciación que han aumentado la productividad a una escala sin precedentes, con lo que la bioinformática ha encontrado dificultades para almacenar y analizar esas grandes cantidades de datos. Se habla de que la biología ha entrado en la era del *"big data"*, enfrentándose a nuevos retos como son el almacenamiento, análisis, búsqueda, difusión y visualización de los datos que requieren nuevas soluciones.

En esta sección se verán algunas de las herramientas y lenguajes de programación más utilizados para abordar los problemas de análisis genómico y cuales son sus características más relevantes. Señalar que no son las únicas y que existen más, tanto de propósito especifico (la mayoría) como general.

5.1. Bioconductor

Bioconductor [39] es un software *open-source*, de libre desarrollo que proporciona herramientas para el análisis y comprensión de los datos de genómica. Está basado en el lenguaje de programación *R* [40]. Las herramientas que proporciona están contenidas en paquetes de *R* por lo que es orientado a objetos y le proporciona ciertas características: un lenguaje interpretado de alto nivel para un rápido prototipado de nuevos algoritmos, sistema de paquetes, acceso a datos de genómica *online*, soporte completo estadístico y capacidades de visualización de los modelos.

Los paquetes están bien documentados, incluso con ejemplos de uso. Ofrece además las herramientas junto con un flujo de trabajo a seguir para

realizar distintos análisis y mostrar sus resultados de forma gráfica.

5.2. Bioperl

Bioperl [41] es un conjunto de herramientas bioinformáticas basadas en módulos del lenguaje de programación *Perl*. Puesto que es modular, algunos módulos necesitarán de otros para llevar a cabo su función. Posee, entre otras cosas, interfaces gráficas, almacenamiento persistente en bases de datos y herramientas para procesar y traducir los resultados de varias de aplicaciones, para poder utilizarlos. Se utiliza para producir nuevas herramientas más complejas.

5.3. Genome Analysis ToolKit

Genome Analysis Toolkit [5, 3] (*GATK*) es el estandard *de facto* para el análisis y descubrimiento de variantes de genoma. Es un conjunto de herramientas genérico que puede ser aplicado a cualquier tipo de conjunto de datos y problemas de análisis de genoma, ya que se puede usar tanto para descubrimiento como para validación.

Soporta datos provenientes de una amplia variedad de tecnologías de secuenciación y aunque inicialmente se desarrolló para genética humana, actualmente puede manejar el genoma de cualquier organismo. *GATK* ofrece una estructura o *framework* que se ocupa de ejecutar las distintas herramientas sobre los datos de entrada. Estas herramientas pueden ser las que trae por defecto el propio programa u otras desarrolladas por los usuarios, permitiendo cualquier combinación compatible de las mismas. *GATK* sugiere varios flujos de trabajo a seguir para conseguir determinados resultados, pero se pueden realizar otros flujos personalizados.

Los requerimientos del programa son únicamente un sistema compatible con *POSIX* y Java [42] instalado. Ofrece también la posibilidad de mostrar los resultados gráficamente usando *R*. En cuanto al rendimiento, utiliza un sistema *MapReduce* para particionar el trabajo a realizar sobre la gran cantidad de datos de entrada y utiliza hilos para acelerar el proceso.

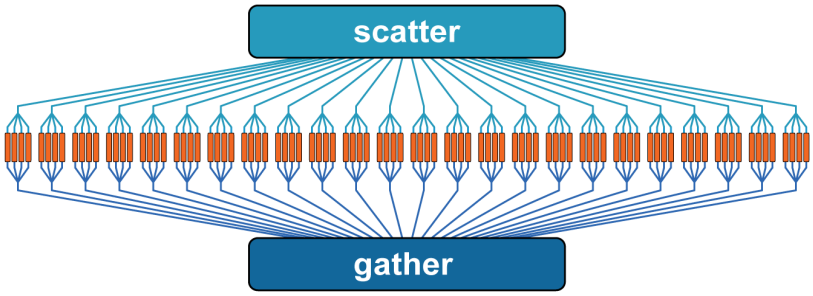

Figura 5.1: Modelo paralelo utilizado por *GATK* para el procesamiento en *cluster* [3].

Para el procesamiento a gran escala utiliza una estrategia que consiste en partir los datos de entrada y ejecutar cada parte en distintas máquinas independientemente, llamada *scatter-gather*. En la Figura 5.1 se representa el modelo *scatter-gather* que utiliza *GATK* donde cada tarea naranja sería una instancia distinta del programa.

En el aspecto del rendimiento, se echa en falta un procesamiento paralelo a nivel de *cluster* que permita a una tarea terminar antes y no tanto hacer varias a la vez. Además, no se centra en obtener un alto rendimiento, en parte por usar Java, prefiriendo la facilidad de programación y la portabilidad que ofrece este lenguaje de programación.

5.4. Bowtie

Bowtie [43] es un alineador de lecturas cortas de genoma (propósito específico) muy rápido y eficiente con la memoria. Este programa es muy usado pero tiene el inconveniente de que solo sirve para el alineamiento. Se puede utilizar como herramienta de altas prestaciones al inicio de un flujo de trabajo que utilice otras herramientas que la complementen. Este programa puede utilizar los *cores* del procesador usando hilos pero no tiene soporte interno al procesamiento en *cluster*.

5.5. OpenCB

OpenCB [44] es un proyecto que nace como alternativa a los proyectos que normalmente se focalizan en un lenguaje de programación como el de Bioconductor o Bioperl. Este proyecto proporciona software avanzado y *open-source* para el análisis de datos de genómica con una alta productividad.

El proyecto está organizado en cuatro subproyectos diferentes, cada uno centrado en resolver un problema concreto. Estos son el *"High-Performance Genomics"* (HPG), centrado en acelerar los análisis usando tecnologías *HPC* (High Performance Computing); *"Cloud computing"*, para almacenar y difundir los datos; *"Distributed NoSQL databases"*, para manejar la gran cantidad de datos; y *"Big data visualization"*, para obtener una representación visual de los resultados.

HPG es un proyecto escalable, de alto rendimiento y *open-source*, que consiste en diferentes herramientas y algoritmos para el análisis de datos a escala genómica. Utiliza HPC para acelerar los algoritmos y las herramientas para el análisis, usando el lenguaje de programación C y buscando los algoritmos que mejor se ajustan a cada problema en concreto. Se busca también la escalabilidad, que permita acelerar las ejecuciones cuantos más recursos computacionales estén disponibles.

5.6. Conclusiones

En esta Tesis tenemos como referencia la *suite* de herramientas *GATK* puesto que es la más utilizada. La razón de la elección radica en que aun siendo la más utilizada, no es una herramienta pensada para ofrecer un alto rendimiento. A pesar de esto, sus ventajas son la facilidad a la hora de procesar cualquier *dataset* de entrada, sea cual sea el secuenciador que lo haya generado. Esto unido a la capacidad para procesar el *dataset*, sea del tamaño que sea, y la posibilidad de aplicarle cualquier algoritmo justifican el amplio uso de esta herramienta.

En cuanto a sus desventajas, la principal es la falta de rendimiento como ya se ha comentado. La herramienta en sí hace su trabajo y obtiene resultados, pero no es factible para una aplicación cotidiana, por ejemplo, al ámbito de la sanidad, donde no es lo mismo que se obtengan los resultados de un paciente a los 2 meses que a las 2 horas.

En cuanto a la comunidad de genómica, *GATK* ha dejado de ser *open-source*, al menos parcialmente. Lo que se pretende por tanto es replicar esta herramienta para conservar sus ventajas pero intentando eliminar sus desventajas, ofreciéndoselo a la comunidad científica. Para ello se aplicarán técnicas de computación de altas prestaciones, usando lenguajes de programación eficientes con memoria y los recursos del sistema y además llevando la ejecución al ámbito de los sistemas multiprocesador.

Adicionalmente el trabajo de Tesis se realiza en colaboración con el Centro de Investigación Príncipe Felipe (CIPF), en el proyecto HPG (*High Performance Genomics*) desarrollando herramientas similares a GATK para la comunidad científica. Los principios del proyecto HPG coinciden con los nuestros, realizar los análisis de forma eficiente, rápida y compartirlo con la comunidad mediante *open-source*.

Capítulo 6

Anteproyecto de Tesis

Primeramente se describirá en qué consiste el proceso de descubrimiento de variantes, dando una breve introducción a la genética y al *ADN*. Se describirá desde dónde se parte y cuales son los objetivos a cumplir durante la realización de esta tesis. Finalmente se presentará un desglose de las tareas que se llevarán a cabo para alcanzar los objetivos propuestos y cuál será su planificación.

6.1. Introducción al descubrimiento de variantes

6.1.1. ADN

El *ADN* [7] (ácido desoxirribonucleico) contiene las instrucciones genéticas usadas en el desarrollo y funcionamiento de todos los organismos vivos conocidos y algunos virus, y es responsable de su transmisión hereditaria. Se puede comparar el ADN con una receta o código, ya que a partir del mismo se obtienen las instrucciones necesarias para construir las células. Los segmentos de ADN que contienen esta información son los denominados genes, mientras que el resto de secuencias tienen propósitos estructurales y reguladores en el uso de esta información. En la Figura 6.1 tenemos la representación gráfica que muestra dónde reside el ADN.

