Universitat Autònoma de Barcelona

Departament d'Arquitectura de Computadors i Sistemes Operatius

Màster en Computació d'Altes Prestacions

Gestión de Recursos en nodos Multi-Core de Memoria Compartida.

Memoria del trabajo de ”Iniciación a la Investigación. Trabajo de Fin de Máster” del “Máster en Computació d'altes prestacions”, realizado por Tharso de Souza Ferreira, bajo la dirección de Juan Carlos Moure presentada en la Escuela de Ingeniería (Departamento de Arquitectura de Computadores y Sistemas Operativos)

2010

Iniciación a la investigación. Trabajo de fin de máster

Máster en Computación de Altas Prestaciones.

Título: Gestión de Recursos en nodos Multi-Core de Memoria Compartida.

Realizada por Tharso de Souza Ferreira en la Escuela de Ingeniería, en el Departamento Arquitectura de Computadores y Sistemas Operativos

Dirigida por: Juan Carlos Moure

Firmado

Director
Juan Carlos Moure

Estudiante
Tharso de S. Ferreira

Agradecimientos

A mis padres y mis hermanas, ya que sin su aliento, confianza y apoyo no sería capaz de llegar tan lejos.

A mi amigo Alexandre Strube, el primero en proporcionarme conocimientos que yo nunca seré capaz de pagar, y por todo el apoyo que me ha dado lejos de casa.

Juan Carlos, Porfi y Toni, por su paciencia en todas las reuniones, por el conocimiento proporcionado, y por estar a mi lado en cada paso.

Emilio y Lola por confiar en mi trabajo y proporcionarme oportunidades para crecer y aprender dentro de una familia, llamada CAOS.

Eduardo, Carlos, Aprígio, César, Sandra, Andrea y João por ser parte de mi familia lejos de mi casa, y proporcionar una amistad que quiero guardar por mi vida entera.

Abstract

Resource management in Multi-core processors has become more important with the evolution of applications and architectures. However, this management is very complex. For example, the same parallel application running several times with the same input data in a single multi-core node, can have variable times of execution. There are many hardware and software factors that affect performance. The way that resources (computation and memory) are distributed among the processes or threads, possibly belonging to multiple applications that compete with each other, is crucial to determine its performance. The difference between doing this distribution of resources without knowing real application requirements and the distribution with a specific purpose is increasing. The best way to do this distribution is automatically, with minimal intervention from the programmer.

It is important to emphasize that the way an application is executed on an architecture, is not necessarily the best and this situation can be improved by appropriate management of available resources. A proper management of resources can offer advantages to application developers, through a correct abstraction in the administration of these resources, as well as the computing environment where it runs, allowing a greater number of applications running with the same quantity of resources. Therefore, this resource management does not require changes in the application or in its execution.

With the intention to propose policies for resource management, the behavior of intensive applications in computation and memory was analyzed. This analysis was done through the study of parameters location between the cores, necessity of using shared memory, the size of the load input, the distribution of data within the processor and the granularity of work. Our purpose is to identify how these parameters affect the efficiency of implementation, identify bottlenecks and propose possible improvements. Another proposal is to adapt the strategy already used by the Scheduler with the intention to obtain better results.

Keywords: Resource Management, Scheduler, Cluster, Bioinformatics Applications.

Resum

La gestió de recursos en els processadors multi-core ha guanyat importància amb l'evolució de les aplicacions i arquitectures. Però aquesta gestió és molt complexa. Per exemple, una mateixa aplicació paral.lela executada múltiples vegades amb les mateixes dades d'entrada, en un únic node multi-core, pot tenir temps d'execució molt variables. Hi han múltiples factors del maquinari i programari que afecten el rendiment. La forma en què els recursos del maquinari (còmput i memòria) s'assignen als processos o threads, possiblement de diverses aplicacions que competeixen entre si, és fonamental per determinar aquest rendiment. La diferència entre fer assignació dels recursos sense conèixer la veritable necessitat de l'aplicació, amb una assignació amb un objectiu específic és cada vegada més gran. La millor manera de fer aquesta assignació és automàticament, amb una mínima intervenció del programador.

És important destacar, que la forma en que l'aplicació s'executa en una arquitectura no necessàriament és la més adequada, i aquesta situació pot millorar a través de la gestió adequada dels recursos disponibles. Una apropiada gestió de recursos pot oferir avantatges tant al desenvolupador de les aplicacions, a través d'una correcta abstracció en l'administració d'aquests recursos, així com a l'entorn informàtic on aquesta s'executa, permetent un major nombre d'aplicacions en execució amb la mateixa quantitat de recursos. Així mateix, aquesta gestió dels recursos no requeriria introduir canvis a l'aplicació, o a la seva estratègia operativa.

Per tal de proposar polítiques per a la gestió dels recursos, es va analitzar el comportament de aplicacions intensives de còmput i intensives de memòria. Aquest dejalo anàlisi es va dur a terme a través de l'estudi dels paràmetres d'ubicació entre els cores, la necessitat d'usar la memòria compartida, la mida de la càrrega d'entrada, la distribució de les dades dins del processador i la granularitat de treball. El nostre objectiu és identificar com aquests paràmetres influeixen en l'eficiència de l'execució, identificar colls d'ampolla i proposar possibles millores. Una altra proposta és adaptar les estratègies ja utilitzades pel Scheduler amb la finalitat d'obtenir millors resultats.

Paraules clau: Gestió de recursos, planificació del treball, Cluster, Aplicacions Bioinformàtica.

Resumen

La gestión de recursos en los procesadores multi-core ha ganado importancia con la evolución de las aplicaciones y arquitecturas. Pero esta gestión es muy compleja. Por ejemplo, una misma aplicación paralela ejecutada múltiples veces con los mismos datos de entrada, en un único nodo multi-core, puede tener tiempos de ejecución muy variables. Hay múltiples factores hardware y software que afectan al rendimiento. La forma en que los recursos hardware (cómputo y memoria) se asignan a los procesos o threads, posiblemente de varias aplicaciones que compiten entre sí, es fundamental para determinar este rendimiento. La diferencia entre hacer la asignación de recursos sin conocer la verdadera necesidad de la aplicación, frente a asignación con una meta específica es cada vez mayor. La mejor manera de realizar la es resolverlos automáticamente, con una mínima intervención del programador resolver el problema de la gestión de recursos.

Es importante destacar, que la forma en que la aplicación se ejecuta en una arquitectura no necesariamente es la más adecuada, y esta situación puede mejorarse a través de la gestión adecuada de los recursos disponibles. Una apropiada gestión de recursos puede ofrecer ventajas tanto al desarrollador de las aplicaciones, como al entorno informático donde ésta se ejecuta, permitiendo un mayor número de aplicaciones en ejecución con la misma cantidad de recursos. Así mismo, esta gestión de recursos no requeriría introducir cambios a la aplicación, o a su estrategia operativa.

A fin de proponer políticas para la gestión de los recursos, se analizó el comportamiento de aplicaciones intensivas de cómputo e intensivas de memoria. Este análisis se llevó a cabo a través del estudio de los parámetros de ubicación entre los cores, la necesidad de usar la memoria compartida, el tamaño de la carga de entrada, la distribución de los datos dentro del procesador y la granularidad de trabajo. Nuestro objetivo es identificar cómo estos parámetros influyen en la eficiencia de la ejecución, identificar cuellos de botella y proponer posibles mejoras. Otra propuesta es adaptar las estrategias ya utilizadas por el Scheduler con el fin de obtener mejores resultados.

Palabras clave: Gestión de recursos, planificación del trabajo, Cluster, Aplicaciones Bioinformática.

Resumo

A gestão de recursos nos processadores multi-core ganhou importância com a evolução das aplicações e arquiteturas. Porém esta gestão é muito complexa. Por exemplo, uma mesma aplicação paralela executando muitas vezes com os mesmo dados de entrada, em um único nó multi-core, pode ter tempos de execução muito variáveis. Existem muitos fatores de hardware e software que afetam este rendimento. A forma que os recursos (cômputo e memória) se distribuem entre os processos ou threads, possivelmente de várias aplicações que competem entre si, é fundamental para determinar o seu rendimento. A diferença entre fazer esta distribuição de recursos sem conhecer a verdadeira necessidade da aplicação, por uma distribuição com um objetivo específico é cada vez maior. A melhor maneira de fazer esta distribuição é automaticamente, com uma mínima intervenção do programador.

É importante destacar, que a forma em que a aplicação se executa em uma arquitetura não é necessariamente a mais adequada, e está situação pode melhorar através da gestão adequada dos recursos disponíveis. Uma apropriada gestão dos recursos pode oferecer vantagens tanto para o desenvolvedor das aplicações, através de uma correta abstração na administração destes recursos, assim como o ambiente informático onde se executa, permitindo um maior numero de aplicações em execução com a mesma quantidade de recursos. Assim mesmo, esta gestão de recursos não requer introduzir mudanças na aplicação, ou na sua forma de execução.

Com a intenção de propor políticas para a gestão de recursos, analisamos o comportamento de aplicações intensivas em cômputo e memória. Está análise foi feita através dos estudos dos parâmetros de localização entre os cores, a necessidade de uso da memória compartilhada, o tamanho da carga de entrada, a distribuição dos dados dentro do processador e a granularidade de trabalho. Nosso objetivo é identificar como estes parâmetros influem na eficiência da execução, identificar gargalos e propor possíveis melhoras. Outra proposta é adaptar a estratégia já utilizada pelo Scheduler com a intenção de obter melhores resultados.

Palavras Chave: Gestão de Recursos, Planificação de Trabalhos, Cluster, Aplicações Bioinformáticas.

Contenido

Lista de Figuras

Capítulo 1 Introducción

1.1 Descripción General

La evolución y el aumento de la capacidad de los recursos disponibles siempre han ido junto con la necesidad de mejoras computacionales. Cuando se piensa en mejorar el cómputo de un ordenador, el primer punto que se evalúa es aumentar la capacidad del hardware, suponiendo que es el principal medio de mejora. Debido a las limitaciones en el hardware, la paralelización de la aplicación es una solución muy atractiva, ya que la idea de realizar más operaciones en el mismo instante de tiempo compensa la mejora de hardware. El crecimiento progresivo del rendimiento de los procesadores en los últimos años, hoy capaz de producir los procesadores multi-core, explora la capacidad en las aplicaciones paralelas.

Buscando una capacidad de cómputo que está por encima de los límites impuestos por las computadoras convencionales, los clusters se colocaron como una opción para satisfacer esta necesidad de procesamiento, más concretamente los clusters de alto rendimiento, pero sigue siendo una opción con un alto costo. Una opción en comparación con la compra de un cluster de alto rendimiento, que promete explorar la capacidad de cómputo, es el uso de clusters del tipo COTS(Commodity Off-The-Shelf): un entorno informático construido con los equipos comunes, proporcionando un buen rendimiento. En cuanto a la forma de uso, podemos definir dos tipos de clusters, llamado Beowulf o cluster dedicado, cuando se construye para un procesamiento controlado, para un tipo particular de uso. En contrapunto a esta opción están los clusters no dedicados, que no tiene carga controlada por un sistema, y hacen uso de los recursos disponibles para llevar a cabo el procesamiento en paralelo [**1**].

Por su menor costo, los clusters no dedicados se distribuyen en muchos centros de investigación, pequeñas y medianas empresas a través de la mayor disponibilidad de recursos. Cuando el objetivo es paralelizar, es una buena idea conseguir más capacidad de cálculo de costo mínimo, un punto muy atractivo de las redes de estaciones de trabajo (Network of Workstations, NOW) [**2**]. Sin embargo, su uso no se explota al máximo, hay muchos recursos que quedan libres la mayoría del tiempo.

Aplicaciones por la forma de manipular sus datos, tiene los recursos la gran mayoría del tiempo libre, o está usando una cierta cantidad de un recursos específico, mientras que otra parte está la mayor parte de su tiempo libre [**1**]. En el caso de clusters no dedicados, algunas aplicaciones son de especial importancia: las aplicaciones locales, que deben tener una mayor prioridad en este contexto. Un usuario puede estar ejecutando un vídeo que se almacena en el disco local, que se refleja en el uso de CPU, en determinados períodos, sin embargo, hace un

mayor uso de la memoria de otras aplicaciones, por ejemplo, un compilador que hace uso intensivo de CPU durante su ejecución. La pregunta entonces es cómo proceder cuando se desea utilizar un determinado programa en un entorno paralelo y carece de los recursos necesarios. En muchos entornos informáticos, el planificador de trabajos está configurado detener las aplicaciones hasta que el sistema tenga todos los recursos necesarios. Dependiendo de la aplicación y la cantidad de recursos que necesita, la aplicación puede quedar en un estado de starvation. En esta situación, son importantes las políticas que se figuran en el Scheduler. Para este caso, lo que hace el sistema operativo, es aumentar la prioridad del proceso que está esperando demasiado tiempo en la cola, hasta que comience su ejecución. Hay muchas lagunas que se abren en tal situación, de acuerdo con las políticas que tiene programado el Scheduler. De acuerdo con el tipo de aplicación que está pendiente de ejecución, la decisión podría ser partir los datos para que se inicie su ejecución. Sin tener que esperar a todos los recursos necesarios, lo cierto es que habrá un aumento en el tiempo de ejecución total por no disponer de los recursos en exclusivo, pero evita una larga espera en las colas del sistema.

Es importante destacar que utilizar el tiempo de inactividad de las computadoras que pertenecen a NOWs para ejecutar aplicaciones paralelas ya no es una fantasía. Una serie de trabajos ya se ocupan de este tema, que muestran diferentes formas de utilizar la capacidad de cómputo disponible [**3**] , [**4**], [**5**].

La evolución de los procesadores y la entrada de arquitecturas multi-procesador, multi-core y multi-thread, añadiendo potencia de cómputo y aumentando el paralelismo, también abrió más espacio para la planificación de trabajos en clústers. Esto permite, además de la planificación de las tareas en el cluster, realizar la planificación de las tareas en multi-cores. La complejidad de tomar la decisión de cómo distribuir la carga dentro del nodo multi-core aumenta con cada cambio de la arquitectura. En la Figura 1 se muestra un procesador de la arquitectura multi-processador.

