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

5 de junio 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: Dr. ...

Tutor: Dr. ...

Cotutor: Dr. ...

5 de junio de 2013

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

Este capítulo trata de ofrecer una visión general sobre la linea de investigación que se quiere seguir durante la realización de esta tesis doctoral y cuales 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 [25, 13] (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 [25, 13] o ácido ribonucleico) y además se transmite de generación en generación.

Un gen [25, 13] es una región de ADN que influye en una característica particular de un organismo (color de los ojos por ejemplo). Estos 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 [25] 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 [33] 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 varías máquinas simultáneamente para obtener resultados. Además será necesario contar con mecanismos de gestión y tratamiento de grandes cantidades de datos para poder manejar los que producen los secuenciadores.

1.2. Estructura de la memoria

?????

Capítulo 2

Asignaturas Cursadas

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.

2.1.3. Relación con el tema de investigación

Esta asignatura está relacionada con todos los temas de investigación, puesto que para cualquiera necesitas 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[17] y OPNET[19]. 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*[23] 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 que analizando su modelo pueda 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 con 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*[37] (*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 [35] 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 utiliza 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 [34, 41] 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 Single 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.

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 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 simultanea.

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.

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 (POSIX), directivas (OpenMP) y uso de GPUs (CUDA).

Los POSIX [42] (*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 [42] es la interfaz de programación con hilos que ofrecen las librerías POSIX. Esta interfaz nos ofrece funciones simples y potentes necesarias para crear cualquier código que utilice hilos.

OpenMP [41, 29, 26] es un API de programación paralela (basada en directivas del compilador, rutinas de librería y variables de entorno), en C/C++ y Fortran, para multiprocesadores de memoria compartida. Puede ejecutarse en sistemas Unix y Windows NT. 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 utiliza por debajo el modelo *fork-join* (Figura 3.1) 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

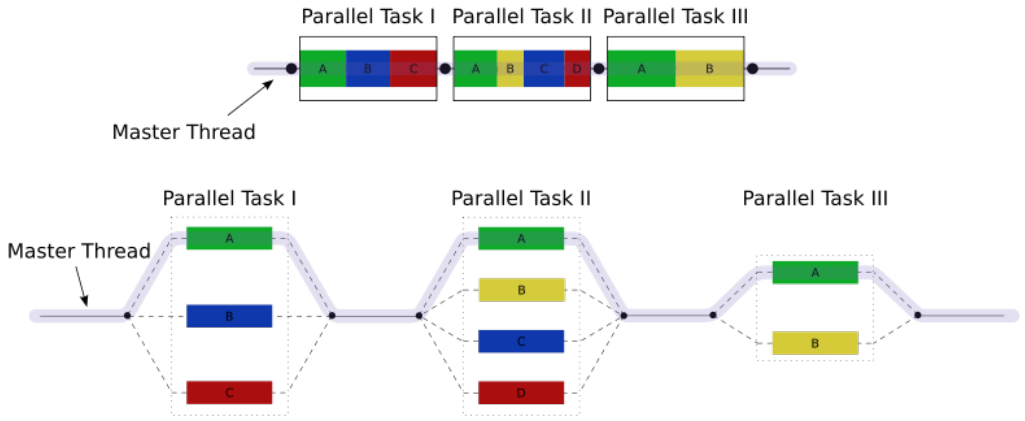

Figura 3.1: Modelo paralelo *Fork-Join* usado por *OpenMP*.

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*[39]. Las GPUs son máquinas SIMD según la taxonomía de Flynn (véase Sección 3.1) utilizando el modelo de memoria compartida. Una diferencia con los procesadores con varios núcleos 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 son las *SSE* [37] (*Streaming SIMD Extensions*) que no es un API en sí, sino 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 simultanea en 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 envío de mensajes.

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[39, 41, 36] (MPI) y Parallel Virtual Machine (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* [18], *LAM* [28] (*Local Area Multicomputer*) y *MPICH* [36] (*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 este 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* [36]. 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.

3.2.3. Partitioned Global Address Space - PGAS

El modelo de PGAS [43] 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.

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*, *Co-Array Fortran*, *Titanium*, *X-10* y *Chapel*.

OmpSS [27] 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 procesar-

los. 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 de ambos modelos de programación 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 computo 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 tanto económicamente como 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.

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.

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*[14]):

- **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 donde es crítico en sistemas de tiempo real.

- **Variedad**: No solo se manejan muchos datos, sino además estos pueden tener distinta estructura como pueden ser 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 [44, 31] 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, cómo 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* [44, 11], *Hive* [12], *Cascalog* [5], *mrjob* [16] ... A continuación se comentará *Hadoop* entre otros.

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.

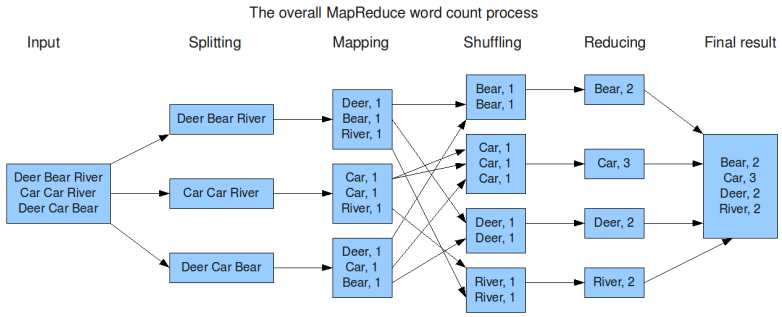

Figura 4.1: Modelo paralelo *MapReduce* para contar palabras.

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 y adicionalmente junto con información sobre el progreso de la ejecución.

Puesto que 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[44, 11] (*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 [44, 1] 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.

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 [2] 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*[20]. 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 [3] es un conjunto de herramientas bioinformáticas basadas en módulos del lenguaje de programación *Perl*. Está orientado a objetos, por lo que para llevar a cabo una tarea, algunos módulos necesitarán de otros para llevar a cabo su función. Posee, entre otras cosas, interfaces gráficas, almacenamiento persistente en bases de datos y procesadores de los resultados de varias de aplicaciones para poder usarlos. Se utiliza para producir nuevas herramientas más complejas.

5.3. Genome Analysis ToolKit

Genome Analysis Toolkit [32, 7] (*GATK*) es el estandard de facto para el análisis y descubrimiento de variantes de genoma. Es un conjunto de herramientas genérico que pueden ser aplicadas 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 herra-mientas 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 [15] 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.

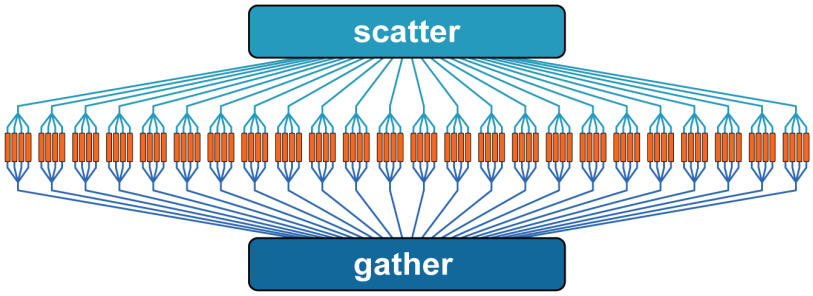

Figura 5.1: Modelo paralelo utilizado por *GATK* para el procesamiento en *cluster*.

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 [4] 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 [40] 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 grupo 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 del *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).

Lo que se pretende por tanto es replicar esta herramienta para conservar sus ventajas pero intentando eliminar sus desventajas. 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.

Capítulo 6

Trabajo de investigación: Recalibrador de altas prestaciones usando paralelismo

En este capitulo se detalla el trabajo llevado a cabo durante la realización del máster. Se presentara el ámbito en el que se ha trabajado, desde donde se ha partido y cuales han sido los pasos seguidos hasta llegar al final de este trabajo. Básicamente el objetivo del trabajo era conseguir un software recalibrador acelerado, usando técnicas de optimización y paralelismo que fuesen más eficientes y rápidas que las utilizadas por el de referencia.

Primeramente se describirá en que consiste el proceso de descubrimiento de variantes, qué función tiene un recalibrador en ese proceso, pasando seguidamente a lo que se ha realizado a partir de ahí y qué pasos se han seguido. Se hará una descripción del algoritmo implementado y sus estructuras de datos, haciendo finalmente una evaluación de rendimiento.

6.1. Introducción al descubrimiento de variantes

6.1.1. Introducción al ADN

El *ADN* [25] (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

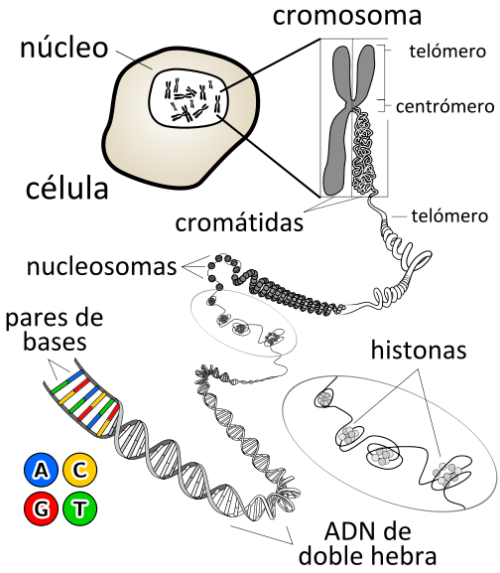

Figura 6.1: ¿Dónde se encuentra el ADN?.

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, está 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 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. Un gen 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...

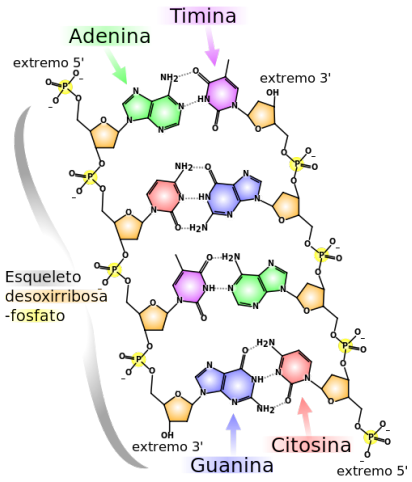

Figura 6.2: Representación química del ADN.

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.

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* [38] (*Sequence Aligment Map/Binary Alignment Map*), el cual es independiente de la tecnología de obtención del ADN.

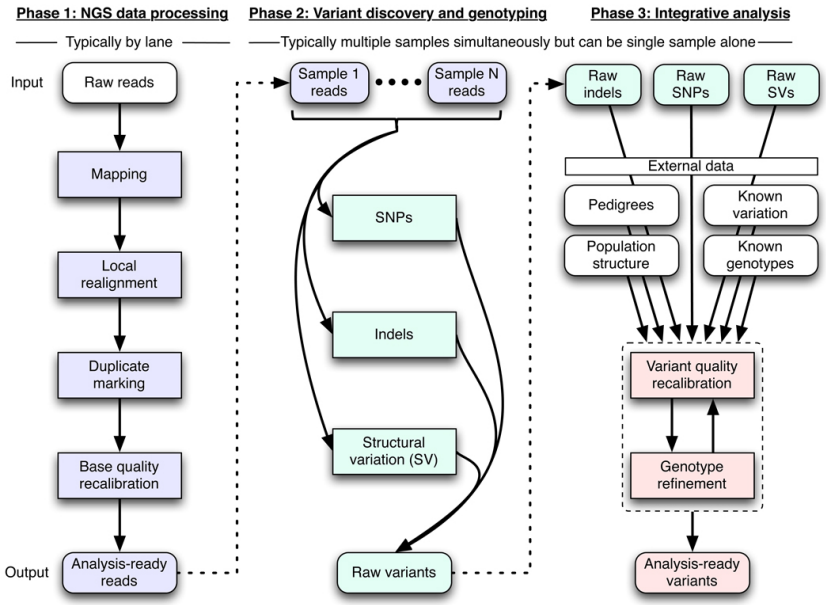

Figura 6.3: Proceso de descubrimiento de variantes en el ADN.

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*), pequeños *indels*, etc.

En la última fase se analizan las mutaciones obtenidas y la estructura del ADN para tomar conclusiones. En la Figura 6.3 se puede ver gráficamente estas tres fases.

6.1.3. 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 mapea en un rango de enteros, denominado calidad *Sanger* [30]. Para ello se aplica la fórmula

$$Q = -10 * \log_{10} P \tag{6.1}$$

cuya entrada es un rango de números reales [0,1] y su salida en el rango de número 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 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 las combinaciones de dos nucleótidos en las lecturas (contexto de dinucleótido).

6.1.4. Formato *BAM/SAM*

6.2. Búsqueda de información

6.3. 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 [15]. 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 el 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 opto 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 te 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 sin sacrificar rendimiento.

Para este trabajo se incorporó el uso de las instrucciones *SSE* [37], puesto que el compilador de C proporciona las funciones optimizadas para su uso, y en caso de no existir esas 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.

6.4. Diseño del software

6.5. 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.

6.5.1. Editor de código y compilación

Geany [6] 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* [9]. *Autotools* será tenido en cuenta para trabajo futuro, además de otra herramienta llamada *Scons* [21] basada en scripts de *Phyton*.

6.5.2. Control de versiones

Para el control de versiones del código se ha utilizado *Git* [8], esto me permite guardar la historia 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* [22], por lo que puedes 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 con *Git* que he seguido 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, creo un nuevo flujo de trabajo y realizo los cambios sobre el mismo. Cuando esa funcionalidad está añadida, combino ese flujo de trabajo con el *master* obteniendo el programa con la característica implementada en ese flujo principal. Esto me permite trabajar en varías funcionalidades de forma separada, a la vez y sin interferir unas sobre otras, lo cual facilita el manejo del código.

6.5.3. Depuración

GDB [10] (*GNU* debugger) es un depurador de código en línea de comandos. Entre otras cosas te permite comenzar la ejecución de un programa e incluir puntos de ruptura. En caso de que el programa termine su ejecución de forma anormal, te permite ver el estado de las variables y la pila de programa justo en el momento del error. Mayoritariamente he usado

este programa para encontrar el origen de los fallos de segmentación que me he 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 he utilizado para depuración es *Valgrind* [24]. Es un co njunto 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 he 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 le he dado ha sido localizar puntos del código del recalibrador donde no se liberaba bien la memoria, provocando que la memoria se llene y disminuyendo el rendimiento de forma drástica.

6.6. Algoritmo de recalibrado

El algoritmo de recalibrado usado se compone de dos fases. La primera fase consiste en

6.6.1. Recogida de datos

6.6.2. Recalibrado

6.7. Estructura de datos utilizadas

6.8. Evaluación del recalibrador obtenido

Bibliografía

[1] Amazon S3 service webpage. http://aws.amazon.com/es/s3/. Accessed: 2013-05.

[2] Bioconductor webpage. http://www.bioconductor.org. Accessed: 2013-05.

[3] Bioperl webpage. http://www.bioperl.org. Accessed: 2013-05.

[4] Bowtie webpage. http://bowtie-bio.sourceforge.net. Accessed: 2013-05.

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

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

[7] Genome Analysis Toolkit webpage. http://www.broadinstituthttp://www-01.ibm.com/software/data/bigdata/e.org/gatk. Accessed: 2013-05.

[8] Git website. http://git-scm.com/. Accessed: 2013-05.

[9] GNU build system manual. http://www.gnu.org/software/automake/manual/html_node/index.h Accessed: 2013-05.

[10] GNU Debugger website. http://http://www.gnu.org/software/gdb/. Accessed: 2013-05.

[11] Hadoop webpage. http://hadoop.apache.org/. Accessed: 2013-05.

[12] Hive: A data warehouse system for Hadoop webpage. http://hive.apache.org/. Accessed: 2013-05.

[13] Human Genome Project Information webpage. http://www.ornl.gov/sci/techresources/Human_Genome/project/. Accessed: 2013-05.

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

[15] Java website. https://www.java.com. Accessed: 2013-05.

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

[17] Network Simulator 2 webpage. http://www.isi.edu/nsnam/ns/. Accessed: 2013-05.

[18] OpenMPI webpage. http://www.open-mpi.org. Accessed: 2013-05.

[19] OPNET webpage. http://www.opnet.com/. Accessed: 2013-05.

[20] R Project for Statistical Computing webpage. http://www.r-project.org. Accessed: 2013-05.

[21] Scons website. http://www.scons.org/. Accessed: 2013-05.

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

[23] UPPAAL webpage. http://www.uppaal.org/. Accessed: 2013-05.

[24] Valgrind website. http://valgrind.org/. Accessed: 2013-05.

[25] Bruce Alberts, Alexander Johnson, Julian Lewis, Martin Raff, Keith Roberts, , and Peter Walter. *Molecular Biology of the Cell*. Garland Science, 2002.

[26] OpenMP ARB. *OpenMP Application Program Interface: Version 4.0 RC2*. 2013.

[27] Javier Bueno, Judit Planas, and Alejandro Duran. Productive Programming of GPU Clusters with OmpSs. *IEEE 26th International Parallel and Distributed Processing Symposium*, pages 557–568, 2012.

[28] Greg Burns, Raja Daoud, and James Vaigl. *LAM: An Open Cluster Environment for MPI*. 1994.

[29] Rohit Chandra, Leonardo Dagum, Dave Kohr, Dror Maydan, Jeff McDonald, and Ramesh Menon. *Parallel Programming in OpenMP*. Morgan Kaufmann Publishers, 2001.

[30] Peter J. A. Cock, Christopher J. Fields, Naohisa Goto, Michael L. Heuer, and Peter M. Rice. The Sanger FASTQ file format for sequences with quality scores, and the Solexe/Illumina FASTQ variants. *Nucleic Acids Res*, pages 1767–1771, 2010.

[31] Jeffrey Dean and Sanjay Ghemawat. MapReduce: Simplified Data Processing on Large Clusters. *Proceedings of the 6th Symposium on Operating System Design and Implementation*, pages 137–150, 2004.

[32] Mark A DePristo, Eric Banks, Ryan Poplin, Kiran V Garimella, and Jared R Maguire. A framework for variation discovery and genotyping using next-generation DNA sequencing data. *Nature genetics*, 43(5), 2011.

[33] Pettersson E, Lundeberg J, and Ahmadian A. Generations of sequencing technologies. *Genomics*, 93:105–111, 2009.

[34] Michael J. Flynn. Some Computer Organizations and Their Effectiveness. *IEEE Transactions on Computers*, c-21(9):948–949, 1972.

[35] Ananth Grama, Anshul Gupta, George Karypis, and Vipin Kumar. *Introduction to Parallel Computing*. Addison Wesley, 2003.

[36] William Gropp, Ewing Lusk, and Anthony Skjellum. *Using MPI : Portable Parallel Programming with the Message-passing Interface Scientific and Engineering Computation*. MIT Press, 1999.

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

[38] Heng Li, Bob Handsaker, Alec Wysoker, Tim Fennell, Jue Ruan, Nils Homer, Gabor Marth, Goncalo Abecasis, and Richar Durbin. The Sequence Alignment/Map format and SAMtools. *Bioinformatics*, 25, 2009.

[39] Norm Matloff. *Programming on Parallel Machines*. University of California, Davis.

[40] Ignacio Medina, Joaquin Tarraga, Francisco Salavert, and Cristina Y. Gonzalez. OpenCB webpage. http://www.opencb.org. Accessed: 2013-05.

[41] Michael Jay Quinn. *Parallel Programming in C with MPI and OpenMP*. McGraw-Hill, 2003.

[42] W. Richarch Stevens and Stephen A. Rago. *Advanced Programming in the UNIX Environment: Second dition*. Addison Wesley Professional, 2005.

[43] Tim Stitt. An Introduction to the Partitioned Global Address Space Programming Model. *CNX.org*, 2010.

[44] Pete Warden. *Big Data Glossary*. O'Reilly Media, 2011.