El ADN es un polímero de nucleótidos formado por muchas unidades simples (nucleótidos) conectadas entre sí (Figura 6.2). Los nucleótidos están formados por un azúcar, una base nitrogenada (pueden ser Adenina, Timina, Citosina y Guanina) y un grupo fosfato para interconectarlos. Puesto que

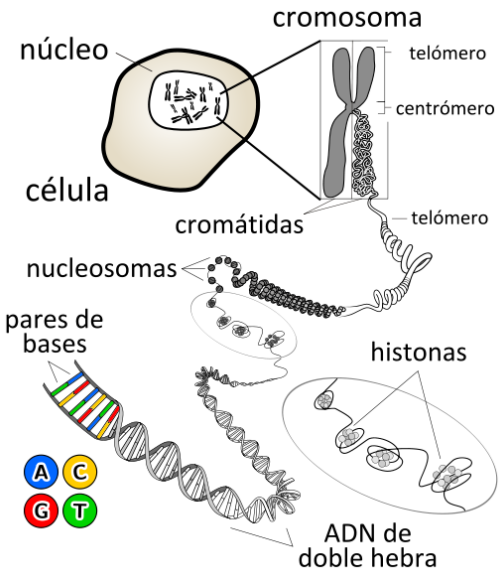

Figura 6.1: ¿Dónde se encuentra el ADN? [4].

lo único que distingue a un nucleótido de otro es la base nitrogenada, la secuencia de ADN se representa por una secuencia de estas bases. En estas secuencias cada base se representa por su inicial, por lo que tienen una representación del tipo $AGTCTAGATCG\ldots$ En los organismos vivos el ADN se presenta como una doble cadena de nucleótidos.

Como ya se ha dicho, un gen es una secuencia del ADN que contiene información genética, concretamente es una unidad de herencia que influye en una característica particular de un organismo (como el color de los ojos, por ejemplo). A partir de los genes se producen las proteínas del organismo, que son las encargadas de generar los músculos, pelo, enzimas...

6.1.2. Descripción del proceso de descubrimiento de variantes

Puesto que un cambio en el ADN puede traducirse en un cambio brusco de las proteínas que genera el organismo, esto puede dar lugar a trastornos, enfermedades, etc. El proceso de descubrimiento de variantes tiene como objetivo localizar variaciones, mutaciones o *indels* (*insertions/deletions*) en una muestra de ADN con respecto a otra, por ejemplo para poder determinar la causa de enfermedades o prevenirlas.

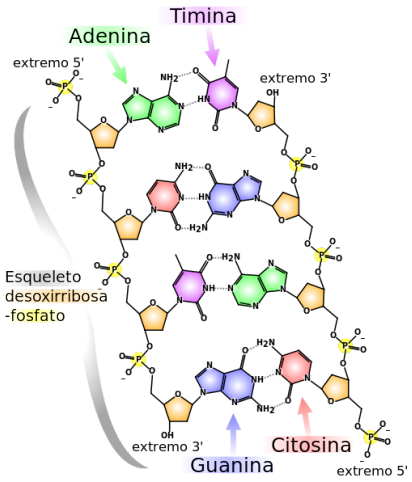

Figura 6.2: Representación química del ADN [4].

Un ejemplo sería analizar el ADN de una población (conjunto de individuos) que tengan un determinado tipo de cáncer y otra población que no lo tengan. Estos individuos tendrán muchas similitudes en el ADN puesto que pertenecen a la misma especie, pero también diferirán en otras. Cuando encontremos la secuencia de ADN común a la población con cáncer, pero que no aparezca en la población sin cáncer, tendremos la razón de ese cáncer localizada y podremos tratarlo y prevenirlo.

El proceso de descubrimiento de variantes tiene tres fases. En la primera fase se obtienen lecturas de ADN en un formato específico y dependiente de la plataforma que las haya obtenido. Lo que se hace con estas lecturas es transformarlas a un único formato genérico, con unas calidades bien calibradas, mapeadas y alineadas con su ADN de referencia. El formato utilizado es el *SAM/BAM* [45] (*Sequence Aligment Map/Binary Alignment Map*), el cual es independiente de la tecnología de obtención del ADN.

En la segunda fase se analizan los ficheros *SAM/BAM* para obtener posiciones del ADN que, según evidencia estadística, sean mutaciones respecto al ADN de referencia. Esto incluye diferencias en una sola posición de la cadena de ADN (*SNP*, *Single Nucleotide Polymorphism*), pequeños *indels*, etc. Esta fase genera ficheros *VCF* (*Variant Call Format*) que contienen las diferencias con el ADN referencia.

En la última fase se analizan las mutaciones obtenidas y la estructura del

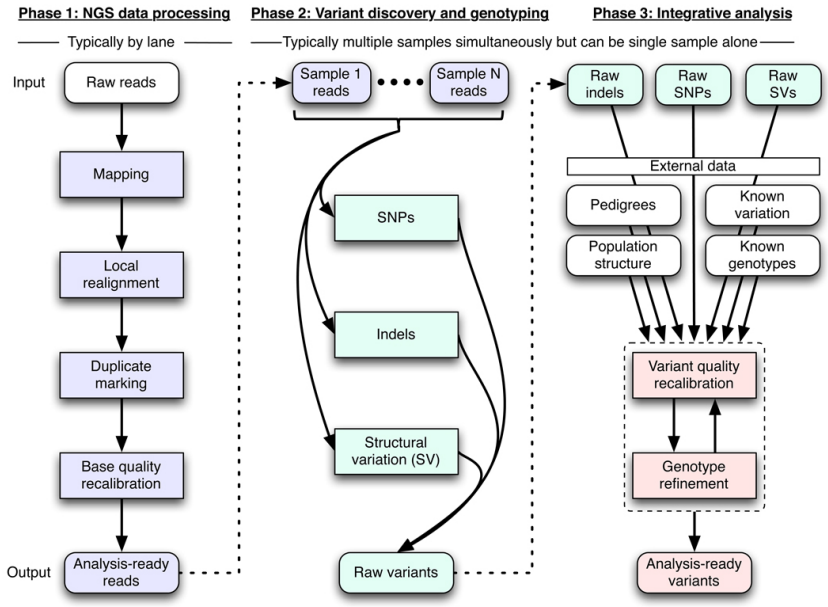

Figura 6.3: Proceso de descubrimiento de variantes en el ADN [5].

ADN para generar conclusiones. En la Figura 6.3 se puede ver gráficamente estas tres fases.

6.2. Punto de partida

GATK será la herramienta de referencia para la realización de esta tesis, abarcando parte de la primera fase y la segunda del proceso de descubrimiento de variantes. Según la Figura 6.3, se quiere implementar el recalibrado de las calidades de las bases (*Base quality recalibration*) de la primera fase del descubrimiento de variantes, desarrollando un recalibrador. También se quiere implementar la segunda fase completa, desarrollando un descubridor de variantes.

Las implementaciones tendrán que ser eficientes y aprovechar al máximo todos los recursos computacionales de los que se disponga en el entorno que se ejecuta, por ejemplo en un *cluster*. GATK no incorpora completamente estas características, puesto que solo aprovecha la capacidad de proceso mediante hilos y no utiliza estructuras de datos eficientes ni gestión de memoria óptima (en parte por utilizar Java).

6.3. Objetivos

6.3.1. Objetivo general

Desarrollar herramientas que permitan realizar el recalibrado de las calidades de las bases (*Base quality recalibration*) y toda la segunda fase del descubrimiento de variantes. Estas herramientas deberán ser eficientes con los recursos computacionales y con la gestión de memoria.

6.3.2. Objetivos específicos

El objetivo general de la tesis se puede desglosar en una serie de objetivos específicos, que mediante su cumplimiento permitirán alcanzar el objetivo general:

- Implementar un recalibrador que realice el recalibrado de la calidad de las bases y genere ficheros BAM ya recalibrados.
- Implementar un descubridor de variantes que realice toda la segunda fase de descubrimiento de variantes y genere ficheros de resultados VCF.
- Desarrollar implementaciones lo más eficientes posible, utilizando tecnologías paralelas que permitan explotar los recursos de cómputo disponibles en cada situación.
    - Usar las tecnologías disponibles en los microprocesadores para acelerar la ejecución.
    - Usar las capacidades de memoria compartida de los procesadores *multicore*.
    - Usar las capacidades de paso de mensaje de los sistemas multiprocesador o *clusters*.
    - Mantener las comunicaciones entre procesos al mínimo.
    - Reducir el número de operaciones I/O al mínimo.
- Usar estructuras de datos óptimas que permitan aprovechar las características de las arquitecturas y el sistema de memoria de los computadores. Explotar la localidad temporal y espacial de los datos en memoria.

6.4. Planificación de tareas

En esta sección se presenta un listado de las diferentes tareas a llevar a cabo para la consecución de los objetivos anteriormente marcados en este capítulo. En la Figura 6.4 se muestra la planificación temporal de estas tareas mediante un diagrama de Gantt. La tesis doctoral se llevará a cabo en 4 años, habiendo pasado ya casi un año en el momento de escribir este documento.

La Tabla 6.1 muestra de forma resumida qué tareas se van a llevar a cabo y su duración aproximada. Hay que destacar que algunas tareas se tienen que realizar durante los 4 años de tesis de forma continua, como es la documentación de la propia Tesis y la publicación de resultados o difusión. A continuación se describe en detalle en qué consiste cada una de las tareas.

6.4.1. Descripción de tareas

En esta sección se describirán las tareas a realizar durante el desarrollo de la tesis doctoral.

1 - Realización del máster

Esta tarea consiste en la superación del curso de Máster de Tecnologías Informáticas Avanzadas de la Universidad de Castilla-La Mancha. Para ello se superarán las asignaturas ofertadas por el máster que tengan que ver con el tema de investigación. Esta tarea acaba con la realización de este documento, describiendo el progreso alcanzado durante el primer año de tesis, y su defensa ante un tribunal.

2 - Estado del arte

Esta tarea corresponde a la documentación y estudio del estado del arte en modelos y sistemas de programación paralela, tratamiento de grandes cantidades de datos, y situación actual de las herramientas utilizadas en bioinformática para el análisis de ADN. Los resultados de este estudio y sus conclusiones se recogen en los capítulos de “Estado del arte” de este documento.