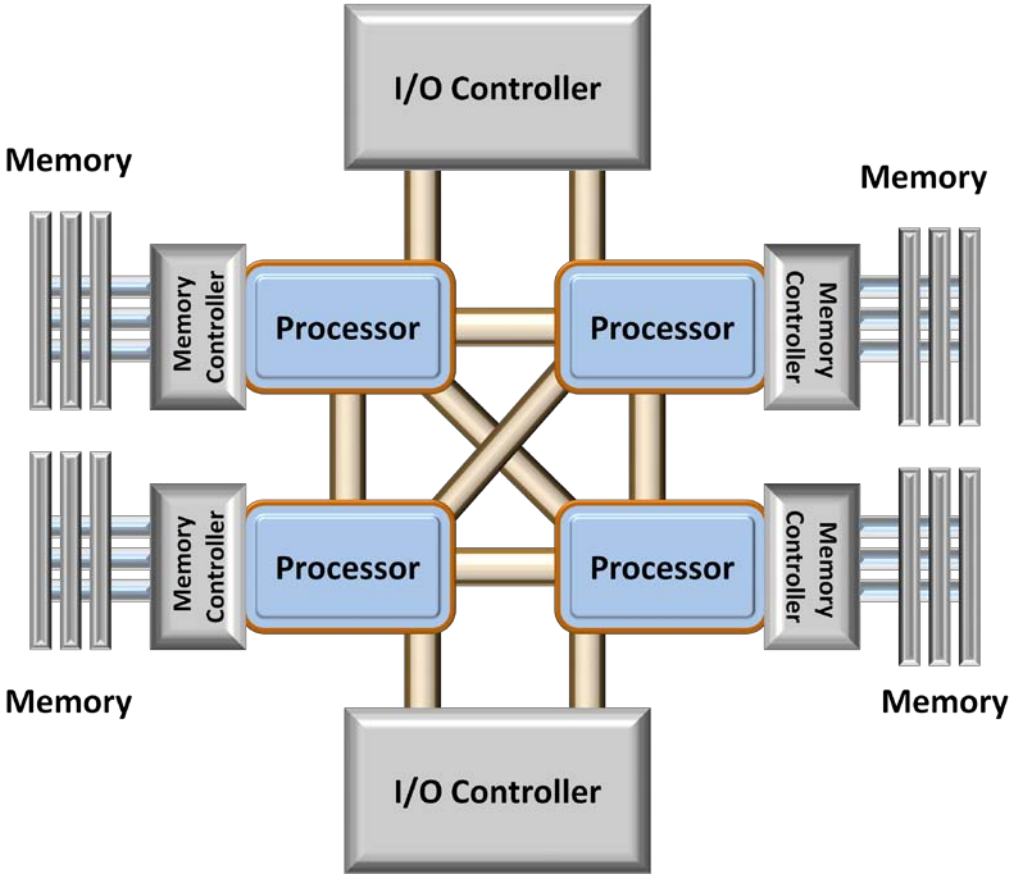

Figura 1 – Arquitectura del Multi-processador.

La decisión de cómo distribuir las cargas de trabajo, ya no es una decisión simple, porque además de trabajar en una arquitectura que hace uso de más de un procesador, la tecnología multi-core y multi-threaded, como se muestra en la Figura 2, aumenta el nivel de complejidad de dicha decisión.

Una buena planificación de la carga de trabajo, se convierte en fundamental para el principio de hacer un buen uso de los recursos disponibles, y cómo distribuirlo. En [**6**] se estudia, las ventajas y desventajas de la forma de distribuir las cargas de trabajo en chips multi-core.

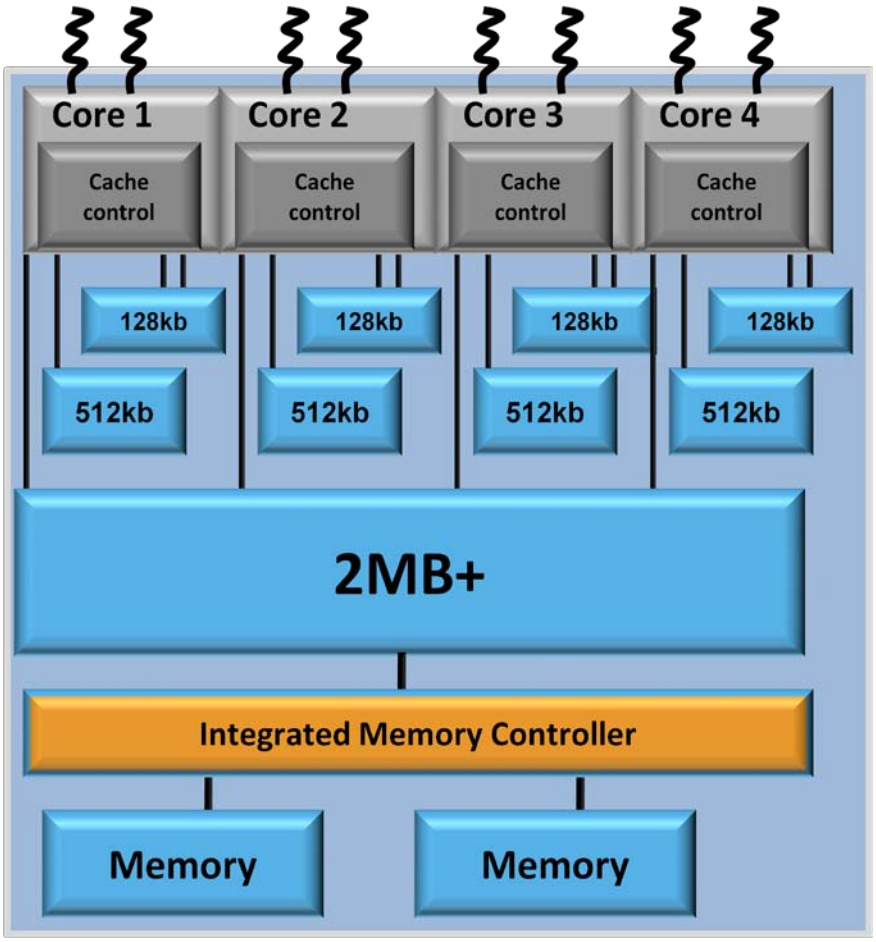

Figura 2 – Arquitectura del Multi-core.

Acercarse a estos conceptos para la planificación y gestión de los recursos, las aplicaciones científicas que están disponibles en el mercado. Agregando conceptos de la High Performance Computing a análisis de rendimiento de las aplicaciones de bioinformática como BWA (Burrows-Wheeler Align) [**7**], [**8**] .

El objetivo que se persigue con aplicaciones como BWA, es el alineamiento de secuencias genéticas, que se relaciona con el estudio de los seres vivos en general. El ser humano tiene un código genético, compuesto por cadenas largas, llamadas ADN (***D****eoxyribo****n****ucleic* ***A****cid),* que son cadenas de moléculas de azúcar, que están conectadas por compuestos como la Adenina, Timina, Guanina y Citosina, agrupados en parejas. Estos compuestos están representados por cadenas, mediante las letras A, T, G y C. El código del ADN de una persona ejemplificada en formato FASTA, que es utilizado como el archivo de entrada en BWA, sería algo como en la figura 3.

```
>Genoma_humano
GGCCCCCCACCTCCCCCCGTGGCAGCTCCAACCCCAGCTTTTTCACTAGT
AAGGCAGTCGGGCCCCTGGGCCACGCCCACTCCCCCAAGCGGGGAAGGAG
CTTCGCGCTGCCGCTTGGCTGGGGACTGGGCACCGCCCTCCCGCGGCTCC
TGAGCCGGCTGCCACCAGGGGTGCGCGCCAGCGGTGTCCGGGAGCCTAGC
GGCGCGTGTGCAGCGGCCAGTGCACCTGCTCTGGCCCTCGCCGCGGTCTC
TGCCAGGACCCCGACGCCCAGCCTGACCCTGCCATTCAGCGGGGCTGCGG
CTCCACGGCCTGCGACAGCAGCCCCACCTGGCATTCAGCGCGCTCCCGGG
GGCAGAGGTCGCGGTGTCCTCACGCTGTGGTGCCGGCCTACAACCCCCAC
GCCGGGCTCGGGCCCGGCGGAGGAGGGCGATGCTCCCCGGGTAGGACAAA
CCGGTCACCTGGGCTGCGAGGGCGGCTTAGGGGCAGAAGCGGCGGTCCAG
GGCCGCCTGGCGCAGCAGCCTGTCCCAGCCGCGGTCCCTGCAGTCCCTCC
```

Figura 3 – Genoma Humano en formato FASTA.

En contrapunto BWA hace uso de un segundo archivo de entrada, este caso dos reads de extracción, que son pequeñas muestras extraídas de un organismo vivo, y si se debe buscar en la secuencia de referencia, en este caso en el genoma. La figura 4 se ilustra un de archivo de entrada utilizado por BWA en formato FASTQ con una serie de pequeños reads de extracción.

read_1.fq

```
@B_TITR_1_1_332_588
GATTCGAGAGACGCAGACGCACGAGGAGAA
+
<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
@B_TITR_1_1_668_35
ATATCGGATGACACAATATGGGAGGTTGAC
+
<<<<<<<<<<<<<<<<<<<<<<<<<<<<'
@B_TITR_1_2_843_403
TGTAGCTTTTCATGACAATTTTATAGGTGT
+
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;9
@B_TITR_1_3_618_922
TCCAAACTCTTCATAATTCCATTTATAAAC
+
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
@B_TITR_1_3_99_934
TCTAGGTAAATTACACGGTCTTTTCCTGAA
+
;;;;;;;;;;;;;;;;;;;;;;;;;/;;;1
@B_TITR_1_3_104_863
TCGTGTCAACATTCTTTCATTAACATTATT
+
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
```

Figura 4 – Reads de extracción en formato FASTQ.

El punto clave del alineamiento de secuencias, no es más que encontrar el lugar de una pequeña secuencia extraída de un organismo vivo dentro de una secuencia larga de referencia.

Dependiendo de la forma de trabajar, BWA representa un tipo de aplicación que hace uso intensivo de la CPU y la memoria, muy importantes para los objetivos de este proyecto de investigación. En [**9**], se observa que la mejor aplicación del suffix tree para la indexación de un genoma humano, se traduce en 50 GB de uso de memoria, muy por encima de la capacidad disponible en los ordenadores de hoy en día. En [**7**], dice que son necesarios, 12 GB de memoria para construir el suffix tree del genoma humano. Pero [**10**] presenta un algoritmo basado en el BWT, que requiere menos de 1 GB de memoria para crear la estructura del genoma humano durante la indexación.

Como parte de la propuesta para resolver el problema del uso de los recursos libres en clusters no dedicados, una opción es el desarrollo de políticas para el Scheduler, para las arquitecturas multi-core, de modo que el usuario puede ejecutar aplicaciones locales como si estuviera solo. El sistema debe realizar la gestión dinámicas de las cargas en paralelo, buscando el mejor rendimiento posible. Una alternativa diferente para el manejo de este tipo de problema, propone el uso del paradigma de MapReduce, que administraría los recursos de forma automática, el sistema determin la parte compleja del problema, decidir cómo distribuir y equilibrar la carga, haciendo la distribución y la gestión datos, la sincronización y comunicación, además del manejo de la tolerancia a fallos.

1.2 Objetivo

En los últimos años, con la creciente necesidad de cómputo, en especial de las aplicaciones con fuertes requerimientos de accesos a datos, ha surgido la necesidad de que el Scheduler sea capaz de tomar decisiones de manera transparente sin intervención del usuario. Dos puntos que refuerzan esta necesidad, son el uso intensivo de los clusters no dedicados en centros de investigación, junto con la potencia de las arquitecturas multi-cores y multi-thread, que aumentó el nivel de complejidad. La pregunta entonces sería como poner diferentes cargas de trabajo que se ejecuten en un entorno computacional, de modo que todas las aplicaciones hagan uso eficiente de los recursos. Además de no afectar a otras aplicaciones en el mismo entorno.

El objetivo general de esta investigación es analizar el problema de la planificación de los nodos de cómputo multi-thread, multi-core y multi-procesador de memoria compartida. Se

deben identificar las características de las aplicaciones con paralelismo de datos, que afectan al uso de los recursos de cómputo, memoria y disco.

Para alcanzar estos objetivos, se define el estudio computacional de una aplicación bioinformática, de alineamento de secuencias genéticas. Utiliza la aplicación un paradigma de paralelismo de tipo SPMD (Single Process, Multiple Data), y realiza un uso intensivo de cómputo y memoria. Se mide la eficiencia de la aplicación con limitaciones de recursos diversos, compitiendo con la carga de trabajo local. Finalmente se buscan posibles cuellos de botella y se identifican oportunidades de mejora en la política de gestión de los recursos. La idea es ser capaz de diseñar políticas, automáticas de gestión de recursos que faciliten la tarea al programador no especializado de arquitecturas multi-core.

1.3 Organización del Trabajo

En este capítulo han introducido el contexto en general que caracteriza nuestro proyecto de investigación, y hasta dónde queremos seguir. Los capítulos siguientes están dedicados a detallar los puntos de nuestro trabajo, organizados de la siguiente manera:

Capítulo 2.Planificación. Se muestra en este capítulo como son en la actualidad las políticas del Scheduler así como el trabajo actual que se desarrolla. También se muestra la importancia para nuestro proyecto de investigación y los objetivos que queremos conseguir.

Capítulo 3.MapReduce. Muestra la importancia de MapReduce a los objetivos de este proyecto de investigación, así como el estado actual de este paradigma. Como el uso de este paradigma es importante para la política de planificación de tareas.

Capítulo 4. Aplicaciones Bioinformáticas. Presenta la herramienta Bioinformática que utilizamos para contextualizar este trabajo de investigación y su importancia de modo que podemos alcanzar los objetivos establecidos.

Capítulo 5. Experimentación. En este capítulo se detallan todas las evaluaciones que hemos hecho, así como las medidas obtenidas con respecto a BWA (Burrows-Wheeler Alineador) como aplicación principal, y también a una aplicación NAS Parallel Benchmakr, con limitaciones de recursos.

Capítulo 6. Conclusiones y trabajos futuros. Se concluye, presentando los problemas que quedan como líneas abiertas, señalando los caminos a seguir como trabajo futuro.

Capítulo 2 Planificación

2.1 Introducción

En la actualidad, los sistemas de tiempo compartido, tienen una ejecución aparentemente simultánea, donde muchos procesos están cambiando entre el estado de ejecución y el estado de espera en un tiempo muy corto. Los algoritmos de los sistemas operativos actuales deben resolver varios puntos de conflicto, tales como:

- Tiempo de respuesta adecuada;
- Buen rendimiento para trabajar en segundo plano;
- Evitar el starvation de los procesos;
- Conciliar las necesidades de los procesos de alta y baja prioridad.

Para combinar los factores de buen rendimiento, así como evitar los problemas mencionados, se utiliza un conjunto de reglas para determinar el nuevo proceso a ejecutar, a las que se denominan política de planificación. Los planificadores de hoy usan la técnica de tiempo compartido (time sharing), donde varios procesos se ejecutan en la misma serie de tiempo, ya que el tiempo de CPU se divide (time slice). Cada espacio de esta serie de tiempo de CPU disponible se establece para la ejecución de un proceso. En general, un procesador puede ejecutar cualquier proceso en un instante, si el proceso actual no termina cuando expira su time slice, se produce un cambio de contexto.

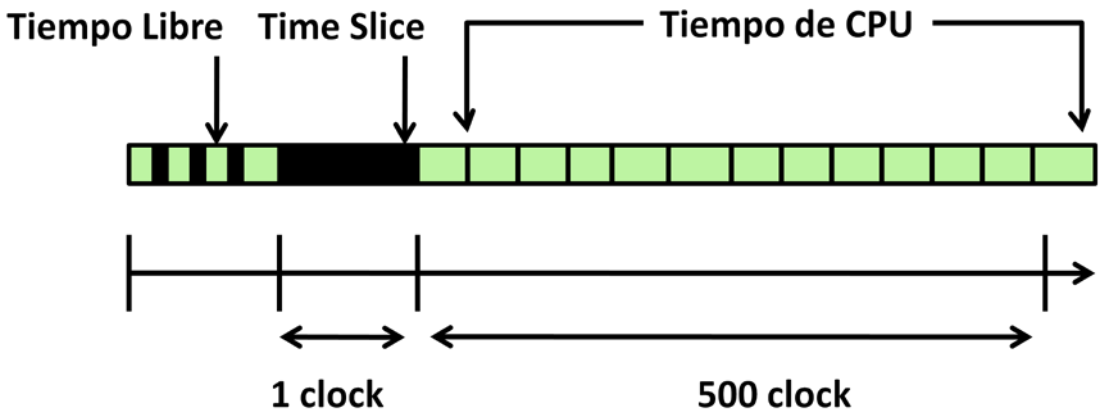

Figura 5 – Time Slice.

La política de planificación también se basa en una organización de los procesos de acuerdo con su prioridad. A veces utiliza algoritmos complejos para determinar la prioridad actual de un proceso. Cada proceso se asocia con un valor, que le dice al planificador lo importante que es para ejecutar el proceso en la CPU.

En general, el planificador hace la supervisión del proceso de forma dinámica. Comprueba qué procesos se están ejecutando y ajusta periódicamente su prioridad. De esta manera, los procesos que tenían el uso de la CPU durante mucho tiempo, ven reducidas sus prioridades dinámicas. Por el contrario, si un proceso pasa demasiado tiempo sin hacer uso de los recursos, se premia y gana prioridad en el sistema.

Dentro del contexto de la planificación, se destacan dos tipos de procesos, clasificados como limitados por entrada y salida (I/O-bound) y por CPU (CPU-bound). Los primeros tienen un fuerte uso de los dispositivos de I/O, y pasan la mayor parte de su tiempo esperando las operaciones de entrada y de salida. Los procesos clasificados como CPU-bound hacen uso intensivo de operaciones de cómputo, lo que requiere un gran tiempo de uso de CPU. Dentro de esta clasificación podemos dividir los casos entre:

- Procesos batch: No requieren la interacción del usuario, pero la mayoría de las veces se ejecutan en background. No necesitan tener un tiempo de retorno muy rápido, en general son los compiladores de lenguaje o los programas de computación científica.

- Procesos Real-time: Necesitan requisitos estrictos de recursos. En general nunca deberían ser bloqueado por procesos de menor prioridad, y su tiempo de respuesta debería acotado. Son aplicaciones de audio y vídeo, o recibir datos de sensores físicos.

- Procesos Interactivos: Interactúan constantemente con el usuario, pasan mucho tiempo esperando la pulsación de botones o las operaciones del mouse. Cuando se recibe una entrada, el proceso debe ser despertado rápidamente. Los programas interactivos son, por ejemplo, editores de texto o aplicaciones de gráficos.

Algunos procesos batch pueden ser clasificados como I/O o CPU, según su aplicación. Mientras que los programas en tiempo real son reconocida explícitamente por el planificador. Para decidir qué proceso debe ser considerado, el planificador utiliza una heurística basada en el

comportamiento de cada proceso para decidir como los procesos deben clasificarse. En general, el planificador debe favorecer los procesos interactivos en lugar de los procesos batch.

Considere la siguiente situación, donde se ejecutan un editor de texto y un compilador. El editor de texto se considera un programa interactivo, entonces debe tener una prioridad más alta que un compilador. Pero a menudo es suspendido porque espera el ingreso de datos del usuario entre cada interrupción. Así que cada vez que el usuario presiona una tecla, se hace una interrupción de llamadas, y el proceso editor de texto es despertado. Además de determinar que la prioridad dinámica del proceso editor de textos, es más grande que la prioridad en actual del proceso en ejecución. Así pues, la necesidad de que cada interacción, el planificador realiza un cambio de contexto. Como resultado, la aplicación del editor de texto reiniciar rápidamente y la interacción del usuario, en este caso la tecla presionada se muestra en la pantalla del ordenador. Cuando una interacción dada es procesada, el proceso se suspendió de nuevo y el proceso compilador puede volver a ejecutar.

Antiguos algoritmos para el planificador, eran muy simples, con cada cambio, la lista del proceso tenía que ser revisada, calculada las prioridades y seleccionado el mejor proceso para ejecutar. Los algoritmos actuales, más sofisticado, seleccionan el mejor proceso para ejecutar de forma independiente la cantidad de procesos existentes. Cada CPU tiene su propia cola de procesos, además de ser capaz de distinguir entre los procesos interactivos y batch.

2.2 Time Sharing

Time Sharing se refiere al término usado para definir el tiempo de inactividad entre los procesos, que pueden ser compartidos con otros procesos para agilizar el sistema. La aplicación de tiempo compartido trae la sensación de que muchas tareas se ejecutan simultáneamente, pero la CPU ejecuta e cada tarea por un tiempo determinado.

Los tiempos son equilibrados de forma que la ejecución de cada tarea no causa una degradación en las otras tareas que comparten el tiempo .La aplicación de este procedimiento es importante ya que en los clusters no dedicados, es muy raro que sólo un usuario puede realizar mantener completamente ocupados los recursos de cpu, memoria, disco y red. Entendiendo que estos equipos permanecen mucho tiempo de inactividad, o en espera de acciones por parte del usuario, la idea es compartir estos recursos inactivos para servir a otros usuarios.

El uso de tiempo compartido tomó gran fuerza a la conclusión de que, debido al patrón de la interacción del usuario. Cuando se haga uso de una serie de recursos seguida de una larga pausa de inactividad. Muchos usuarios trabajando al mismo tiempo permiten que el tiempo libre

se pueda utilizar en otros tipos de ejecuciones, haciendo uso eficiente de los recursos. Al mismo tiempo, también podría permitir a los usuarios a otras el uso de ciertos recursos específicos, como las regiones de memoria o conexiones de entrada y de salida.

2.3 Processadores Multi-core

Los procesadores multi-core se han convertido en una realidad, algunos estudios muestran la importancia y la ventaja en la planificación del trabajo dentro de esta arquitectura [**11**], [**12**] [**13**], [**14**]. En [**11**] se destacó el problema en las limitaciones del hardware, y algunos sistemas operativos son el contrapunto de esta situación. Se muestra la necesidad de un estudio para desarrollar aplicaciones capaces de utilizar adecuadamente el paralelismo que el hardware puede proporcionar. Que el Scheduler del sistema operativo se vuelve más hábil en hacer el uso de la capacidad de los que los procesadores multi-core pueden ofrecer. Esta manera de lograr esta evolución es a través de procesos de hacer mejor clasificación de los procesos, tener la capacidad de adaptarse dinámicamente y detectar la necesidad de cooperación entre los procesos.

La planificación de aplicaciones paralelas en arquitecturas multi-core, es un tema que muchas líneas de investigación, como el uso de la memoria compartida representa una evolución, también puede representar un cuello de botella para la planificación de tareas.

En [**14**] se muestra la posibilidad de trabajar con procesos diferentes en distintos cores. Hacer planificación de tareas, de modo que la arquitectura multi-core puede utilizar de manera productiva las tareas independientes. Haciendo una relación, [**6**] muestra que las tareas que se ejecutan en diferentes cores en una misma zona de memoria, compiten por el uso de los recursos. Situación que puede resultar en una degradación del rendimiento de los procesos como se muestra en la Figura 6. Este resultado muestra la necesidad de un estudio con la intención de minimizar la pérdida de rendimiento. En la figura 7, [**6**] muestra un ejemplo arquitectura Quad-Core Xeon, donde lo autor envía cuatro tareas diferentes para cada núcleo como una manera de verificar la degradación de las ejecuciones en el peor y mejor de los casos.

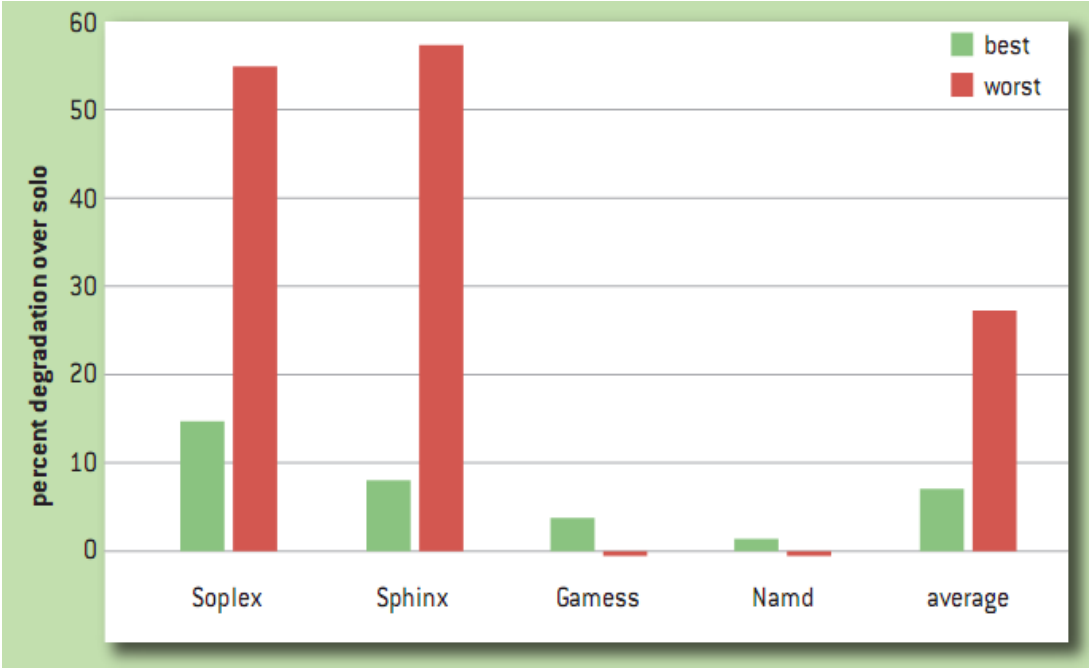

Figura 6 – La degradación del rendimiento.

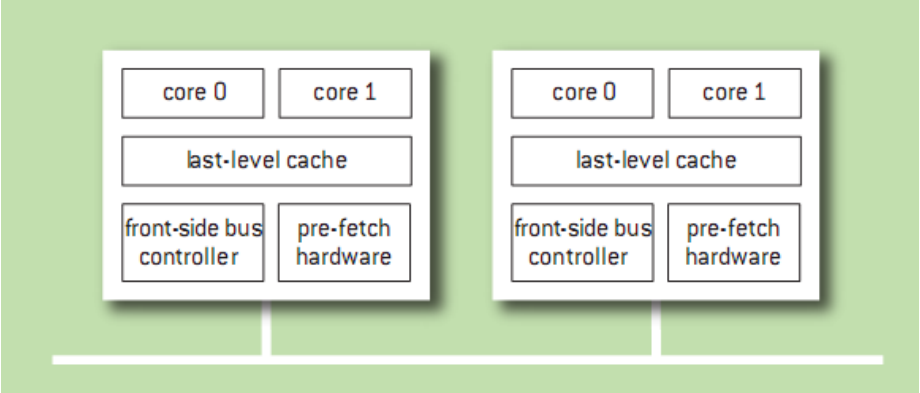

Figura 7 – Esquema de un sistema multicore con dos dominios de memoria.

En [**6**] también se muestra que no sólo la competencia por la memoria compartida es el único punto de contención por los recursos, que provocan la degradación en los procesos. Factores tales como el front-side bus, prefetching de los recursos, y el controlador de memoria, puede causar la degradación en la arquitectura del multi-core.

Además de la dificultad de la planificación de tareas, [**15**] ilustra las dificultades encontradas por los programadores para ejecutar aplicaciones en arquitecturas multiprocesador

y multi-core, teniendo en cuenta que los gestores de recursos actuales empiezan ahora, a tener en cuenta la importancia de considerar los cores, a la hora de su planificación. No les importa la localidad de la memoria entre los cores, los recursos compartidos y otras características de la arquitectura. Como solución a este problema se presenta el uso de SLURM (Simple Linux Utility for Resource Management), que permite realizar la gestión de los recursos teniendo en cuenta la arquitectura H/W del entorno.

Algunos estudios que se dirigen a la planificación de tareas de real-time, por ejemplo, [**13**] y aplicaciones paralelas, tales como [**12**], que usan arquitectura multi-core intentan tomar ventaja de algunos procesadores. En [**13**], la idea sería la necesidad de descorazonar algunas tareas a no se co-planificar, para evitar genera un tráfico significativo a la memoria compartida. El enfoque de esta situación sería la de combinar las tareas que hacen uso del tráfico para la memoria compartida en grupos, y en tiempo de ejecución utilizando la política del Scheduler para reducir la competencia entre los grupos.

La influencia del ancho de banda en planificar aplicaciones real-time y Soft real-time se estudia en [**16**]. En general, el uso de memoria es considerablemente más lento que el uso de procesadores, una situación resuelta por el uso de la memoria caché. La memoria principal y el procesador están conectados por un sencillo bus de memoria y todos los procesadores en el mismo bus. En esta situación, hacer una planificación del bus de memoria es potencialmente importante, garantizar el ancho de banda y baja latencia para aplicaciones.

Capítulo 3 MapReduce

3.1 Introducción

MapReduce es un modelo de programación y framework, desarrollado por Google (2006) para dar soporte a grandes conjuntos de datos y de cómputo en paralelo en cluster de ordenadores. El paradigma MapReduce supone que no todos los datos residen en un lugar y que funcionen todos los datos de forma sincrónica, en cambio, los datos deben trabajar de forma independiente. Los principales conceptos de MapReduce son; la función de Map, que es el proceso de asignación de la solicitud original, o la función que maneja las claves y valores de dos en dos, y el Reduce, la función que es el proceso de agregación de los resultados a un resultado consolidado, es decir, combinar todos esos valores a una clave intermedia [**17**].

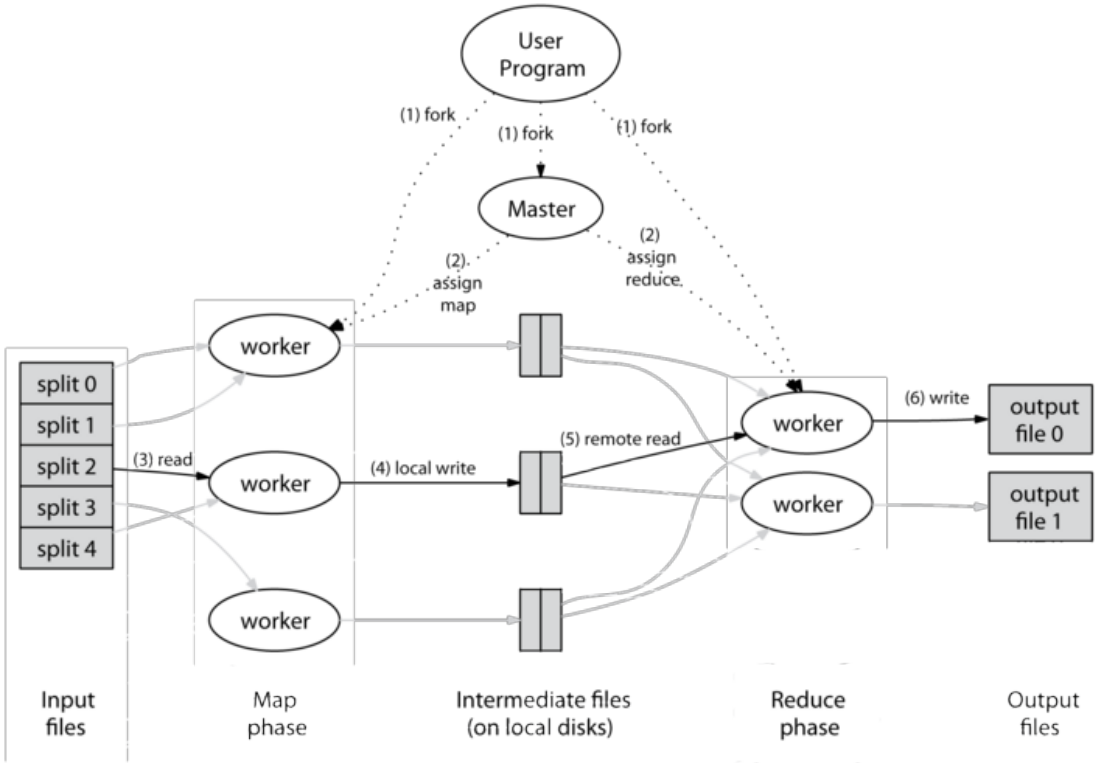

Figura 8 – MapReduce ejecución [17**].**

El funcionamiento de MapReduce tiene como punto principal la función MAP, que distribuye la solicitud en varios equipos mediante la partición del archivo de entrada en un número de bloques M partes, por lo que la entrada puede ser procesada de forma simultánea en

diferentes lugares. La función Reducir se distribuye mediante la separación a través de la clave intermediaria en N partes, usando la función de división. El número de particiones suele ser especificado por el usuario [**17**]. La figura 8 muestra los componentes básicos del paradigma, con las siguientes acciones:

1. La Librería MapReduce primero divide los archivos en M bloques, y crea copias de los ficheiros de datos en los equipos del clúster.

2. Una copia de este programa es la más importante porque está establecido como el Master, mientras que las demás copias, se definen como Workers y recibirá los trabajos del Master. El Máster busca por los Workers que están en espera, y les asigna una tarea MAP o una tarea REDUCE.

3. Los Workers que reciben una tarea de MAP, analiza, los pares de clave y de valores del archivo de entrada para enviar la información a la función MAP definida por el usuario. La información generada por la función de intermediario MAP se almacena en la memoria.

4. A menudo las parejas se escriben en el disco local, dividido en N sitios mediante la función de división. La ubicación de las parejas que estaban en la memoria y se registraron en el disco local se envía de nuevo al Master, porque él es responsable de administrar estos sitios para Reduce de los Workers.

5. Cuando un Workers con REDUCE ha leído todos los datos intermedios, los clasifica de manera que las mismas claves se agrupan. Esta organización es importante porque en general hay muchos MAP para lo mismo REDUCE, y si la cantidad de datos es demasiado grande, puede no encajar en la memoria.

6. Los REDUCE hacen el trabajo sobre los datos intermedios, basados en una única clave, que envían esta información a cada clave correspondiente al conjunto de valores enviados. La salida del REDUCE está conectada a una salida final para esta partición del REDUCE.

7. Cuando todas las tareas MAP y REDUCE se han completado, el master ejecuta el programa usuario de nuevo, en este punto por las llamadas MapReduce volver al código de usuario.

Los programas escritos con las características de MapReduce, tienen la característica de ser ejecutados en paralelo en clusters de ordenadores. Para el programador el principal beneficio de MapReduce es facilidad de crear aplicaciones paralelas, de modo que el programador sólo tiene dos funciones MAP y REDUCE. Para la planificación del trabajo MapReduce tiene las siguientes ventajas:

- El sistema que determina la parte compleja del problema;
- Gestiona los recursos de forma automática;
- Hace la partición de los datos;
- Hace la distribución y equilibrio de carga;
- Distribuye y hace la gestión de Datos;
- Sincroniza y hace la comunicación de datos;
- Hace la tolerancia a fallos.

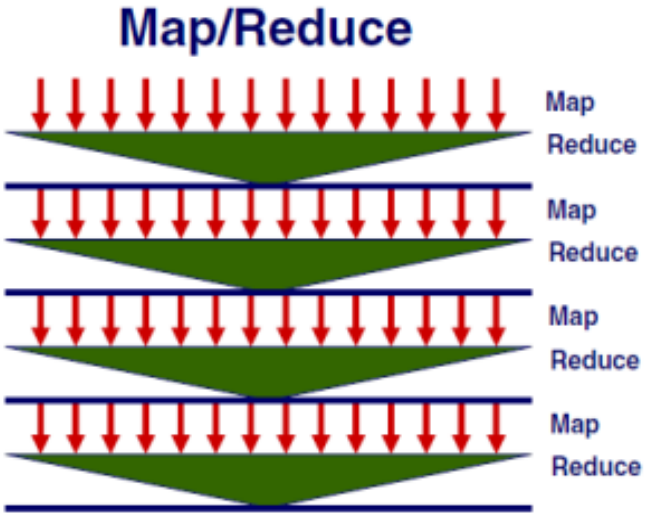

Figura 9 – MapReduce.

Distintas aplicaciones de MapReduce son posibles, todo depende del tipo de entorno. Basado en el modelo de Google MapReduce, Hadoop fue desarrollado en 2007. Tiene el mismo contexto que el MapReduce original, pero con una licencia libre, también dirigió su uso para los

clusters de ordenadores. Para nodos con varios procesadores, algunas aplicaciones ya son conocidas:

- Phoenix: Desarrollado en 2008, es una implementación de MapReduce para memoria compartida, se puede utilizar para la programación de chips multi-core, multi- procesadores de memoria compartida;

- Mars: Es un framework de MapReduce para los procesadores gráficos (GPU), desarrollado en 2008;

- Metis: Una biblioteca MapReduce desarrollado, en 2010, que permite buen desempeño para la mayoría de las cargas de trabajo.

3.2 MapReduce para Multi-core

Actualmente, los chips multi-core se han convertido en mucho más que un procesador, es cada vez más importante explorar todo está capacidad. Las actuales técnicas de programación paralela, hacen uso principalmente de envío de mensajes y memoria compartida. Sin embargo estas técnicas son bastante laboriosa para la mayoría de los desarrolladores, que exige la gestión de las cuestiones de competencia, como la sincronización de los threads.

Phoenix, que se convirtió en el tipo de implementación, que facilita la programación paralela para los chips multi-core. Hace uso de threads para la distribución de tareas paralelas MAP y REDUCE, con la memoria compartida como fuente de la comunicación, evitando la copia de datos. El Scheduler dinámicamente evalúa los procesadores disponibles y hace la distribución y balanceo de carga, maximizando la utilización de los recursos entre las tareas [**18**].

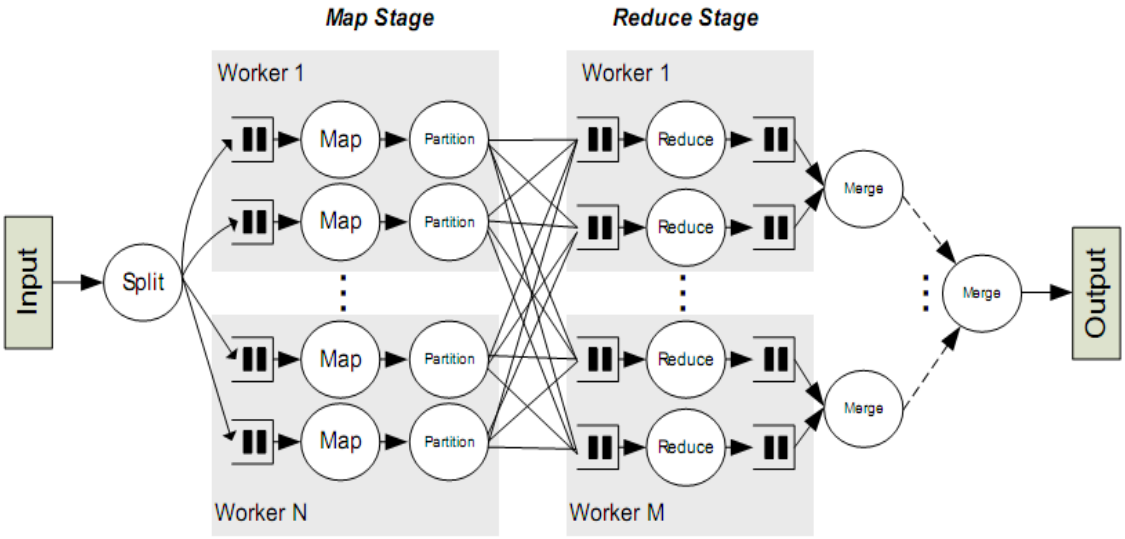

Figura 10 – Flujo de datos para el tiempo de ejecución de Phoenix [18**].**

La figura 10 muestra un flujo de datos básicos sobre el funcionamiento de la ejecución de Phoenix con MapReduce.

Capítulo 4 Aplicaciones Bioinformáticas

4.1 Introducción

Con el crecimiento de los conocimientos sobre todas las formas de vida, se observa también el crecimiento de la cantidad de datos almacenados, entre los que destacan los datos de los proyectos genómicos. Las bases de datos genómicas, aumenta la búsqueda de soluciones para manejar todos esos datos de manera eficiente.

Las bases de datos genómicas han aumentado por los miles de investigadores. Con esta gran cantidad de datos, los investigadores tienen un punto importante, para realizar búsquedas de forma rápida y con calidad, en una base que crece exponencialmente. En una computadora normal, una búsqueda con calidad puede tomar horas, además de consumir recursos más rápido. Para el proceso de búsqueda dentro de las secuencias genómicas se aplican métodos conocidos como la alineación de las secuencias. Hecho por aplicaciones específicas, como BWA (Burrows-Wheeler Alineador), es posible medir el grado de similitud entre dos secuencias. Saber comparar una secuencia del genoma de referencia con un pequeña secuencia extraída de uno organismo cualquier, y saber hasta donde ambos tienen puntos en común.

La primera generación de métodos para trabajar con alineamiento de secuencias, basado en tablas de hash, estaba bien desarrollada, había precisión y fue lo suficientemente rápido como para alinear a corto reads de extracción. Sin embargo estos métodos no eran todavía capaces de realizar un alineamiento con los espacios vacíos para una única entrada. Fueron inadecuados para realizar una alineación en una secuencia que poseía indel, que es la inserción o deleción de una lectura en particular, lo que podría considerarse una mutación [**7**].

4.2 Alineamiento de Secuencias

4.2.1 Burrows-Wheeler Transform

La idea detrás del método de BWT(Burrows-Wheeler Transform) es una permutación reversible de los elementos de un texto. Desarrollado inicialmente en el contexto de compresión de datos, indexación con BWT permite realizar una búsqueda eficaz en el texto de gran tamaño con un uso de poca memoria [**19**], [**20**] .

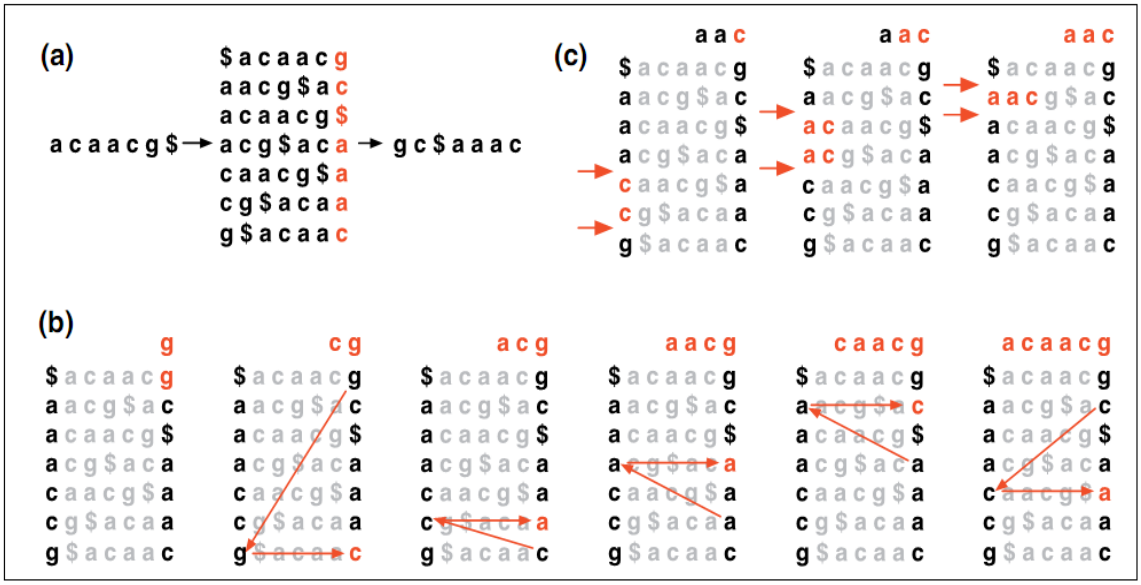

Figura 11 – Burrows Wheeler Transform.

Los pasos para BWT organizar los dados son:

(A) La matriz de Burrows-Wheeler se transforma en 'acaacg';

(B) Las medidas adoptadas para identificar las líneas y el sufijo referencia al prefijo 'aac';

(C) Reverso de permutación, aplicada repetidamente para el último levantamiento para recuperar el texto original BWT.

4.2.2 Burrows-Wheeler Aligner

Otros estudios, han implementado una lectura de secuencia utilizando tablas hash, pero estos programas son muy flexibles con la memoria. Estos programas pueden ser fácilmente paralelizados con multi-thread, pero el uso grande de memoria para la construcción del índice del genoma humano es el principal problema. Por otra parte, la estrategia utilizada por estos programas puede hacer su velocidad sensible a una tasa de error [7].

Buscando superar las dificultades, se ha desarrollado BWA, una aplicación eficiente que realiza alineamiento de secuencias. Haciendo uso de reads de extracción contra una secuencia de referencia larga, contemplando espacios vacíos y los desacuerdos. Es capaz de realizar los alineamientos 10-20 veces más rápido que los métodos tradicionales y con la misma precisión que ya se conoce. Tiene un formato de salida de SAM, que es una forma sencilla de analizar lo

alineamiento, compatible con la mayoría de aplicaciones que realiza alineamiento de secuencias. BWA se basa en la búsqueda inversa utilizando BWT (Burrows-Wheeler Transform), que utiliza una estructura de suffix tree para indexar la secuencia de referencia, y un Suffix Array para comprimir los datos [**7**], [**21**], [**22**].

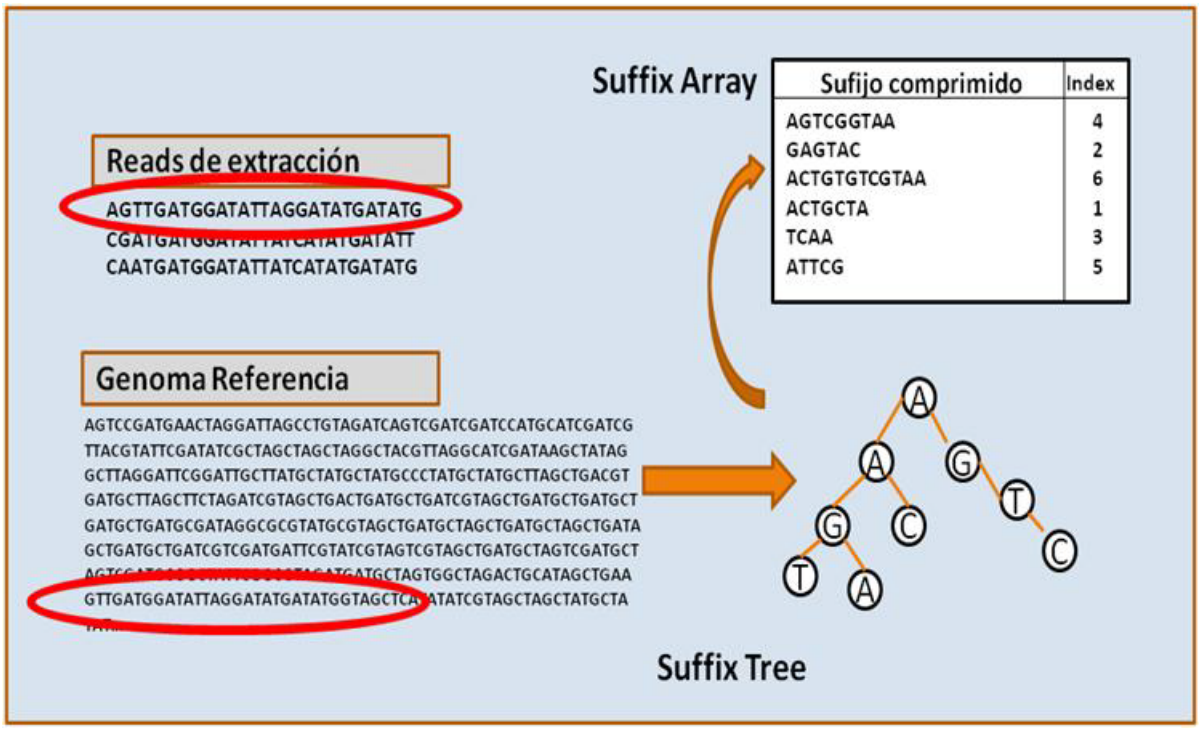

Figura 12 – Burrows Wheeler Aligner.

Capítulo 5 Experimentación

5.1 Introducción

Cuando se tiene la intención de desarrollar una serie de experimentos para comprobar una idea definida, cada uno de ellos tiene una importancia de contextualizar el problema. De esta manera, definir cuáles son los puntos importantes que deben tenerse en cuenta, que a partir de nuestros resultados nos muestran datos comparativos de hasta qué punto se acerca a las respuestas esperadas. Al definir qué puntos son importantes en el contexto de los experimentos, es importante determinar cómo se deben diseñar los escenarios de la experimentación, para responder a las preguntas que hemos definido inicialmente.

Como punto principal, después del diseño de los experimentos, la interpretación y análisis de los resultados son los pasos que toman más tiempo en el contexto general. Para que seamos capaces de hacer conclusiones finales, además de ser capaz de confirmar a través de este estudio si los resultados son satisfactorios, y las conclusiones finales sean correctos.

El objetivo detrás de este de capítulo experimental consiste en realizar una análisis de la aplicación bioinformática BWA (Burrows-Wheeler Align), donde en cada uno de los experimentos se llevan a cabo búsquedas para identificar los factores más importantes en la planificación de tareas e gestión de los recursos.

Inicialmente, para entender mejor la aplicación, se ha realizado un análisis de su rendimiento, el control de su comportamiento en un ambiente paralelo controlado, un cluster no dedicado de 32 nodos. Esto es importante para que pueda tener una visión real, la forma en que la aplicación se comporta cuando dispone de todos los recursos necesarios.

5.2 Escenarios

Para la ejecución de la aplicación BWA, tenemos un cluster de 32 nodos de IBM en el que cada nodo tiene dos procesadores Dual-Core Intel(R) Xeon(R) a 3.00GHz con 4MB de memoria L2, 12 GB de RAM y 160 GB de disco SATA. El punto principal es el procesador de doble núcleo con la memoria compartida, donde se centra nuestro trabajo.

Nuestros experimentos sólo están dirigidas a la arquitectura multi-core, usamos un solo nodo, donde hemos compilado la aplicación BWA, que tiene versiones disponibles en: http://bio-bwa.sourceforge.net/ en este caso, la versión más reciente estable desarrollada en pthreads.

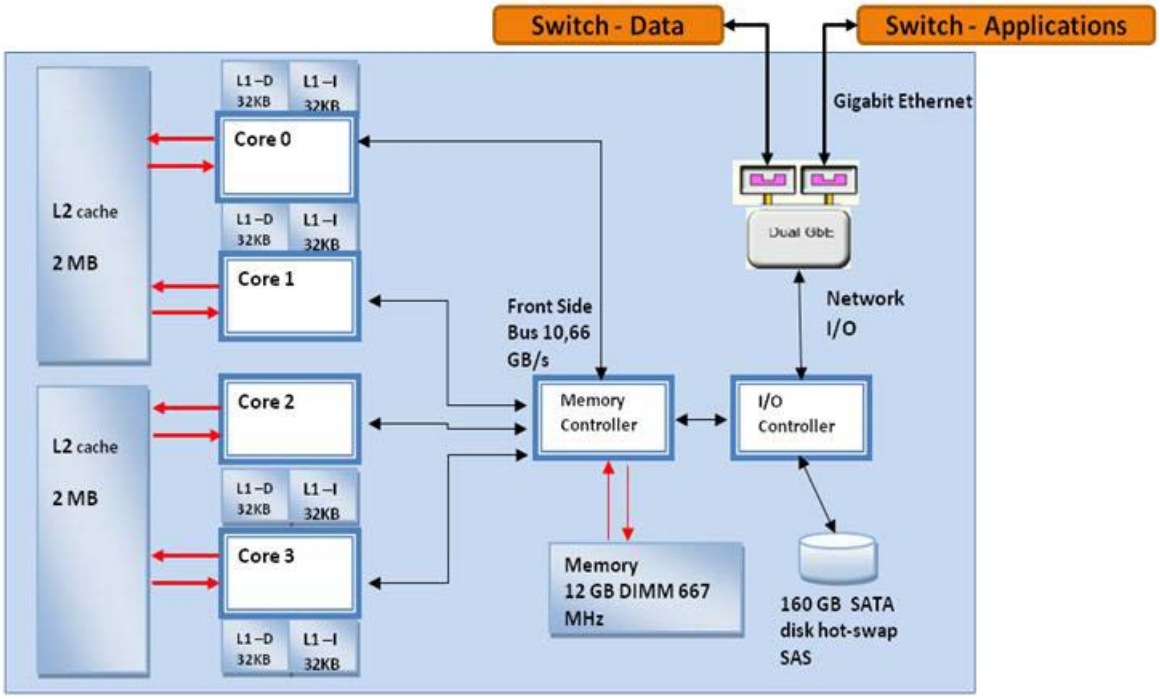

Figura 13 – Arquitectura del nodo.

Después de desempaquetar BWA en el nodo que queremos trabajar, hicimos la compilación usando GCC (GNU Compiler Collection) versión 4.3.2. El genoma humano de referencia utilizado ha sido indexado antes, utilizando la heurística de Smith-Waterman. Esta es relativamente más lenta, pero tiene una tasa de error inferior durante lo alineamiento.

La longitud por defecto de reads de extracción, hicimos uso 30bp para mantenerse por debajo de la 200bp según lo recomendado el Manual de BWA. También evitamos usar reads demasiado grande que nos obligan a esperar mucho por los resultados, se tratando que lo proceso de alineamiento ya es demasiado longo.

El alineamiento realizados por BWA tiene varias opciones que maximizan la calidad o la velocidad del alineamiento. Dado que el objetivo principal no se centra en las diferencias entre una o otra opción, sólo utilizamos la opción -t, que permite el modo multi-threading y establece la cantidad de threads que desea utilizar. Así, exploramos el uso de los cores diferentes del procesador.

Para el desarrollo de la experimentación, también se utiliza una aplicación NAS Parallel Benchmark desarrollada en OpenMP, que está disponible en: http://www.nas.nasa.gov/. En este caso usamos la versión desarrollada en lenguaje de programación C, la versión 2.3. El NAS elegido fue el BT, que hace la resolución de un sistema de síntesis de PDEs no lineales a través

del bloque algoritmo tridiagonal. La clase que elegimos es B, que tiene un tamaño de 102x102x102.

5.3 Conocimiento de la Aplicacion

Cuando se trata de una aplicación paralela, una de las cuestiones definidas es hasta qué nivel puede escalar la aplicación y cuál es su comportamiento dentro de un entorno informático en particular. Reconociendo que existe una limitación de los conocimientos acerca de la aplicación que utilizamos, se plantea la siguiente pregunta: cómo se comportar BWA frente a un Genoma humano de 3 GB. Un genoma humano completo con cinco opciones reads de extracción, 14MB, 140MB, 319MB, 639MB y 1200MB, mientras se ejecuta sin threads BWA. Con dos o cuatro hilos, y que para este tipo de ejecución tiene con tiempo y el consumo de los recursos.

Para entender el paralelismo realizado en BWA, se necesita de la comparación de la ejecución secuencial y el comportamiento, para demostrar los cambios con el paralelismo que ofrece la opción de threads. Para lograr esto, utilizamos una ejecución normal de alineamiento de BWA. Hicimos una llamada al ejecutable sin ofrecer la opción de threads. Informar a la ubicación de nuestra referencia del genoma humano, y los reads de extracción que queremos usar. Como salida de nuestra aplicación tenemos un archivo de formato SAM (Sequence Alignment/Map). La idea de utilizar cinco opciones de reads de extracción es tener un límite inferior y un límite superior, además de la más común para hacer comparaciones.

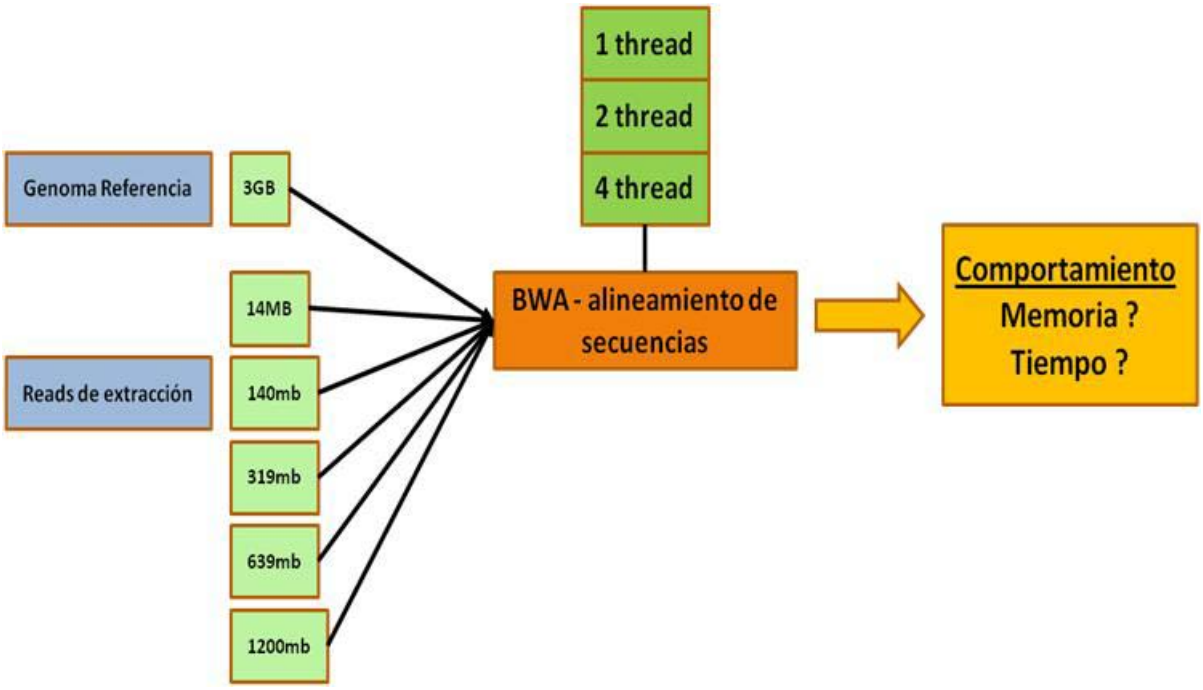

Figura 14 –Conocimiento de la aplicación.

5.3.1 Evaluación de los resultados

Se observa el tiempo de ejecución para la aplicación con 1, 2 y 4 threads, con diferentes tamaños de lecturas de extracción, respectivamente, de las cuales se ilustran en la Figura 15.

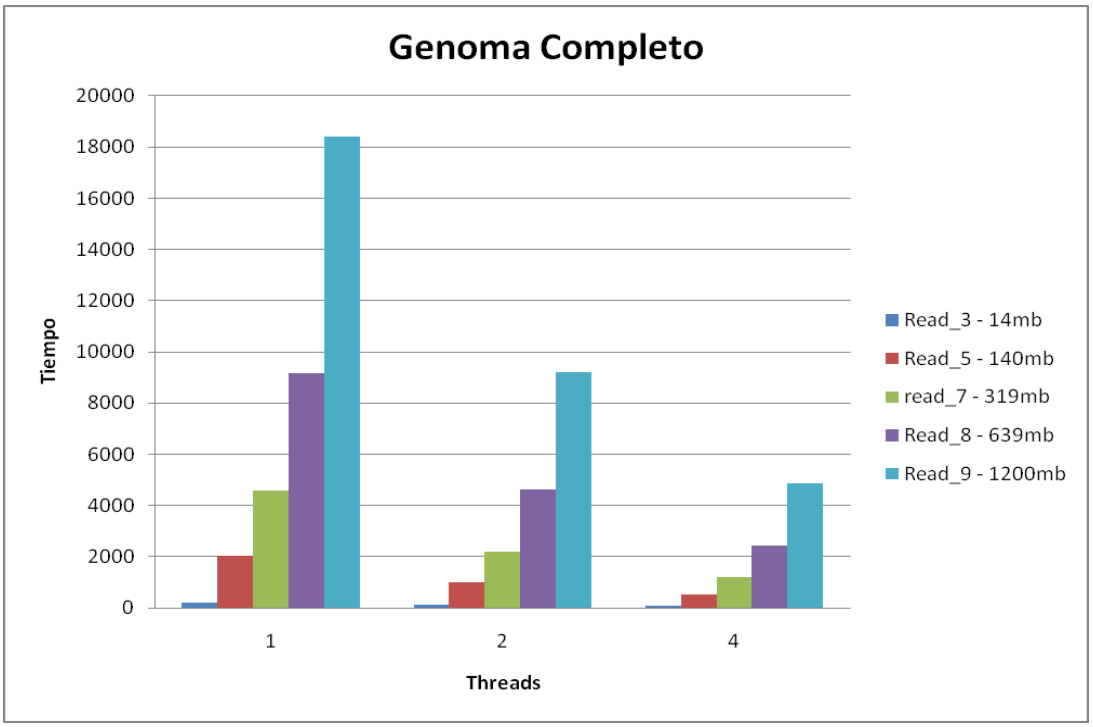

Figura 15 – Tiempo de alineamiento frente los tamaños de lecturas de extracción y threads.

En la figura 16, observamos que el tiempo de ejecución de la aplicación crece con el tamaño de la carga de trabajo que se ha seleccionado, como ya se ha definido.

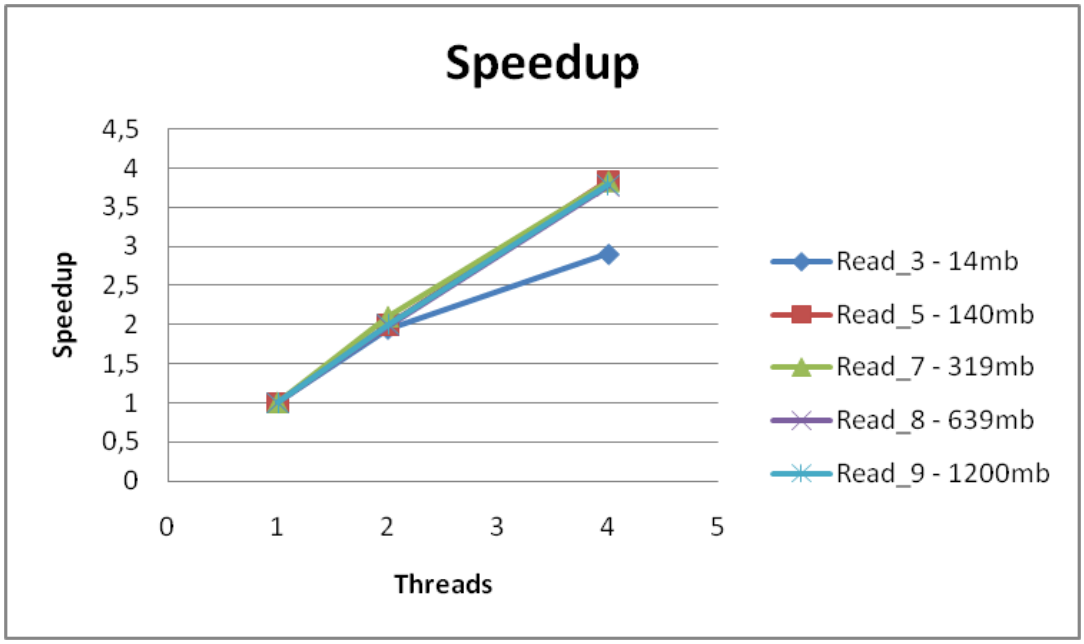

Figura 16 – Speedup.

Obtener un speedup de dos para dos threads casi cuatro para cuatro threads, lo que demuestra que la adaptación de la aplicación BWA escala casi cerca del ideal para la mayoría de las cargas. En contraste, se observó que mediante el uso de tamaños muy pequeños de reads de extracción, el trabajo de la aplicación es demasiado grande para la cantidad de datos que se hace uso. Esta situación se observa en la Figura 17, donde se demuestra la eficiencia de la aplicación, donde read_3, pierde alrededor del 30% de eficiencia.

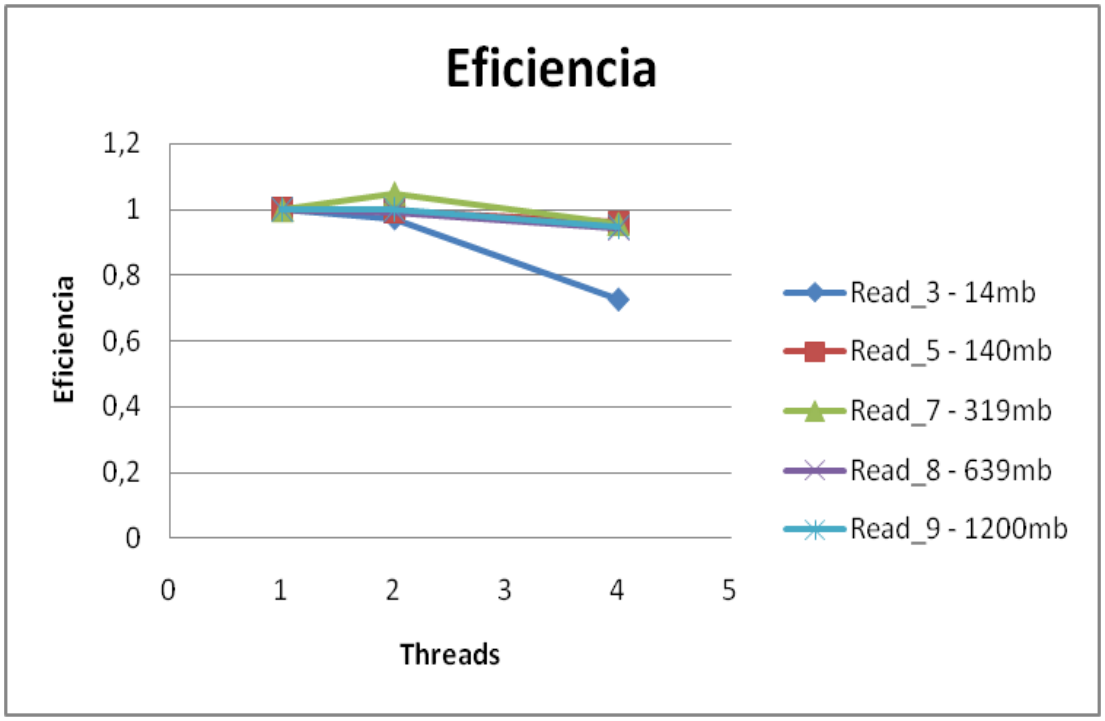

Figura 17 – Eficiencia.

Durante la ejecución, hicimos uso de la información procedente de SGE (Sun Grid Engine), podemos ver que el alineamiento a través de BWA llega a un pico del uso de la memoria principal de 3,4 GB. Menor que la esperada, teniendo en cuenta que se trata de un 3GB del Genoma Humano y reads de extracción con 1,2 GB lo más grande.

5.4 Limitación de la Memoria

Después de entender el funcionamiento de la aplicación, basados en experiencias previas, la hipótesis era: si hay límite de memoria del sistema, la aplicación empieza a traer los datos del disco local, ya que no hay memoria disponible.

La idea de este experimento es que, al limitar el sistema de memoria, la aplicación cambie la forma de trabajar. Traer datos desde el disco local, penaliza la aplicación en tiempo de ejecución, haciendo uso de ancho de banda del disco. Esta situación es importante, como ya se ha definido, una de las cuestiones que este trabajo es el desarrollo de políticas de gestión de recursos. Utilizar los recursos que están libres en un periodo determinado de tiempo.

Para este experimento, nos fijamos el mismo genoma humano de 3GB utilizado anteriormente, y los mismo que reads de extracción anterior, para no cambiar el contexto general del escenario. Hemos en utilizar para 1, 2 y 4 threads, no cambiando el foco de las

ejecuciones multi-core. Un cambio significativo para este experimento son los tres tipos de limitaciones de memoria del sistema en 1GB, 2GB y 3GB.

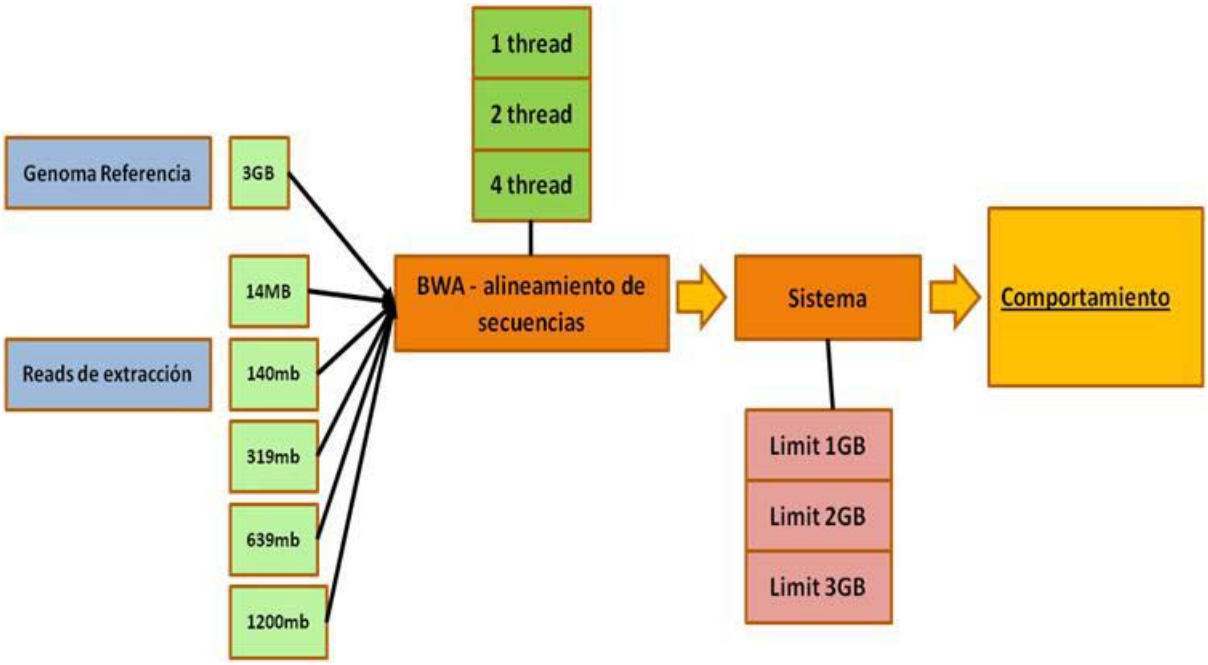

Figura 18 – Experimento Limitación de la memoria.

5.4.1 Evaluación de los resultados

Para este experimento se observó que no confirma la hipótesis previamente definido. La aplicación finalmente devuelve un mensaje de error, detiene su ejecución e informa la falta de memoria disponible. Una política de recursos para esta situación sería la solución, para que independientemente de la falta memoria, la ejecución podría proceder con los requisitos mínimos.

5.5 División de los datos

Haciendo la evaluación de los resultados del experimento anterior, la idea es trabajar con la división de datos, de modo que quepan en la memoria, la solución más adecuada y que utilizan para aplicaciones que con grandes cantidades de datos. Sin embargo una tarea para la que ocupa a los desarrolladores de la aplicación, en lugar de Scheduler, como se define el objetivo futuro de este trabajo. Comprender el funcionamiento de la aplicación BWA por la división de sus datos y su comportamiento es importante para el desarrollo de políticas de planificación.

Para desarrollar este experimento, se utilizó el mismo genoma humano 3GB, ya propuesto anteriormente. El principal cambio en este escenario es la división del genoma en más partes.

Dado que la decisión de dividir el genoma es una tarea un tanto complicada para saber exactamente donde a partir de los datos, por lo que la división no afecte negativamente al alineamiento final. La partición de datos de manera incorrecta puede generar resultados no válidos. La decisión correcta fue utilizar un grupo de cromosomas individuales, porque esto sería una forma de trabajar con la parte del genoma, sin comprometer los resultados finales. Para este experimento se utilizaron los cromosomas 1, 2, 6, 7, 21 y 22, que representaron valores expresivos durante el alineamiento individual.

El tamaño total de la carga fue seleccionado en 940MB, ligeramente por debajo del límite propuesto por primera vez en el experimento anterior, que se establece en 1 GB. Como reads de extracción hemos utilizado la carga de 319MB.

Para este experimento, hemos establecido la siguiente hipótesis: Ejecutar la aplicación con una menor cantidad de carga y limitación de memoria por SGE (Sun Grid Engine) nos permiten iniciar la aplicación. Pero en comparación con el alineamiento del genoma humano completo hecho con los mismos parámetros. Hubo una penalización de tiempo tal y como cita en el experimento inicial, ya que BWA es el tipo de aplicación que es más eficiente con mayor volumen de trabajo.

5.5.1 Evaluación de los resultados

El comportamiento de la aplicación BWA con la partición de la carga de trabajo puede verse en la Figura 19.

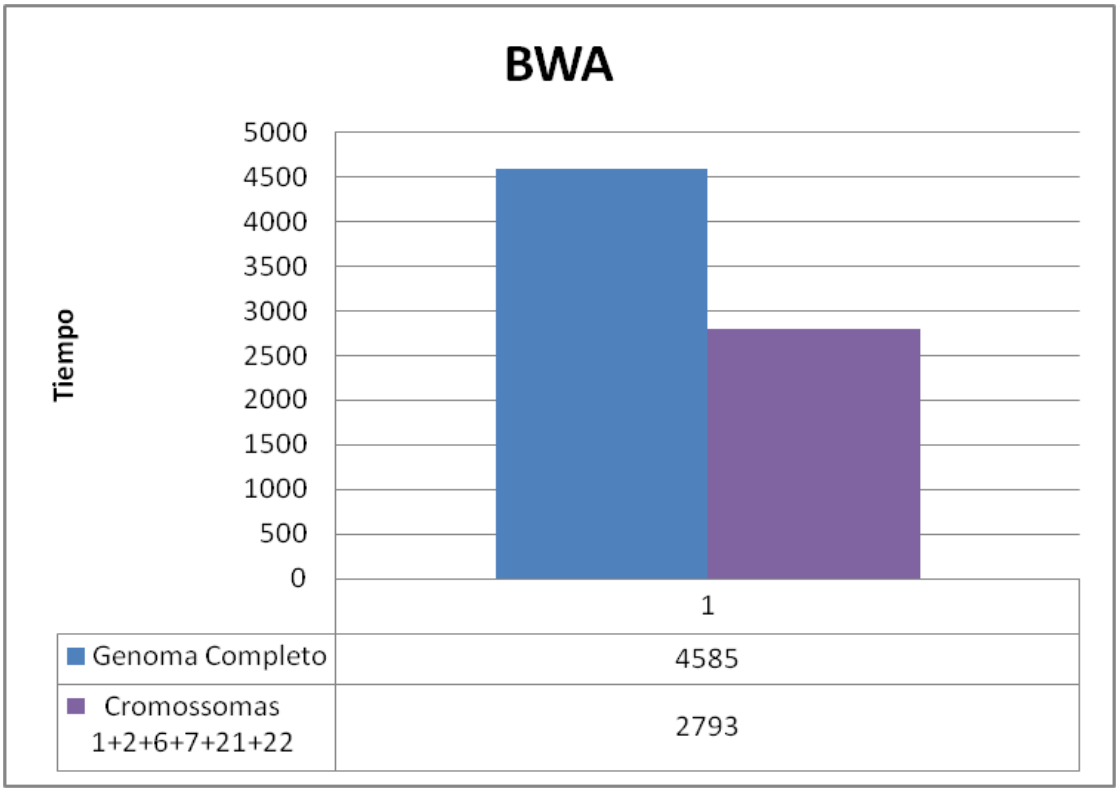

Figura 19 – Genoma Completo vs Genoma partido.

Podemos ver que, en comparación con el genoma humano completo, el reparto de la carga de trabajo genera una gran penalización de tiempo. Sabiendo que la carga de trabajo de salida representa aproximadamente un tercio de todo el genoma, sin embargo, la alineación de las partes del genoma representa el 60% del tiempo empleado para hacer la alineación del genoma completo. El uso máximo de memoria se mantuvo en 750 MB, mientras que el genoma completo con 3,4 GB de memoria. Para una política de planificación es importante tener en cuenta que la partición de los datos es una opción, aunque haga una penalización en los tiempos como muestra la figura 19. Los entornos informáticos en general no quedan con recursos libres, entonces hacer la ejecución con los recursos disponibles es primordial.

5.6 Degradación entre los threads

El principio de este experimento se basa en una hipótesis que ha sido estudiado por otros autores, en el que los threads que se ejecutan en la misma zona de memoria pueden competir por la utilización de los recursos compartidos.

La idea es que esta situación provoca la degradación, en vez de lo que podría lograrse mediante la ejecución en un ambiente libre de contención de los recursos [**6**]. El punto principal

de este experimento es que la distribución de cargas de trabajo entre los cores aumenta o disminuye la degradación de la aplicación.

Para lograr los resultados, decidimos hacer este experimento sin el uso de la afinidad entre los procesos, eliminando la táctica del Scheduler en enviar siempre el mismo proceso para el procesador que se ejecutó anteriormente, para hacer uso de los datos, en particular, podría haber dejado en su memoria caché [**23**]. De modo que podemos evaluar como se degrada distribución de la carga de los procesos que se ejecutan dentro de la arquitectura de multi-core.

Para este experimento, se utiliza el Genoma Humano de 3GB, se utiliza un conjunto de read de extracción de 140MB. También realizamos la ejecución de dos aplicaciones en el mismo nodo de multi-core, para que ambas compartiesen recursos. En este caso usamos la aplicación BWA que se utilizada anteriormente, en sus opciones de 1, 2 y 4 threads, y en de forma concurrente ejecutamos NAS Parallel Benchmark BT clase B , con la opción de 1, 2 y 4 threads.

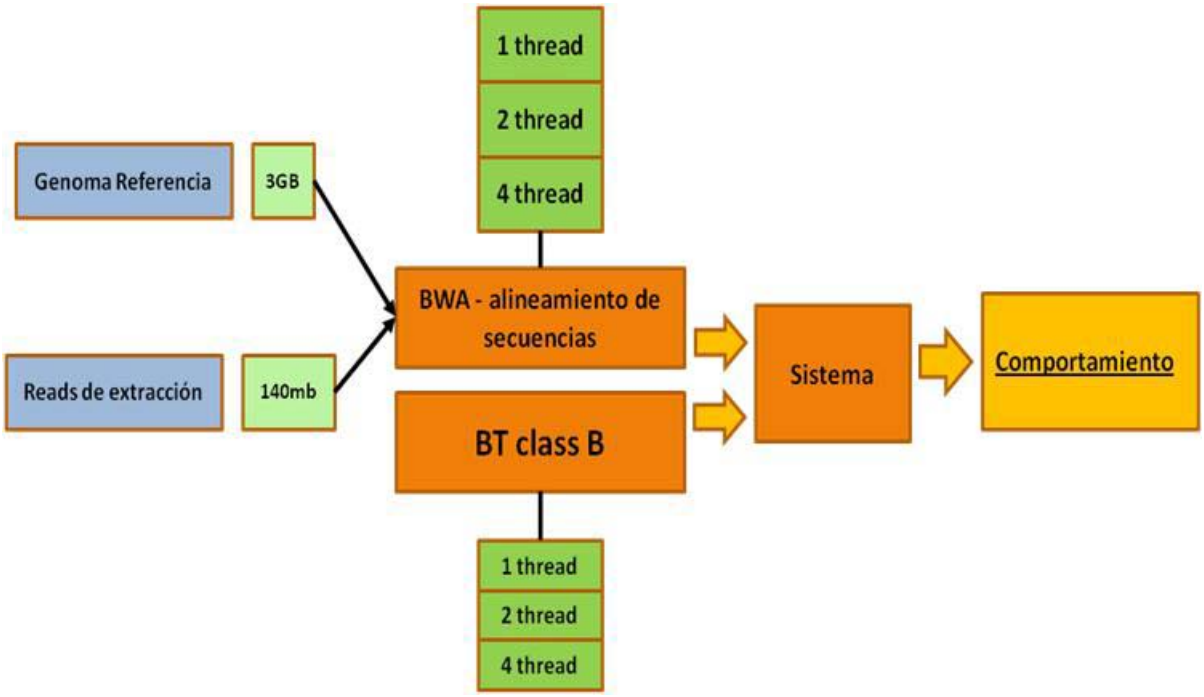

Figura 20 – Experimento degradación entre los threads.

5.6.1 Evaluación de los resultados

En la figura 21, mostramos la distribución de la carga entre los núcleos del procesador, de acuerdo con la cantidad de threads que se asigna a cada aplicación.

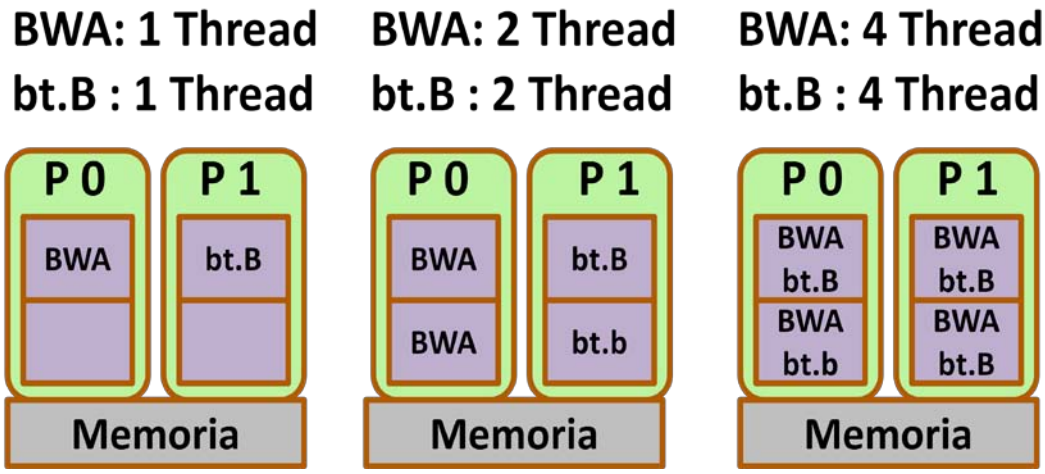

Figura 21 – Distribución de la carga en los procesadores.

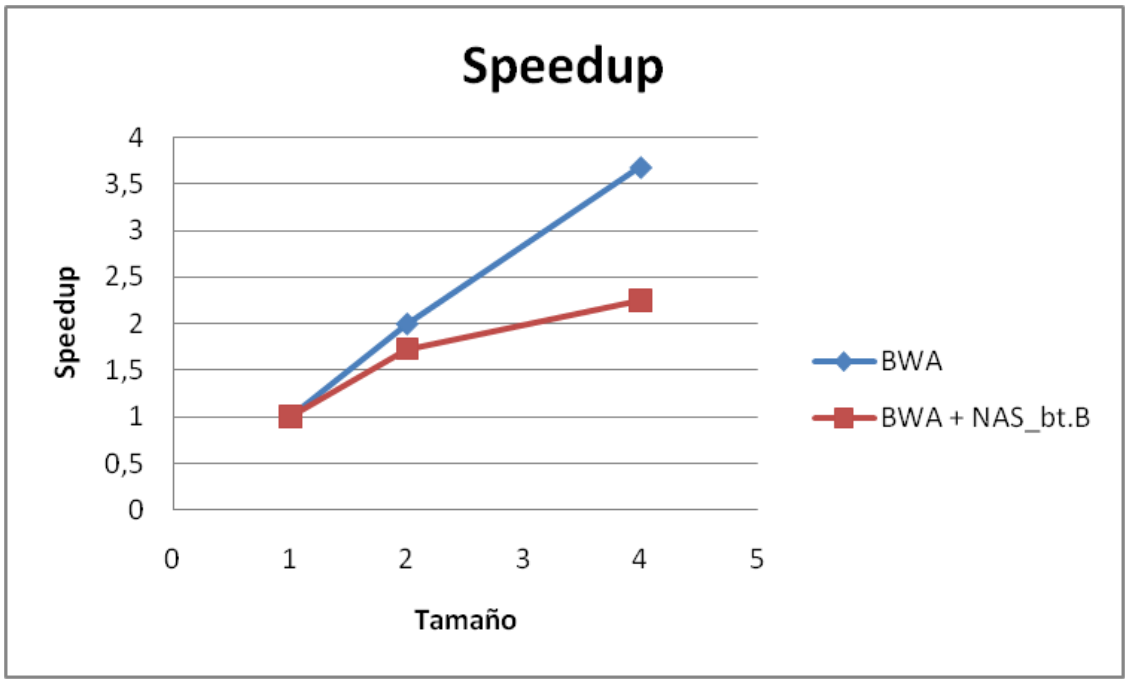

Figura 22 – Speedup BWA vs BWA+NAS.

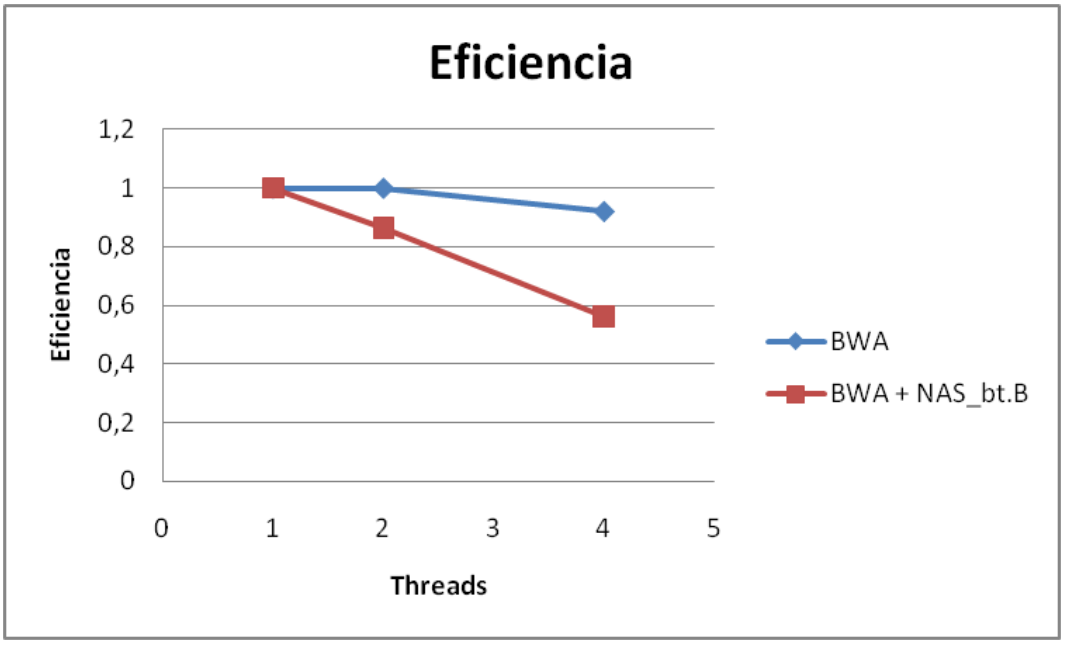

Figura 23 – Eficiencia BWA vs BWA+NAS.

Podemos ver que cuando se ejecuta BWA solo, haciendo uso de los recursos exclusivamente, su speedup esta cada vez más a cerca del ideal, como se muestra en el comportamiento de la aplicación en los experimentos anteriores. En cuanto al reparto de recursos en el nodo multi-core, podemos ver que la ejecución con un thread para cada aplicación, no hay degradación del BWA. Para ejecutar con dos threads, la degradación comienza a ser visible, aunque ambos siguen teniendo recursos disponibles para ambas aplicaciones. Esta situación refleja el uso de memoria compartida, ya que el uso ha de ser más intenso, y los cambios de contexto se vuelven frecuentes. Para la opción de cuatro threads, ambas aplicaciones no solo comparten la memoria, sino que también hace comparten el tiempo de CPU. Situación que tiene consecuencias sobre el rendimiento, ya que la arquitectura no tiene la tecnología multi-threading, lo que obligó a realizar cambios de contexto entre las dos aplicaciones. La misma situación se muestra en la Figura 23, donde la eficiencia de la aplicación, cuando se utiliza todos los recursos disponibles, la comparación de la ejecución que comparte recursos con NAS. La pérdida de eficiencia es del 10% con dos threads, y casi el 50% para los cuatro threads.

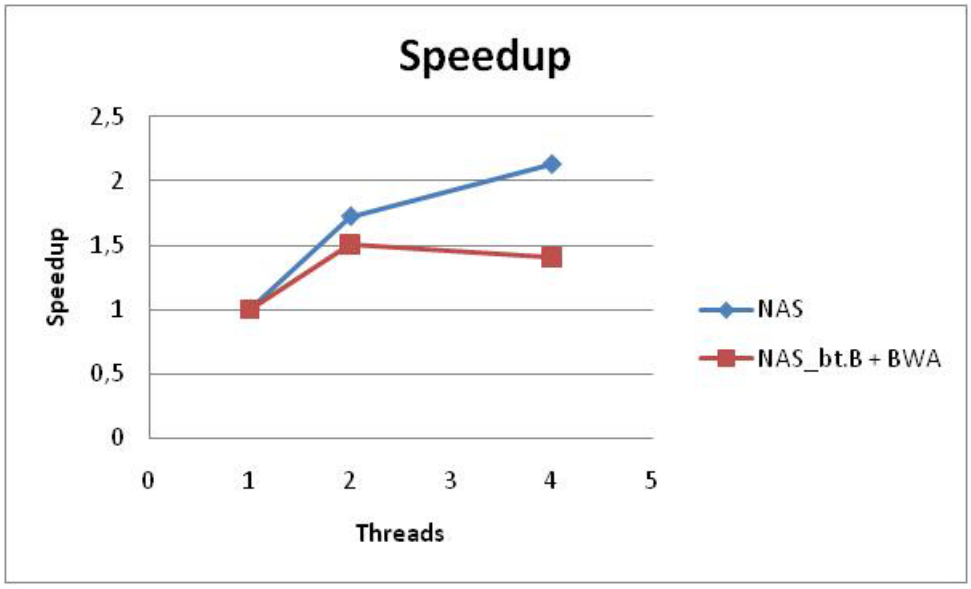

Figura 24 – Speedup NAS vs NAS+BWA.

Sobre la influencia de la BWA frente al NAS, podemos ver la en la Figura 24, que el NAS que se utiliza no escala bien como BWA, incluso en ejecuciones donde tiene todos los recursos, alcanzando un pico de 2,1. Frente el uso de la BWA, se verificó que la degradación sigue la misma situación que sufre BWA cuando ejecutamos NAS, pero a un ritmo más grande. Como hemos visto anteriormente, BWA hace un mejor uso de los recursos que tiene disponible, lo que explica su ventaja en relación la ejecución del NAS.

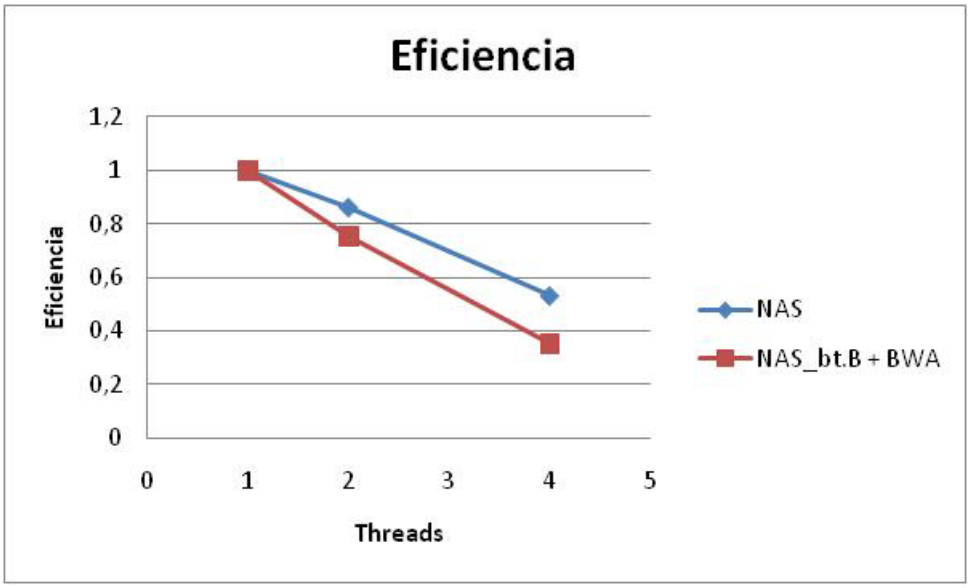

Figura 25 – Eficiencia NAS vs NAS+BWA.

Podemos ver en la Figura 25, que la aplicación NAS que usamos para nuestra experimentación, todavía es una aplicación que incluso con todos los recursos disponibles, no tiene un buen rendimiento. Por lo tanto podemos concluir que aún no es la mejor opción de aplicación para hacer una mezcla de cargas, porque hace un buen uso del paralelismo que el procesador puede ofrecer. Su eficiencia en el procesador solamente, mostró una pérdida de alrededor de 20% para 2 y 4 threads, un punto más que nos permite decir que dicha aplicación no hace buen uso de la capacidad del procesador. Esta situación controlada por una política de gestión de los recursos podría aumentar el uso de la aplicación BWA frente a aplicación NAS. Situación que nos deja un camino a seguir en futuros experimentos, ya que la posibilidad de potenciar el desempeño de una aplicación contra otra, que no utiliza los recursos disponibles de una manera útil es claramente demostrada en este experimento.

Capítulo 6 Conclusiones y trabajo futuro.

6.1 Conclusiones

El objetivo de este proyecto de investigación es contribuir a la solución de los problemas de planificación de aplicaciones en el nodo multi-core de memoria compartida. Nuestro estudio tuvo como punto principal, analizar y comprender los problemas de la planificación en los nodos de cómputo multi-threaded, multi-core y multi-procesador de memoria compartida, arquitectura que hoy en día se utiliza de manera importante. La planificación de aplicaciones paralela en arquitecturas multi-core, todavía es un tema que deja muchas líneas abiertas, y abre muchas vías de mejora.

Hacer coexistir una carga en paralelo y una carga local, de manera que ambos puedan hacer un buen uso de estos recursos en lo entorno informático. Durante la ejecución de ambas cargas, la carga local puede ser ejecutado sin verse afectada por cualquier tipo de interactividad que puede tener en lo entorno. Por el contrario, las cargas paralelos deben hacer un uso eficiente de los recursos que están disponibles. Esto implica la creación de nuevos sistemas y métodos de planificación, para que ambas las cargas puedan ser administradas de manera transparente al sistema. Proponemos el uso del paradigma MapReduce que permite la automatización y la simplificación de la planificación, lo que permite un mayor control por el sistema y menos por el programador.

Persiguiendo este objetivo, se utilizó una aplicación bioinformática BWA, además de la aplicación NAS Parallel Benchmark para contextualizar y comprender cómo el planificador maneja diferentes cargas, pero lo hacen con uso intensivo de cómputo y memoria. También se realizó una revisión bibliográfica destinada a la planificación en las arquitecturas multi-core y gestión de recursos.

Con base en estudios, y los resultados de las ejecuciones observadas, se concluye que el uso de MapReduce para las arquitecturas multi-core proporciona una ganancia en la planificación de tareas. Permitir la distribución y equilibrio de carga, lo que facilita el proceso de creación de aplicaciones paralelas que utilizan gran cantidad de datos. La mescla de las cargas en las arquitecturas multi-core, sobre el suporte del paradigma de MapReduce proporciona beneficios y reduce los problemas de cuellos de botella en relación con el uso de memoria compartida. Una mezcla de cargas de trabajo dentro de una arquitectura multi-core es muy variable dependiendo de su ubicación, y la memoria compartida tiene una gran influencia

en él. Esta situación nos deja muchos trabajos futuros que debe seguir como en objetivos a largo plazo.

6.2 Trabajos Futuros

Al llegar a la meta establecida en este trabajo de investigación, las líneas siguen abiertas para que podamos avanzar, ampliando los conocimientos sobre la planificación del trabajo en la arquitectura multi-core. Para esto, nuestra intención que este trabajo de investigación permita incorporar nuevas estrategias y metodologías para mejorar la planificación de las tareas. Aumentar las posibilidades de mezcla e identificar más allá del uso de memoria compartida, otros puntos de contención de los recursos que son de gran influencia en el rendimiento de las aplicaciones.

La definición de la política de planificación tareas, consiste en una gran cantidad de normas y procedimientos destinados a promover una buena distribución de tareas y el buen uso de los recursos dentro de una arquitectura determinada. Esto nos permite mejorar aún más el área del desarrollo de este trabajo y más elementos que pueden ser cruciales en las estrategias de la planificación.

Un factor importante en nuestra investigación fue que hicimos nuestros experimentos en un entorno de cómputo controlado. Así que queremos en un enfoque cercano ser capaz de aplicar políticas, en un entorno de uso real, que hace uso de otras aplicaciones, de manera que pueda proporcionar nuevas situaciones y aumentar nuestras posibilidades de conocimiento.

MapReduce todavía es un paradigma muy nuevo, teniendo trabajos recientemente publicados. Un estudio más detallado de este paradigma aún puede contribuir con más soluciones y situaciones que son útiles para nuestro trabajo en un seguimiento futuro.

Como línea abierta, la gestión de los recursos con el uso de I/O para aplicaciones que se integran el trabajo es un camino a seguir. Dónde podemos plantear la gestión aplicaciones que hacen uso de muchos GB de datos, e permitan una administración eficiente de los recursos y mantener el foco en la línea de la computación de alto rendimiento.

Bibliografía

[1] