Índice general

Capítulo 1

Introducción

1.1. Introducción y motivación

El ADN de todos los organismos vivos conocidos contiene las instrucciones genéticas usadas en su desarrollo y funcionamiento, es cómo una plantilla de la que saldrá el organismo final. El papel de la molecula 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 (proteinas y moleculas de ARN) y además se transmite de generación en generación.

Un gen 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 de 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 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 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 destinados al análisis del ADN, pero ninguna ha conseguido una alta calidad y completitud en cuanto a:

- 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 pueden ser los clusters.
- Falta de documentación.

Estos inconvenientes se convierten en un cuello de botella cuando los precios de los recursos de computo 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

Estado del arte

2.1. Entornos paralelos

La computación paralela [12] 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 mas problemas más difíciles de abordar, el tratamiento de grandes datasets, llegando a ser de los más grandes en el ámbito científico. Analizar estos datasets requiere una gran cantidad de potencia computacional 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.

2.1.1. Taxonomía de Flynn

La taxonomía de Flynn [11, 16] 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 mas viables que otras.

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

Después tenemos los 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 GPU, las cuales están siendo ampliamente utilizadas en computación científica.

Los MISD tienen varíos 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 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 correr varios procesos simultáneamente sobre varios conjuntos de datos. A mayor escala tendríamos los cluster, 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.

2.1.2. Modelos de programación paralela

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

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.

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 locks para evitar el acceso simultaneo 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.

Algunos paradigmas de programación utilizados en este modelo son los threads (POSIX), directivas (OpenMP) y uso de GPUs (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 utilizan.

En un sistema UNIX, un proceso no es mas 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 librerías POSIX. Esta interfaz nos ofrece funciones simples y potentes necesarias para crear cualquier código que utilice hilos.

OpenMP[16, 8, 6] 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 correr 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 2.1) 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. Además 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 tener un snapshot de la memoria 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*[14]. Las GPUs son máquinas SIMD según la taxonomía de Flynn (véase Sección 2.1.1) utilizando el modelo de memoria compartida. Una diferencia con los multicores que también utilizan este modelo, es que las GPUs pueden tener miles de hilos corriendo simultáneamente. Un programa en CUDA tiene dos partes, el código que se

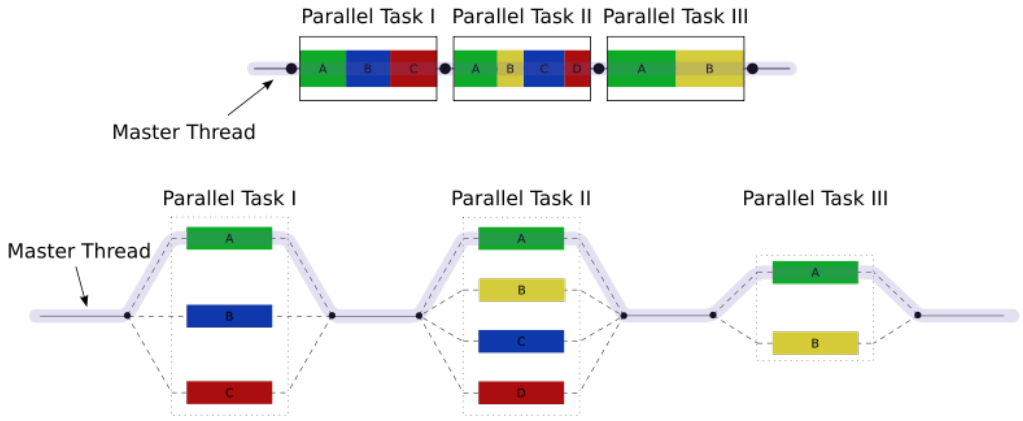

Figura 2.1: Modelo paralelo *Fork-Join* usado por *OpenMP*.

ejecuta en el *host* o procesador central (CPU), y el código que se ejecuta en el *device* (GPU) llamado *kernel*. Debido a esto es necesaria una CPU que trabaje en conjunto con la GPU. Cualquier aplicación que siga encaje en 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.

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 cores. En este modelo, la interacción entre procesos corriendo en distintos nodos se realiza mediante en el envío de mensajes.

Los mensajes proporcionan un modo de transferir datos, tareas y sincronizar procesos. Puesto que la interacción se consigue mediante envio/recepción de mensajes, las operaciones básicas de este paradigma de programación son *send* y *receive*. Es necesario además un mecanismo que asigne un ID único 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 (MPI) y Parallel Virtual Machine (PVM).

Message Passing Interface[14, 16, 13] 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. Tiene implementaciones tanto en código libre, como propietarias. Entre las implementaciones más conocidas se encuentran *Open MPI*, *LAM* y *MPICH*.

La forma de procesar, en un entorno MPI, un programa 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*[13]. Esta API es la parte de MPI que maneja el I/O, ofreciendo distintas operaciones de lectura y escritura. Permite además determinar como se van a distribuir los datos entre los procesos, y como 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.

Partitioned Global Address Space - PGAS

El modelo de PGAS[18] 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

OmpSS[7] es un modelo de programación que utiliza como base OpenMP, 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.

2.2. Tratamiento de grandes cantidades de datos

Una de las características más significativas en el tratamiento de genoma, es la gran cantidad de datasets que se generan. Estos datos hay que manejarlos de forma eficiente y adecuada, para preservar su integridad y 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.

2.2.1. MapReduce

MapReduce[9] 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

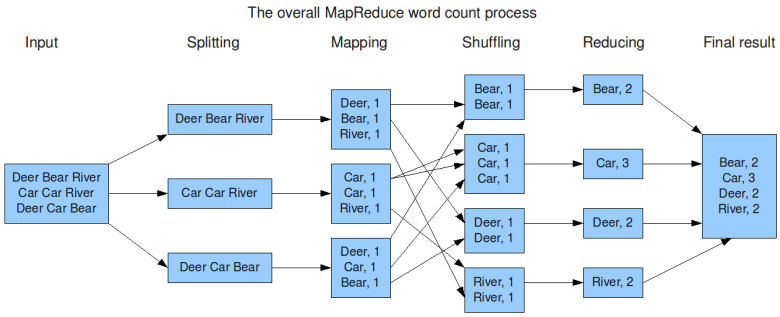

Figura 2.2: Modelo paralelo *MapReduce* para contar palabras.

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 computo 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. Todo esto incluido en una simple librería, así fue como nació MapReduce (de mano de Google).

El computo 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 2.2 muestra gráficamente un ejemplo para contar palabras usando el modelo *MapReduce*.

2.3. 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 utilizado 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.

2.3.1. Bioconductor

Bioconductor[1] 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*[5]. 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.

2.3.2. Bioperl

Bioperl[2] 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 parseadores de los resultados de varias de aplicaciones para poder usarlos. Se utiliza para producir nuevas

herramientas más complejas.

2.3.3. Genome Analysis ToolKit

Genome Analysis Toolkit[10, 4] (*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* está formado por un engine que se ocupa de correr 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 y permitiendo su 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 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.

Para el procesamiento a gran escala utiliza una estrategia scatter-gather a groso modo que consiste en partir los datos de entrada y ejecutar cada parte en distintas máquinas independientemente. En la Figura 2.3 se representa el modelo scatter-gather que utiliza el *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.

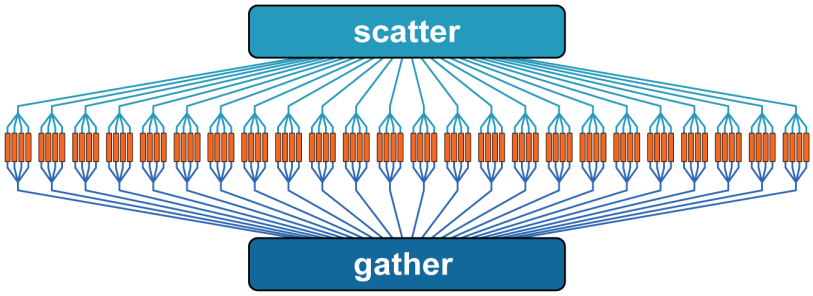

Figura 2.3: Modelo paralelo utilizado por *GATK* para el procesamiento en cluster.

2.3.4. Bowtie

Bowtie[3] 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.

2.3.5. OpenCB

OpenCB[15] es un grupo del Centro de Investigación Príncipe Felipe (CIPF) que nace como alternativa a los proyectos que normalmente se focalizan en un lenguaje de programación como el de Bioconductor o Bioperl orientados a la biología. Este grupo proporciona software avanzado y open-source para el análisis de datos de genómica con una alta productividad.

El grupo está organizado en cuatro subproyectos diferentes 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.

Capítulo 3

Asignaturas Cursadas

3.1. Generación de documentos científicos en informática

3.1.1. 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.

3.1.2. 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.

3.1.3. 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.

3.2. Introducción a la programación de arquitecturas de altas prestaciones

3.2.1. 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.

3.2.2. 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 computo 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 que 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.

3.2.3. 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 partes (en 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.

3.3. Tecnologías de red de altas prestaciones

3.3.1. 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.

3.3.2. 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.

3.3.3. 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 mas 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.

Bibliografía

[1] *Bioconductor webpage*, http://www.bioconductor.org, Accessed: 2013-05.

[2] *Bioperl webpage*, http://www.bioperl.org, Accessed: 2013-05.

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

[4] *Genome analysis toolkit webpage*, http://www.broadinstitute.org/gatk, Accessed: 2013-05.

[5] *R project for statistical computing webpage*, http://www.r-project.org, Accessed: 2013-05.

[6] OpenMP ARB, *Openmp application program interface: Version 4.0 rc2*, 2013.

[7] Javier Bueno, Judit Planas, and Alejandro Duran, *Productive programming of gpu clusters with ompss*, IEEE 26th International Parallel and Distributed Processing Symposium (2012), 557–568.

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

[9] Jeffrey Dean and Sanjay Ghemawat, *Mapreduce: Simplified data processing on large clusters*, Proceedings of the 6th Symposium on Operating System Design and Implementation (2004), 137–150.

[10] 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** (2011), no. 5.

[11] Michael J. Flynn, *Some computer organizations and their effectiveness*, IEEE Transactions on Computers **c-21** (1972), no. 9, 948–949.

[12] Ananth Grama, Anshul Gupta, George Karypis, and Vipin Kumar, *Introduction to parallel computing*, Addison Wesley, 2003.

[13] William Gropp, Ewing Lusk, and Anthony Skjellum, *Using mpi : Portable parallel programming with the message-passing interface scientific and engineering computation*, MIT Press, 1999.

[14] Norm Matloff, *Programming on parallel machines*, University of California, Davis.

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

[16] Michael Jay Quinn, *Parallel programming in c with mpi and openmp*, McGraw-Hill, 2003.

[17] W. Richarch Stevens and Stephen A. Rago, *Advanced programming in the unix environment: Second edition*, Addison Wesley Professional, 2005.

[18] Tim Stitt, *An introduction to the partitioned global address space programming model*, CNX.org (2010).

Índice de figuras

Índice de tablas