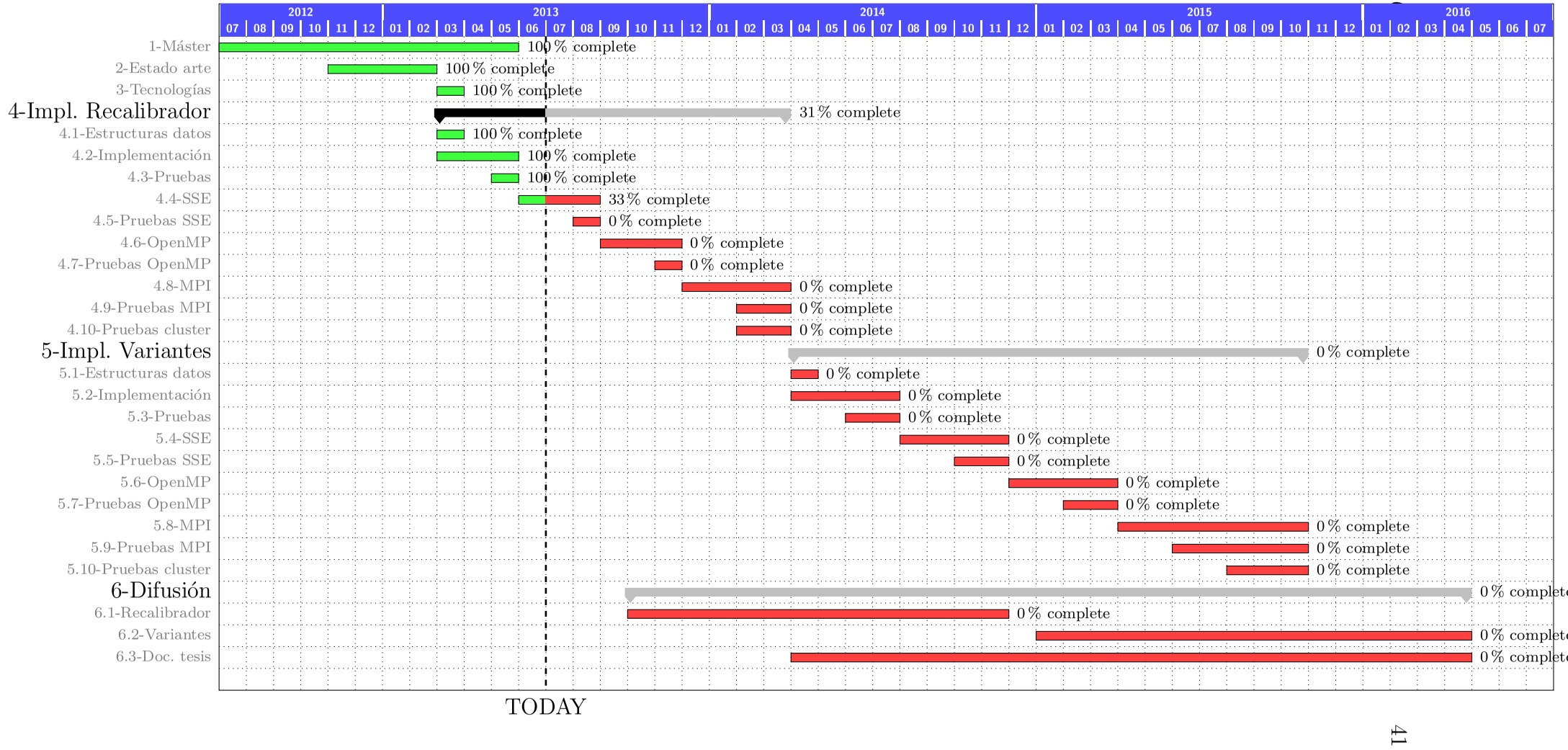

Figura 6.4: Diagrama Gant: desarrollo de la tesis de doctorado.

3 - Elección de tecnologías

Elección de tecnologías que se usarán en el desarrollo del software para el descubrimiento de variantes. Es importante la elección del lenguaje de programación, que determinará las librerías a utilizar y su adecuación a los entornos de computación de altas prestaciones. Otro aspecto a tener en cuenta es el modelo de programación en sistemas distribuidos que se utilizará (memoria compartida, paso de mensajes...). Puesto que el objetivo es obtener la máxima eficiencia a través del paralelismo, hay que elegir las tecnologías de cómputo paralelo viables y las herramientas para manejarlas. En la elección de tecnologías se tiene que contar con el consenso del grupo de investigación del CIPF.

4.1 - Elección de estructuras de datos para el recalibrador

Esta tarea engloba el diseño y elección de las estructuras de datos que se utilizarán en la implementación del recalibrador. Estas estructuras de datos deberán ser eficientes en el uso de la memoria y almacenar la información necesaria para el funcionamiento del algoritmo. Puesto que se busca la eficiencia, hay que buscar un compromiso entre almacenar el mínimo número de datos necesarios para obtener un buen rendimiento sin llenar la memoria disponible. En la elección de estructuras de datos se tiene que contar con el consenso del grupo de investigación del CIPF.

4.2 - Implementación del recalibrador

Implementar el recalibrador de forma secuencial, imitando el comportamiento del recalibrador incorporado en GATK pero ya introduciendo las mejoras de utilizar código y estructuras de datos eficientes. Dado un fichero BAM de entrada, el código debe procesarlo y producir un fichero BAM de salida con las calidades recalibradas, en un tiempo menor que el utilizado por el recalibrador de GATK.

4.3 - Pruebas del recalibrador

Esta tarea consiste en evaluar si efectivamente el recalibrador desarrollado genera una salida correcta y el rendimiento que obtiene, comparándolo

Num	**Actividad**	**Descripción**	**Tiempo (meses)**
1	Máster	Realización de los cursos del máster de tecnologías informáticas avanzadas	11
2	Estado arte	Búsqueda de información y recopilación del estado del arte de las tecnologías de computación paralelas y bioinformática	4
3	Tecnologías	Elección de las tecnologías a utilizar durante el desarrollo de las herramientas	1
4		***Implementación del recalibrador***	
4.1	Estructuras datos	Elección y diseño de las estructuras de datos que se utilizarán en el recalibrador	1
4.2	Implementación	Implementación de un recalibrador secuencial en C	3
4.3	Pruebas	Pruebas para evaluar la validez de los datos obtenidos y el rendimiento del recalibrador secuencial	1
4.4	SSE	Utilización de la tecnología SSE para aumentar el rendimiento del recalibrador	3
4.5	Pruebas SSE	Pruebas para evaluar la validez de los datos obtenidos y el rendimiento del recalibrador con SSE	1
4.6	OpenMP	Utilización de OpenMP para aumentar el rendimiento del recalibrador	3
4.7	Pruebas OpenMP	Pruebas para evaluar la validez de los datos obtenidos y el rendimiento del recalibrador con SSE + OpenMP	1
4.8	MPI	Utilización de MPI para aumentar el rendimiento del recalibrador	4
4.9	Pruebas MPI	Pruebas para evaluar la validez de los datos obtenidos y el rendimiento del recalibrador con SSE + OpenMP + MPI	2
4.10	Pruebas cluster	Pruebas para evaluar la validez de los datos obtenidos y el rendimiento del recalibrador con SSE + OpenMP + MPI en un sistema cluster	2
5		***Implementación del descubridor de variantes***	
5.1	Estructuras datos	Elección y diseño de las estructuras de datos que se utilizarán en el descubridor	1
5.2	Implementación	Implementación de un descubridor secuencial en C	4
5.3	Pruebas	Pruebas para evaluar la validez de los datos obtenidos y el rendimiento del descubridor secuencial	2
5.4	SSE	Utilización de la tecnología SSE para aumentar el rendimiento del descubridor	4
5.5	Pruebas SSE	Pruebas para evaluar la validez de los datos obtenidos y el rendimiento del descubridor con SSE	2
5.6	OpenMP	Utilización de OpenMP para aumentar el rendimiento del descubridor	4
5.7	Pruebas OpenMP	Pruebas para evaluar la validez de los datos obtenidos y el rendimiento del descubridor con SSE + OpenMP	2
5.8	MPI	Utilización de MPI para aumentar el rendimiento del descubridor	7
5.9	Pruebas MPI	Pruebas para evaluar la validez de los datos obtenidos y el rendimiento del descubridor con SSE + OpenMP + MPI	5
5.10	Pruebas cluster	Pruebas para evaluar la validez de los datos obtenidos y el rendimiento del descubridor con SSE + OpenMP + MPI en un sistema cluster	3
6		***Difusión***	
6.1	Recalibrador	Publicación de los resultados y conclusiones obtenidos en la implementación del recalibrador	14
6.2	Variantes	Publicación de los resultados y conclusiones obtenidos en la implementación del descubridor de variantes	16
6.3	Doc. tesis	Documentación y escritura de Tesis Doctoral	25

Tabla 6.1: Planificación de tareas y duración.

con el rendimiento del recalibrador de GATK.

4.4 - Introducción de instrucciones SSE en el código del recalibrador

Una vez implementado el recalibrador, hay que acelerar los cálculos utilizando instrucciones SSE donde sea posible. En principio esto debería acelerar las partes de código donde se utilice, gracias al modelo de sistema SIMD.

4.5 - Pruebas SSE en el recalibrador

Estas pruebas se centrarán en las partes de código del recalibrador que implementen operaciones con SSE. Estas partes de código deberán ser comparadas con su respectiva versión del algoritmo secuencial inicialmente obtenido. Para ello se analizará el rendimiento utilizando SSE comparándolo con el obtenido en la versión sin SSE. También debe comprobarse la validez de los ficheros BAM de salida.

4.6 - Implementación de OpenMP en el recalibrador

Una vez incorporadas instrucciones SSE al recalibrador, se implementará el uso de OpenMP en el mísmo. Esto permitirá utilizar las capacidades de memoria compartida de los recursos computacionales, incrementando el rendimiento.

4.7 - Pruebas OpenMP en el recalibrador

Al igual que en las pruebas descritas anteriormente, el objetivo de esta tarea es la validación de los resultados obtenidos por el recalibrador y una comparación del rendimiento obtenido respecto a las anteriores versiones. En este caso lo que se evalúa son las partes de código que utilicen OpenMP, comprobando el correcto funcionamiento de la esta nueva funcionalidad añadida junto a la funcionalidad anterior basada en SSE.

4.8 - Implementación de MPI en el recalibrador

La ultima funcionalidad a añadir al recalibrador es el soporte multiprocesador utilizando la librería MPI. Esta tarea consiste por tanto en la implementación del modelo de paso de mensajes para permitir el reparto de trabajo entre múltiples procesadores, utilizando MPI. Esto permitirá incrementar el rendimiento cuantos más procesadores disponga el sistema en el que se ejecute el recalibrador.

4.9 - Pruebas MPI en el recalibrador

Estas pruebas consisten en validar los resultados obtenidos por el recalibrador que incorpora MPI y su mejora de rendimiento respecto a las versiones anteriores. Para ello se realizarán pruebas con la implicación de varios procesadores.

4.10 - Pruebas en cluster del recalibrador con SSE + OpenMP + MPI

Esta tarea evalúa el recalibrador desarrollado en un entorno *cluster* con cientos de nodos. En este entorno se probarán todas las funcionalidades implementadas en el recalibrador de forma simultánea (SSE + OpenMP + MPI), aprovechando los recursos que ofrecen los procesadores del *cluster*. Se evaluará también el rendimiento final obtenido mediante el uso de estos modelos de programación paralelos y cual es el resultado final en cuanto a rendimiento y eficiencia ganado al utilizar el recalibrador desarrollado.

5 - Desarrollo del descubridor de variantes

Las tareas llevadas a cabo para el desarrollo de un descubridor de variantes son análogas a las llevadas a cabo en el desarrollo del recalibrador. Primeramente se diseñarán las estructuras de datos que serán necesarias para la correcta ejecución de los algoritmos y manejar eficientemente la memoria.

Una vez diseñadas las estructuras de datos se implementará una primera versión del descubridor de variantes, la cual servirá de base para añadir más funcionalidades. Hechas las correspondientes pruebas a la primera versión del descubridor de variantes, se procederá a la implementación y prueba de las

funcionalidades SSE, OpenMP y MPI en el mismo. Finalmente se probará el descubridor de variantes obtenido en un sistema *cluster* con cientos de nodos, evaluando su eficiencia y rendimiento.

6.4.2. Situación actual

En el momento de la elaboración de este documento, se han logrado desarrollar las primeras tareas hasta la implementación de las instrucciones SSE en el algoritmo del recalibrador, estando actualmente en la tarea de pruebas de SSE. Una descripción más detallada de las tareas completadas y en progreso se presentará en el capítulo 7, junto a una visión global sobre lo que se ha alcanzado durante la realización del año de máster.

Capítulo 7

Trabajo de investigación: Recalibrador de altas prestaciones usando paralelismo

En este capítulo se detalla el trabajo llevado a cabo hasta la fecha en que es redactado este documento. Se presentarán las tareas realizadas y hasta donde se ha llegado, según la planificación del anteproyecto de tesis. El objetivo del trabajo de máster es conseguir un software recalibrador acelerado, usando técnicas de optimización y paralelismo que sean más eficientes y rápidas que las utilizadas por el recalibrador de GATK.

7.1. Descripción de un recalibrador

El ADN se obtiene mediante máquinas de propósito específico llamadas secuenciadores. Se basan en técnicas bioquímicas cuya finalidad es la determinación del orden de los nucleótidos en la secuencia de ADN. Hay distintos factores que influyen en la toma de lecturas del ADN, que pueden llevar a errores. Estos factores se expresan mediante una calidad de error que el secuenciador asigna a cada lectura, en el momento de tomarla, dependiendo de qué posibilidad hay de que sea errónea.

La probabilidad de que una lectura sea errónea se encuentra en un intervalo de enteros, denominado calidad *Sanger* [46]. Para ello se aplica la fórmula

$$Q = -10 \times \log_{10} P \tag{7.1}$$

cuya entrada es un rango de números reales [0,1] y su salida está en el rango

de números naturales positivos [0,93] (acotado superiormente). La razón de utilizar esta calidad es que solo es necesario usar un byte para su expresión, aunque se pierda precisión en la probabilidad de error.

Un recalibrador ajusta las calidades asignadas a cada lectura en un fichero *BAM* para que sean más cercanas a la probabilidad de error real, teniendo en cuenta todas las lecturas del fichero y comparando si difieren del ADN de referencia.

Por ejemplo, si tenemos un fichero, aún no recalibrado, donde todas sus bases tienen calidad 25 y nos encontramos al recalibrar que 1 de cada 100 lecturas difieren del ADN referencia, entonces su calidad sería 20 (aplicando la fórmula 7.3) y se ajustarían las calidades teniendo esto en cuenta. Pero el recalibrador tiene en cuenta más cosas, como que las lecturas de los últimos ciclos del secuenciador presentan más errores que las primeras, o la probabilidad de que se den determinadas combinaciones de dos nucleótidos en las lecturas (contexto del dinucleótido).

7.1.1. Formato *BAM/SAM*

SAM [45] es un formato de texto para almacenar los datos de secuenciación de ADN en una serie de columnas ASCII tabuladas. Este formato está pensado para ser legible por humanos. *BAM* es el formato alternativo a *SAM*, almacenando los datos de forma binaría, comprimida e indexada.

Los formatos *BAM/SAM* contienen la misma información aunque en diferente formato. Están compuestos por una cabecera opcional, que puede contener información adicional sobre el fichero y las lecturas que contiene, seguida de las lecturas de secuenciación. Cada lectura contiene información de la secuencia de ADN que contiene y cómo se alinea con el ADN de referencia.

7.2. Tareas completadas o en progreso

7.2.1. Estado del arte

Se ha completado la tarea de búsqueda de información y obtención del estado del arte sobre: herramientas disponibles para el análisis de ADN, tec-

nologías paralelas útiles para aumentar el rendimiento de dichas herramientas y formas de tratar grandes cantidades de datos.

7.2.2. Elección de tecnologías

Una de las decisiones a tomar al comenzar el trabajo consistía en elegir el lenguaje de programación. El software de referencia, *GATK*, utiliza Java [42]. Este lenguaje es ampliamente utilizado en el desarrollo de aplicaciones web y de servicios debido a su facilidad de uso, además de tener una curva de aprendizaje rápida. Es un lenguaje orientado a objetos e interpretado, lo cual permite su ejecución en distintas plataformas sin necesidad de recompilar. Su gestión de la memoria es automática.

Se desechó Java puesto que es interpretado por su máquina virtual (*JVM Java Virtual Machine*), lo cual produce una sobrecarga en la ejecución que no compensa su portabilidad. Además su gestión automática de memoria es también un inconveniente en sistemas donde el uso de la memoria es crítico para el rendimiento.

Se optó finalmente por utilizar C, puesto que es un lenguaje que te permite hacer cualquier cosa, aunque no sea tan fácil como Java. La otra variante, C++, se desechó puesto que no necesitamos un enfoque orientado a objetos. C permite un control total sobre la memoria y sus datos, lo cual permite optimizar el código para que se ajuste al sistema de memoria donde se alojará y utilizar estructuras de datos óptimas, con la consecuente ganancia en rendimiento. Además el código en C es compilado y optimizado por el propio compilador. Permite fácilmente la inclusión de código en ensamblador, lo cual puede ser un punto crítico en la ejecución de algunas operaciones básicas ejecutadas masivamente en el código. Como ventaja final, existe una amplia variedad de librerías para este lenguaje, depuradas y optimizadas durante años que facilitan bastante la programación, llegando incluso a ganar rendimiento.

En cuanto a las librerías a utilizar para aprovechar el paralelismo y obtener mayor rendimiento se ha optado por MPI, OpenMP y SSE. Las instrucciones SSE permiten aprovechar las capacidades SIMD de los procesadores actuales, lo cual permitirá aumentar el rendimiento en determinadas operaciones que involucren datos almacenados en vectores de forma continua. OpenMP permitirá aprovechar los distintos *cores* de cada procesador, puesto que actualmente todos los procesadores son *multicore*. Finalmente MPI permitirá la ejecución en entornos de múltiples procesadores, como puede ser

un *cluster*. La combinación de estas tres tecnologías nos permitirá explotar los recursos de cómputo basados en microprocesador de forma eficiente.

Otro recurso, presente actualmente en la mayoría de *clusters*, son las GPU. Las GPU nos permitirían ganar mucho rendimiento en operaciones masivas sobre vectores o matrices, por lo que podríamos orientar las estructuras de datos para que utilicen este tipo de operaciones. Con ello podríamos utilizar GPU cuando estuvieran disponibles en el sistema en que se ejecuta la herramienta, con la consiguiente ganancia de rendimiento que se obtendría.

Esta tarea ha sido completada.

7.2.3. Elección de estructuras de datos para el recalibrador

Las estructuras de datos debían ser eficientes y almacenar información sobre diversos tipos de contadores. Se ha llegado a la conclusión de que la mejor estructura de datos a utilizar en este caso son las matrices, pues que esta forma de almacenamiento permite aprovechar la localidad espacial y temporal en memoria al estar todos los datos de forma continua. En la sección 7.4.4 se describen detalladamente las estructuras de datos que se han utilizado, puesto que se entienden mejor una vez introducido el algoritmo de recalibrado (sección 7.4). Esta tarea ha sido completada.

7.2.4. Implementación del recalibrador

Se ha desarrollado el código de un algoritmo recalibrador secuencial, que utiliza eficientemente las estructuras de datos que se han diseñado y por tanto la memoria. El algoritmo intenta utilizar el mínimo de operaciones necesarias para los cálculos, haciendo especial hincapié en las operaciones *I/O*. Esta tarea ha sido completada.

7.2.5. Pruebas del recalibrador

Esta tarea ha sido completada. Se han realizado diversas pruebas de rendimiento, sobre un fichero BAM de entrada en el recalibrador obtenido y el que incorpora la herramienta GATK. Las pruebas procuraron evaluar ambas herramientas en igualdad de condiciones. Como se verá en la sección

7.5, los resultados obtenidos en cuanto a rendimiento han sido prometedores desde la primera versión.

7.2.6. Implementación de SSE en el recalibrador

Para este trabajo se incorporó el uso de las instrucciones *SSE* [12], puesto que el compilador de C proporciona las funciones optimizadas para su uso. En caso de no existir estas funciones, se pueden utilizar las instrucciones *SSE* directamente en ensamblador (puesto que C lo permite). El uso de *SSE*, en las partes del código que lo permitían, aumentaron el rendimiento como se verá más adelante.

Esta tarea está en progreso pero ya se implementa en la parte de código que compara las cadenas de ADN en la primera fase para detectar si es fallo.

7.2.7. Pruebas SSE en el recalibrador

Esta tarea se ha iniciado de forma parcial. Ya se ha realizado alguna prueba sobre las instrucciones SSE que se van incorporando. En la sección 7.5 se verá el rendimiento obtenido al utilizar estas instrucciones.

7.3. Herramientas utilizadas

Para el desarrollo del recalibrador se utilizaron varios tipos de herramientas, tanto para la edición del código como para su depurado. A continuación serán descritas.

7.3.1. Editor de código y compilación

Geany [47] es un editor de texto con algunas funciones de los entornos de desarrollo integrados (*IDEs*). Es sencillo y rápido, ofreciendo una forma simple de manejar los ficheros de un proyecto. Puesto que lo he utilizado sólo para editar el código, no me han sido necesarias funciones adicionales de otros *IDEs* o de compilación.

Para compilar el código he utilizado *Makefile*, el cual es ampliamente utilizado para la compilación automática, usando ficheros de texto. Lo he utilizado sin herramientas de compilación adicionales, pero existen otras herramientas que generan automáticamente los *Makefiles* y los actualizan sin necesidad de que el programador intervenga. Un ejemplo de estas herramientas es el *GNU build system* o *Autotools* [48]. *Autotools* será tenido en cuenta para trabajo futuro, además de otra herramienta llamada *Scons* [49] basada en scripts de *Phyton*.

7.3.2. Control de versiones

Para el control de versiones del código se ha utilizado *Git* [50], que permite guardar el historial de cambios que ha sufrido el programa durante su desarrollo y permite revertir cambios para recuperar versiones anteriores. Ofrece un control de versiones distribuido, al contrario que otras herramientas como *Subversion* [51], por lo que se puede trabajar en local, y cuando sea posible, combinar los cambios con el repositorio global. La razón de elegir *Git* fue su facilidad de uso, por ejemplo a la hora de crear y combinar distintos flujos de trabajo.

La forma de trabajar que se ha seguido con *Git* consiste en tener un flujo de trabajo llamado *master* donde se recogen las características ya implementadas en el programa. A la hora de añadir una nueva funcionalidad al programa se crea un nuevo flujo de trabajo y se llevan a cabo los cambios sobre el mismo. Cuando esa funcionalidad está añadida, se combina ese flujo de trabajo con el *master* obteniendo el programa con la característica implementada en ese flujo principal. Esto permite trabajar en varías funcionalidades de forma separada, a la vez y sin interferir unas sobre otras, lo cual facilita el manejo del código.

7.3.3. Depuración

GDB [52] (*GNU* debugger) es un depurador de código en línea de comandos. Entre otras cosas permite comenzar la ejecución de un programa e incluir puntos de ruptura. En caso de que el programa termine su ejecución de forma anormal, permite ver el estado de las variables y la pila de programa justo en el momento del error. Mayoritariamente se ha usado este programa para encontrar el origen de los fallos de segmentación que se han encontrado durante el desarrollo del recalibrador. Cuando ocurre un

fallo de segmentación el propio sistema operativo vuelca el contenido de la memoria en ese momento en un fichero. *GDB* permite restaurar la ejecución al momento del fallo usando ese fichero y ofreciendo una visión del estado del programa en ese momento.

El otro programa que se ha utilizado para depuración es *Valgrind* [53]. Es un conjunto de herramientas de depuración que permiten detectar automáticamente muchos fallos en el manejo de memoria e hilos, permitiendo analizar los programas en detalle.

Durante el desarrollo del recalibrador se ha utilizado la herramienta *memcheck* de *Valgrind*. Esta herramienta permite detectar errores en la memoria, como accesos no válidos, uso de variables no inicializadas, liberación incorrecta de memoria reservada, fugas de memoria... El principal uso que se le ha dado ha sido localizar puntos del código del recalibrador donde no se liberaba bien la memoria, provocando que la memoria libre se agote y disminuyendo el rendimiento de forma drástica.

7.4. Algoritmo de recalibrado

El algoritmo de recalibrado usado se compone de dos fases. La primera fase consiste en recopilar datos sobre los ficheros *BAM*. La segunda fase es la recalibración en sí, utilizándose los datos obtenidos en la primera fase para ajustar las calidades de *BAM* usando fórmulas estadísticas. A continuación se describe el proceso formal de recalibrado y cómo se realizan las dos fases en el algoritmo.

7.4.1. Proceso formal de recalibrado

Las calidades de un fichero *BAM* se expresan mediante un número entero en el rango $[0, 93]$. Este rango indica una probabilidad de error que se obtiene a partir de la fórmula 7.2. Según esta fórmula, si tenemos que una lectura de una base tiene como probabilidad de error 0,01 (1 de cada 100 será errónea), la calidad en el *BAM* será 20. Esta forma de representar la probabilidad de error hace que se pierda precisión pero a cambio permite representar la calidad usando un único byte (un carácter).

$$Q_{sanger}(p) = -10 \times \log_{10}(p) \tag{7.2}$$

Obtener la probabilidad de error a partir de una calidad es posible utilizando la operación inversa de la fórmula 7.2 (usando la fórmula 7.3).

$$P_{sanger}(q) = 10^{\frac{-q}{10}} \tag{7.3}$$

La calidad recalibrada de una base se obtiene al aplicar la fórmula 7.4, la cual tiene como entrada los parámetros de la posición de la base leída en el ADN de referencia, el ciclo del secuenciador en que se ha leído y el contexto de dinucleótido (par compuesto por la base anterior y la actual, por ejemplo, "AG").

$$Q^*(r, c, d) = Q(r) + \Delta Q + \Delta Q(r) + \Delta Q(r, c) + \Delta Q(r, d) \tag{7.4}$$

La nueva calidad se obtiene a partir de la calidad original de esa posición ($Q(r)$), sumándole las variaciones o *deltas* que influyen en la misma. Estos *deltas* se calculan según las fórmulas 7.5, 7.6, 7.7 y 7.8.

$$\Delta Q = Q_{sanger}(T_g) - Q_{sanger}\left(\frac{\sum_r P_{sanger}(Q(r)) \times \textit{número bases } r}{\textit{número bases global}}\right) \tag{7.5}$$

$$\Delta Q(r) = Q_{sanger}(T(r)) - Q(r) - \Delta Q \tag{7.6}$$

$$\Delta Q(r, c) = Q_{sanger}(T(r, c)) - Q(r) - (\Delta Q + \Delta Q(r)) \tag{7.7}$$

$$\Delta Q(r, d) = Q_{sanger}(T(r, d)) - Q(r) - (\Delta Q + \Delta Q(r)) \tag{7.8}$$

Puesto que para calcular cada *delta* es necesario contabilizar las lecturas y fallos del fichero que se recalibra, son necesarios dos procesamientos o pasadas sobre el fichero. La primera pasada o fase de recogida de datos contabiliza las bases que lee y si son fallos, datos que necesita para calcular finalmente los *deltas*. Una vez se calculan los *deltas* se hace una segunda pasada que aplica a cada calidad la fórmula que obtiene su calidad final recalibrada.

En la primera fase se recorre el fichero *BAM* contabilizando las lecturas y sus errores, almacenándolos en las estructuras de datos. Con estos datos se puede obtener la tasa de fallos T que se define en la fórmula 7.9, cuyo resultado es utilizado en el cálculo de los deltas. Una vez se ha recorrido el fichero y obtenidas las T, se calculan finalmente los deltas para cada combinación *posición-ciclo-dinucleótido*.

$$T = \frac{\textit{número de fallos}}{\textit{número de bases}} \tag{7.9}$$

Tomando un ejemplo donde hay un rango de calidades $[0, 49]$, 100 ciclos de secuenciador y 12 tipos de dinucleótido, el cálculo de los *deltas* se traduce en un *delta* global, 50 deltas (uno para cada posible calidad), $50 \times 100 = 5000$ *deltas* para cada par calidad-ciclo y $50 \times 12 = 600$ *deltas* para cada par calidad-dinucleótido. Esto se traduce en el cálculo de 5651 *deltas*.

En la segunda fase se vuelve a recorrer el fichero para aplicar la fórmula 7.4 que devuelve la calidad recalibrada para cada base leída. Puesto que tenemos todos los *deltas* calculados en las estructuras de datos, sólo nos llevará 4 sumas el cálculo de cada calidad.

Por ejemplo un fichero de un secuenciador de 100 ciclos, con 35 millones de lecturas (35×10^6) tendrá aproximadamente 3500 millones de bases ($3,5 \times 10^9$). Usando los *deltas* ya calculados sólo usaríamos 4 sumas por cada base, es decir, $4 \times 3,5 \times 10^9 = 14 \times 10^9$ sumas en total. En cambio, si calculásemos los *deltas* a la vez que cada calidad recalibrada necesitaríamos más operaciones que 4 sumas (varios ordenes de magnitud más). Esta es la razón de precalcular los *deltas* en la primera fase, aunque su almacenamiento signifique un coste en memoria adicional.

7.4.2. Fase 1: Recogida de datos

En esta fase se lleva a cabo una recogida de datos sobre el fichero *BAM* que se quiere analizar. Como ya se explicó en la sección 7.1.1, un fichero *BAM* contiene un conjunto de lecturas de *ADN*. Lo que se pretende es contabilizar la tasa de fallos de las lecturas de *BAM* respecto al *ADN* de referencia. La tasa de fallos se obtiene mediante la expresión de la fórmula 7.9. Es por ello que hay que contabilizar tanto los fallos que se han producido como las bases que se han leido.

Para ello el *BAM* de entrada se procesa por lotes, es decir, se lee un conjunto de lecturas del fichero y se procesan. Inicialmente esto se hace de forma secuencial, leyendo un lote y procesándolo. Cuando se termina de procesar este lote se lee el siguiente. Los lotes se leen secuencialmente en el fichero *BAM* usando la librería *samtools* [45].

El procesado de cada lote consiste en leer por orden las lecturas que contiene. Cada una de estas lecturas tiene una secuencia de bases (nucleótidos) en forma de cadena de caracteres, indicando además qué posición de inicio representan en el *ADN* de referencia. Lo que se hace para cada lectura es leer el *ADN* de referencia en la misma posición y la misma longitud, obteniendo la cadena de caracteres del *ADN* de referencia. La lectura de la cadena de referencia se realiza a través de *samtools*.

Una vez se tiene una lectura, con N bases, y su secuencia correspondiente en el *ADN* de referencia, se comparan entre sí. La comparación se realiza base a base, es decir, la base de la lectura en la posición x se compara con la base en la posición x de la referencia. Si las bases no son iguales se contabiliza como un fallo.

Como son necesarias las tasas de fallo a distintos niveles, se contabilizan por separado en las estructuras de datos. La forma de contabilizar los fallos en una base consiste en utilizar la calidad asociada a esa base. Si es un fallo, se incrementa:

- El contador de fallos y de lecturas global (T_g),
- el contador de fallos y de lecturas para el valor de esa calidad ($T(r)$),
- el contador de fallos y de lecturas para el par calidad-ciclo ($T(r, c)$) y
- el contador de fallos y de lecturas para el par calidad-dinucleótido ($T(r, d)$).

En caso de que no sea fallo sólo se incrementarían los contadores de lecturas correspondientes.

Una vez se han recorrido y contabilizado todas las lecturas del *BAM* de entrada, se calculan las tasas de fallos en los diferentes niveles usando los contadores de fallos y lecturas correspondientes. Con estas tasas de fallos se calculan los *deltas* correspondientes. Finalmente tendremos en las estructuras de datos, el *delta* global (ΔQ), el *delta* para cada valor de calidad ($\Delta Q(r)$), el *delta* para cada par calidad-ciclo ($\Delta Q(r, c)$) y el *delta* para cada par calidad-dinucleótido ($\Delta Q(r, d)$).

Para incrementar el rendimiento de esta fase, se han utilizado instrucciones *SSE/SEE2*, concretamente para realizar las comparaciones de bases entre la secuencia de la lectura y la secuencia de referencia. Estas dos secuencias se almacenan de forma contigua en memoria (vector de tipos *char* o enteros de un byte).

Lo que se hace pues, en cada iteración, es cargar en un registro *XMM* de 128 bits (utilizados por las instrucciones SSE) 16 bases del vector de la secuencia leída, utilizando una sola lectura de memoria. En otro registro *XMM* se cargan las 16 bases correspondientes al vector de referencia, mediante otra lectura de memoria. Acto seguido se usa la instrucción de comparación de *SSE2* sobre las 16 bases, obteniendo en un registro *XMM* los resultados de las comparaciones (*0*x*FF* si es igual o *0*x*00* si no). Finalmente se guardan en una variable vector los resultados de las 16 comparaciones con otra instrucción de memoria.

Como efecto resultante al usar *SSE*, los vectores se recorren de 16 en 16 elementos. Si no usamos *SSE* los elementos se recorrerían de 1 en 1. Esto tiene un gran impacto en el rendimiento como se verá en hanla sección 7.5.

7.4.3. Fase 2: Recalibrado

En esta fase se lleva a cabo el proceso de recalibrado en sí. Al igual que en la fase anterior, el fichero *BAM* de entrada se procesa por lotes. El fichero *BAM* con las calidades recalibradas que se genera, se escribe por lotes conforme estos han sido procesados y en el mismo orden que se leyeron.

El procesado de cada lectura consiste en aplicar únicamente la fórmula 7.4 a cada una de las calidades de sus bases, sustituyendo las mismas por

los nuevos valores. Puesto que los *deltas* ya han sido calculados en la fase anterior, se pueden utilizar directamente en los cálculos accediendo a sus valores en las estructuras de datos. Por tanto, el valor final de las calidades corresponde a las cuatro sumas de los *deltas* correspondientes. Finalmente se genera el fichero *BAM* con las nuevas calidades.

7.4.4. Estructura de datos utilizadas

Las estructuras de datos utilizadas para el recalibrado consisten en matrices. Cada *delta* que se calcula necesita de 3 matrices. Una matriz acumula el número de bases que se leen, otra matriz acumula el número de fallos que se han detectado y la ultima matriz almacena los *deltas* calculados. Puesto que hay 4 tipos de *delta* (global, posición, posición-ciclo y posición-dinucleótido), son necesarias 12 matrices.

Las matrices para el *delta* global tiene dimensión 1×1, puesto que sólo hay un caso. Las matrices para el delta de posición tiene dimensión $1 \times n$ siendo n el número de posibles calidades que puede tener una posición. Para los delta de posición-ciclo y posición-dinucleótido la dimensión es $n \times c$ y $n \times d$ respectivamente, siendo c el número de ciclos y d el número de dinucleótidos (lo normal es que d sea 12).

Por ejemplo, tenemos un fichero *BAM* de un secuenciador de 100 ciclos, con 35 millones de lecturas (35×10^6) tendrá aproximadamente 3500 millones de bases ($3,5 \times 10^9$), con calidades comprendidas en el rango $[0, 50]$. Las estructuras de datos utilizadas en este caso serán:

- Para el *delta* global: tres matrices de dimensión 1×1 que contengan el número de bases leídas en total, el número de fallos y su delta. Hacen un total de 2 enteros y un número en coma flotante (*double* en C).

- Para el *delta* de cada calidad: tres matrices de dimensión 1×50 que contengan el número de bases leídas que tengan esa calidad, sus fallos y su *delta*. Hacen un total de 100 enteros y 50 *double*.

- Para el *delta* de cada calidad-ciclo: tres matrices de dimensión 50×100 que contengan el número de bases leídas que tengan esa calidad en ese ciclo, sus fallos y su *delta*. Hacen un total de 10000 enteros y 5000 *double*.

Parámetro	Descripción
`-F`	Ejecuta el proceso de recalibrado completo
`-1`	Ejecuta la primera fase del proceso de recalibrado
`-2`	Ejecuta la segunda fase del proceso de recalibrado
`-C <int>`	Determina el número de ciclos máximo que se analizarán por lectura
`-R <reference>`	Ruta al fichero del ADN referencia
`-I <input>`	Ruta del fichero BAM a recalibrar
`-o <output>`	Ruta donde se almacenará el fichero BAM recalibrado
`-d <data>`	Ruta donde se guarda el fichero de datos resultado de la fase 1
`-i <info>`	Ruta donde se guarda el fichero de datos resultado de la fase 1 en formato de texto
`--help`	Muestra la ayuda sobre los parámetros
`--version`	Muestra la versión del programa

Tabla 7.1: Parámetros configurables del recalibrador desarrollado.

- Para el *delta* de cada calidad-dinucleótido: tres matrices de dimensión 50×12 que contengan el número de bases leídas que tengan esa dinucleótido, sus fallos y su *delta*. Hacen un total de 1200 enteros y 600 *double*.

En el ejemplo anterior tenemos finalmente unos requisitos de memoria de 11302 enteros y 5651 *double*. Teniendo en cuenta que el tamaño de un entero largo son 4 bytes y los tipo *double* son 8 bytes, necesitaremos 90416 bytes (90,5Kb aproximadamente). Actualmente 90 Kb es un tamaño muy pequeño para una memoria de un sistema de altas prestaciones, por lo que el sobrecoste de precalcular los *deltas* es despreciable. Con esto se ha conseguido un uso eficiente de la memoria.

7.4.5. Parámetros de ejecución del recalibrador

En el momento de la elaboración de este documento, el recalibrador obtenido posee algunos parámetros configurables y opciones de ejecución que serán explicados a continuación y recogidos en la Tabla 7.1.

El recalibrador desarrollado cuenta con 2 fases de ejecución, pudiendo ejecutarse sólo una de ellas o ambas. Estas fases corresponden a las fases de recalibrado explicadas en la sección 7.4. Para ejecutar las dos fases se utilizaría el parámetro `-F`.

La primera fase se ejecuta con el parámetro `-F` o `-1`. Esta fase necesita obligatoriamente que los parámetros del número máximo de ciclos (`-C`), la

ruta del fichero de ADN de referencia (`-R`) y la ruta del BAM de entrada (`-I`) sean especificados. Opcionalmente se pueden especificar los parámetros `-d` y `-i`. El parámetro `-d` genera un fichero que contiene las estructuras de datos generadas en la primera fase, por lo que se puede utilizar para realizar la segunda fase de recalibrado sin necesidad de repetir la primera. El parámetro `-i` genera un fichero de texto que muestra los resultados tabulados de la primera fase de recalibrado (el contenido del fichero de datos que se genera con `-d`).

La segunda fase se ejecuta con el parámetro `-F` o `-2`. Esta fase tiene como parámetros obligatorios un fichero BAM de entrada (`-I`), el número máximo de ciclos (`-C`), y un fichero de salida para el BAM recalibrado (`-o`). Si se ejecuta esta fase sin haber ejecutado anteriormente la primera (sólo con el parámetro `-2`), es necesario especificar el fichero de datos que se utilizará y que se generó en la primera fase (`-d`).

7.5. Evaluación del recalibrador obtenido

Como ya se ha visto en la sección 7.2 de tareas completadas, ya se han realizado las pruebas sobre la implementación del recalibrador. En esta sección se describirán estas pruebas y sus resultados.

Las primeras pruebas se realizaron una vez implementada la versión secuencial del nuevo recalibrador, usando las estructuras de datos optimizadas y las optimizaciones automáticas que el compilador ofrece, a excepción de SSE2. El objetivo era conocer el rendimiento que se obtiene con la nueva herramienta comparándolo con el obtenido en las mismas condiciones para la herramienta recalibrador que incorpora GATK.

El entorno de prueba se detalla en la Tabla 7.2. Consiste en un sistema de un solo procesador de 4 núcleos de procesamiento con 4 GB de memoria principal. Las pruebas se han realizado utilizando un BAM de entrada con 35×10^6 lecturas, que con 100 ciclos por entrada hacen un total de $3{,}5 \times 10^9$ bases a recalibrar. Además sólo se ha utilizado un núcleo del procesador.

Los dos recalibradores han realizado el mismo proceso de recalibrado en las mismas condiciones, teniendo en cuenta la posición de las bases, su ciclo de secuenciador y su entorno de dinucleótido. Las pruebas se pueden dividir en tres, primeramente la comparación de la herramienta de

Procesador	Intel Core 2 Quad 2.66GHz, 4 núcleos
Memoria principal	4 GB
Fichero BAM entrada	35×10^6 lecturas
Número ciclos por lectura	100 ciclos
Número total de bases	$35 \times 10^6 \times 100$ ciclos $= 3{,}5 \times 10^9$ bases
Calidad máxima de las lecturas	50

Tabla 7.2: Detalles del sistema utilizado y las pruebas del nuevo recalibrador.

GATK con el recalibrador desarrollado sin SSE2, siendo la segunda prueba una comparación entre la versión del recalibrador sin SSE2 con la que lo implementa de forma automática mediante el compilador. Finalmente la ultima prueba compara el rendimiento obtenido por el recalibrador con las con SSE2 automáticas activadas y una pequeña parte de incluidas de forma manual.

7.5.1. Evaluación del recalibrador sin SSE2

En esta prueba se ha evaluado primeramente el rendimiento de GATK, al cual se le ha pasado como entrada el fichero BAM descrito anteriormente y se ha generado otro fichero BAM recalibrado. El tiempo que tardó GATK en recalibrar el fichero fueron 2685 segundos. Teniendo en cuenta el número de bases que se analizan, obtenemos un *throughput* de $\frac{3{,}5\times10^9}{2685} \approx 1,30 \times 10^6$ bases/segundo.

Al evaluar el tiempo de ejecución de la versión del recalibrador que utiliza únicamente las estructuras de datos optimizadas con SSE automáticas (sin SSE2), se obtuvo el BAM resultado en 880 segundos. Al igual que con GATK se obtiene el *throughput* y se obtiene $\frac{3{,}5\times10^9}{720} \approx 4,86 \times 10^6$ bases/segundo.

Comparando los dos *throughput*, el nuevo recalibrador es mucho más productivo que el implementado por GATK, en términos de *SpeedUp* se ha conseguido $\frac{2685}{720} = \mathbf{3{,}73}$ como factor de aceleración. Este factor nos indica que efectivamente la herramienta desarrollada es más rápida que la herramienta de GATK.

Puesto que no sería justo comparar las dos herramientas sin que GATK utilice su capacidad *multithreading*, se ejecutó con 4 hilos de procesamiento en el mismo entorno de pruebas. El tiempo que obtuvo en este caso es de 2040 segundos, por lo que aumentó su *throughtput* pero con una eficiencia

de sólo $\frac{2685}{2040}/4 * 100 \approx 33\,\%$. Aún así, el nuevo recalibrador es más rápido utilizando un solo hilo de procesamiento.

7.5.2. Evaluación del recalibrador con SSE2 automáticas

Para estas pruebas se activaron las optimizaciones que el compilador GCC implementa automáticamente para utilizar SSE2. Con estas optimizaciones el recalibrador procesó el fichero BAM en sólo 670 segundos, por lo que si lo comparamos directamente con GATK obtenemos un *SpeedUp* de $\frac{2685}{670} = \mathbf{4{,}01}$. Lo cual nos indica que el nuevo recalibrador es hasta 4 veces más rápido que el de GATK en un solo núcleo de procesamiento y utilizando instrucciones SSE2 de forma automática.

7.5.3. Evaluación del recalibrador con SSE2 automáticas y manual

Esta prueba pretende comparar el rendimiento del recalibrador cuando implementa instrucciones SSE2 de forma manual en la primera fase de recalibrado (concretamente en la función que procesa las comparaciones de cada lectura). Se compara respecto a la versión anterior con SSE2 automáticas. Puesto que sólo las implementa actualmente para la primera fase de recalibrado, solo se tendrán en cuenta los tiempos en esta fase, concretamente en lo que tarda en procesar una lectura (de media). Para procesar una lectura sobre el fichero BAM de prueba se necesitaron de media 3.42 microsegundos usando SSE2 de forma manual. En la versión con SSE2 automáticas, el procesado de cada lectura tardó de media 3.64 microsegundos, lo que indica que hubo una mejora de rendimiento menor, es decir, con SSE es un 6 % más rápido.

Aún siendo una pequeña mejora de rendimiento (sólo se ha utilizado SSE2 de forma manual en una pequeña parte del código total) podemos deducir que es factible sacar más rendimiento cuando lo implementemos en el resto.

7.6. Conclusiones

Actualmente se está implementando SSE2 de forma manual en algunas partes de código que son críticas en el rendimiento, con la intención de aprovechar al máximo los recursos de computo de los que se dispone. Esto, junto a la implementación automática de SSE y SSE2 y otras técnicas de optimización del compilador, permite aumentar la eficiencia como ya se ha visto en este capítulo. El siguiente paso puede ser utilizar SSE3 o SSE4, que no son implementadas automáticamente y pueden afectar al rendimiento. Otra razón para implementar manualmente SSE2, es que en procesadores de 32 bits, el compilador no genera código SSE2 automáticamente, por lo que se perdería esta ventaja. También se perdería si utilizamos otro compilador que tampoco genere este tipo de código de forma automática.

Durante la realización de este trabajo se ha logrado el objetivo de conseguir un recalibrador eficiente, aunque aun queda mucha funcionalidad por añadirle. Este recalibrador ya explota estructuras de datos optimizadas y además lo hace unas 4 veces más rápido que GATK, por lo que se va por buen camino. Con la implementación del modelo de memoria compartida con OpenMP y el de paso de mensajes MPI se espera obtener un gran aumento de rendimiento, además de permitir la utilización de un supercomputador. Esto permitiría realizar el descubrimiento de variantes en tiempo adecuado para que sea factible su aplicación, por ejemplo, en el ámbito de la sanidad entre otras.

Capítulo 8

Conclusiones generales

En este capítulo se presentará un resumen de las conclusiones a las que se ha llegado durante la realización de este trabajo.

Se han estudiado las principales tecnologías de computación paralela y su programación, ofreciendo una forma de acelerar el rendimiento de un software mediante su ejecución en un entorno *cluster*. Además, podemos utilizar GPUs para acelerar aún más los procesos que utilicen operaciones vectoriales masivas o que encajen bien con procesadores SIMD. Esto nos permitirá desarrollar herramientas eficientes que sean más rápidas cuantos más procesadores tengan disponibles en un *cluster*.

El tratamiento de datos cuando se manejan de forma masiva es crítico para el rendimiento. Un *dataset* más grande que la memoria principal disponible supondría no poder cargarlo completamente en memoria y por tanto acceder continuamente a disco. Esto supone un decremento importante en las prestaciones. Por tanto se hace necesario un modo de tratar estos datos que permita obtener un acceso por partes, parecido al paradigma *MapReduce*. Una de las soluciones, en un entorno *cluster*, es usar las memorias de todos los nodos disponibles ya que se incrementa la probabilidad de que los datos se carguen en memoria completamente, y por tanto aumente el rendimiento.

También se han visto herramientas utilizadas actualmente en bioinformática para el análisis de ADN. La mayoría de ellas son secuenciales o permiten un mínimo paralelismo de grano grueso utilizando hilos. Otras permiten su ejecución en entornos *cluster* pero son para aplicaciones muy específicas, lo que obliga a ejecutarlas a modo tubería. Se ve por tanto una necesidad de tener un grupo de herramientas, que permitan realizar análisis

completos sin necesidad de herramientas extra. Además estas herramientas ganarían en prestaciones y permitirían explotar el paralelismo en un sistema *cluster*, permitiendo su uso cotidiano en diversos ámbitos de aplicación.

La creación de un conjunto de herramientas que cumplan los requisitos anteriores es factible. Para ello se utilizarán tecnologías de paralelismo que exploten los recursos de cómputo actuales. Se pretende crear este conjunto dentro de la comunidad científica, ofreciendo a la misma la posibilidad de participar y realizar sus análisis sin restricciones.

Durante el primer año ya se han producido avances significativos. Se ha implementado una de las herramientas necesarias en el inicio del proceso de descubrimiento de variantes en C, utilizando estructuras de datos optimizadas para memoria. El rendimiento respecto a la herramienta que se ha tomado como referencia (GATK) es mucho mayor, incluso con esta utilizando hilos, lo cual nos indica el bajo grado de rendimiento que tienen actualmente estas herramientas.

Se han dado los primeros pasos y ya se han obtenido resultados esperanzadores, si todo sale bien dentro de unos años podrían utilizarse estas herramientas de forma cotidiana. Cada vez que un paciente fuese a la consulta del médico no tendría que preguntarle qué le pasa, realizarle un análisis de su ADN sería suficiente para determinar qué le ha pasado, qué le pasa y qué le va a pasar, del mismo modo que te hacen un análisis de sangre. Teniendo en cuenta además que las aplicaciones de este tipo de herramientas no se limitan a la medicina, las posibilidades son muy amplias.

Bibliografía

[1] "OpenMP wikipedia." http://en.wikipedia.org/wiki/OpenMP. Accessed: 2013-05.

[2] T. White, *Hadoop: The Definitive Guide*. Oreilly and Associate Series, Oreilly & Associates Incorporated, 2012.

[3] "Genome Analysis Toolkit webpage." http://www.broadinstitute.org/gatk/. Accessed: 2013-05.

[4] T. Krude, F. Villa, and A. Barbero, *ADN*. Akal/ciencia, Ediciones Akal, 2008.

[5] M. A. DePristo, E. Banks, R. Poplin, K. V. Garimella, and J. R. Maguire, "A framework for variation discovery and genotyping using next-generation DNA sequencing data," *Nature genetics*, vol. 43, no. 5, 2011.

[6] M. P. Sawicki, G. Samara, M. Hurwitz, and E. P. Jr., "Human genome project," *The American Journal of Surgery*, vol. 165, pp. 258–264, 2006.

[7] B. Alberts, A. Johnson, J. Lewis, M. Raff, K. Roberts, , and P. Walter, *Molecular Biology of the Cell*. Garland Science, 2002.

[8] P. E, L. J, and A. A, "Generations of sequencing technologies," *Genomics*, vol. 93, pp. 105–111, 2009.

[9] T. Issariyakul and E. Hossain, *An Introduction to Network Simulator NS2*. SpringerLink : Bücher, Springer Science+Business Media, LLC, 2012.

[10] *Unlocking the Power of OPNET Modeler*. Cambridge University Press.

[11] M. Bernardo and F. Corradini, *Formal Methods for the Design of Real-Time Systems: International School on Formal Methods for the Design*

of Computer, Communication, and Software Systems, SFM-RT 2004. Revised Lectures. Lecture Notes in Computer Science, Springer, 2004.

[12] Intel, *Intel 64 and IA-32 Architectures Software Developer's Manual*, vol. 1. 2013.

[13] A. Grama, A. Gupta, G. Karypis, and V. Kumar, *Introduction to Parallel Computing*. Addison Wesley, 2003.

[14] M. J. Flynn, "Some Computer Organizations and Their Effectiveness," *IEEE Transactions on Computers*, vol. c-21, no. 9, pp. 948–949, 1972.

[15] M. J. Quinn, *Parallel Programming in C with MPI and OpenMP*. McGraw-Hill, 2003.

[16] D. Luebke, M. Harris, N. Govindaraju, A. Lefohn, M. Houston, J. Owens, M. Segal, M. Papakipos, and I. Buck, "GPGPU: general-purpose computation on graphics hardware," in *Proceedings of the 2006 ACM/IEEE conference on Supercomputing*, SC '06, (New York, NY, USA), ACM, 2006.

[17] W. R. Stevens and S. A. Rago, *Advanced Programming in the UNIX Environment: Second dition*. Addison Wesley Professional, 2005.

[18] R. Chandra, L. Dagum, D. Kohr, D. Maydan, J. McDonald, and R. Menon, *Parallel Programming in OpenMP*. Morgan Kaufmann Publishers, 2001.

[19] O. ARB, *OpenMP Application Program Interface: Version 4.0 RC2*. 2013.

[20] N. Matloff, *Programming on Parallel Machines*. University of California, Davis.

[21] W. Gropp, E. Lusk, and A. Skjellum, *Using MPI : Portable Parallel Programming with the Message-passing Interface Scientific and Engineering Computation*. MIT Press, 1999.

[22] A. Geist, *Pvm: Parallel Virtual Machine : A Users' Guide and Tutorial for Networked Parallel Computing*. Scientific and Engineering Computation Series, Cambridge ; London : The MIT Press, 1994.

[23] L. Surhone, M. Tennoe, and S. Henssonow, *Open Mpi*. Betascript Publishing, 2011.

[24] G. Burns, R. Daoud, and J. Vaigl, *LAM: An Open Cluster Environment for MPI*. 1994.

[25] T. Stitt, "An Introduction to the Partitioned Global Address Space Programming Model," *CNX.org*, 2010.

[26] T. El-Ghazawi, W. Carlson, T. Sterling, and K. Yelick, *UPC: Distributed Shared Memory Programming*. Wiley Series on Parallel and Distributed Computing, Wiley, 2005.

[27] R. W. Numrich and J. Reid, "Co-array Fortran for parallel programming," *SIGPLAN Fortran Forum*, vol. 17, pp. 1–31, Aug. 1998.

[28] K. Yelick, L. Semenzato, G. Pike, C. Miyamoto, B. Liblit, A. Krishnamurthy, P. Hilfinger, S. Graham, D. Gay, P. Colella, , and A. Aiken, "Titanium: A high-performance java dialect,"

[29] P. Charles, C. Grothoff, V. Saraswat, C. Donawa, A. Kielstra, K. Ebcioglu, C. von Praun, and V. Sarkar, "X10: an object-oriented approach to non-uniform cluster computing," *SIGPLAN Not.*, vol. 40, pp. 519–538, Oct. 2005.

[30] B. Chamberlain, D. Callahan, and H. Zima, "Parallel programmability and the chapel language," *International Journal of High Performance Computing Applications*, vol. 21, pp. 291–312, 2007.

[31] J. Bueno, J. Planas, and A. Duran, "Productive Programming of GPU Clusters with OmpSs," *IEEE 26th International Parallel and Distributed Processing Symposium*, pp. 557–568, 2012.

[32] "IBM Bigdata state of the art webpage." http://www-01.ibm.com/software/data/bigdata/. Accessed: 2013-05.

[33] P. Warden, *Big Data Glossary*. O'Reilly Media, 2011.

[34] J. Dean and S. Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters," *Proceedings of the 6th Symposium on Operating System Design and Implementation*, pp. 137–150, 2004.

[35] E. Capriolo, D. Wampler, and J. Rutherglen, *Programming Hive*. O'Reilly Media, 2012.

[36] "Cascalog GitHub site." https://github.com/nathanmarz/cascalog. Accessed: 2013-05.

[37] "mrjob GitHub site." https://github.com/Yelp/mrjob. Accessed: 2013-05.

[38] S. L. Garfinkel and S. L. Garfinkel, "An evaluation of amazon grid computing services: EC2, S3, and SQS," tech. rep., Center for, 2007.

[39] R. C. Gentleman1, V. J. Carey2, D. M. Bates3, B. Bolstad4, M. Dettling5, S. Dudoit4, B. Ellis6, L. Gautier7, Y. Ge8, J. Gentry1, K. Hornik9, T. Hothorn10, W. Huber11, S. Iacus12, R. Irizarry13, F. Leisch9, C. Li1, M. Maechler5, A. J. Rossini14, G. Sawitzki15, C. Smith16, G. Smyth17, L. Tierney18, J. Y. Yang19, and J. Zhang1, "Bioconductor: open software development for computational biology and bioinformatics," *Genome Biology*, vol. 5, 2004.

[40] M. J. Crawsley, *The R Book*. Wiley, 2012.

[41] J. Tisdall, *Mastering Perl for Bioinformatics*. O'Reilly Media, 2010.

[42] F. Durán, F. Gutiérrez, and E. Pimentel, *Programación orientada a objetos con Java*. Ediciones Paraninfo. S.A., 2007.

[43] B. Langmead, C. Trapnell, M. Pop, and S. L. Salzberg, "Ultrafast and memory-efficient alignment of short DNA sequences to the human genome," *Genome Biology*, vol. 10, 2009.

[44] I. Medina, J. Tarraga, F. Salavert, and C. Y. Gonzalez, "OpenCB webpage." http://www.opencb.org. Accessed: 2013-05.

[45] H. Li, B. Handsaker, A. Wysoker, T. Fennell, J. Ruan, N. Homer, G. Marth, G. Abecasis, and R. Durbin, "The Sequence Alignment/Map format and SAMtools," *Bioinformatics*, vol. 25, 2009.

[46] P. J. A. Cock, C. J. Fields, N. Goto, M. L. Heuer, and P. M. Rice, "The Sanger FASTQ file format for sequences with quality scores, and the Solexe/Illumina FASTQ variants," *Nucleic Acids Res*, pp. 1767–1771, 2010.

[47] "Geani editor website." http://www.geany.org/. Accessed: 2013-05.

[48] J. Calcote, *Autotools: A Practitioner's Guide to GNU Autoconf, Automake, and Libtool*. No Starch Press Series, No Starch Press, 2010.

[49] S. Knight, "Building software with SCons," *Computing in Science & Engineering*, vol. 7, pp. 79–88, 2005.

[50] J. Loeliger and M. McCullough, *Version Control with Git: Powerful Tools and Techniques for Collaborative Software Development*. Oreilly and Associate Series, O'Reilly Media, Incorporated, 2012.

[51] "Subversion website." http://subversion.tigris.org/. Accessed: 2013-05.

[52] N. Matloff and P. Salzman, *The Art of Debugging with GDB, DDD, and Eclipse*. No Starch Press Series, No Starch Press, 2008.

[53] N. Nethercote and J. Seward, "Valgrind: a framework for heavyweight dynamic binary instrumentation," *SIGPLAN Not.*, vol. 42, pp. 89–100, June 2007.

Currículum vítae

Datos Personales

Nombre	:	Raúl Moreno Galdón
DNI	:	47.095.741-K
Nacimiento	:	1 de octubre 1990, España
Nacionalidad	:	Español
Residencia	:	Profesor Macedonio Gimenez 15, 3N, Albacete, Castilla-La Mancha, España
e-mail	:	raulmorenogaldon@gmail.com

Formación Académica

2008 - 2012	:	Grado en Ingeniería Informática, nota - 9
2012 - 2013	:	Máster en Tecnologías Informáticas Avanzadas

Formación Complementaria

20 horas - 2012	:	Curso de uso de técnicas probabilísticas para localización de un autómata
20 horas - 2012	:	Seminario sobre "Presente y futuro de los sistemas de computación"

Experiencia Laboral

Agosto 2012	:	Diseño y programación de protocolos de comunicación inalámbricos sobre redes de sensores en la *spin-off* RADIS
Febrero 2012 - Abril 2012	:	Diseño y programación de un sistema de control, basado en microprocesador y bus CAN, de un prototipo de silla para minusválidos automática. *Spin-off* RADIS.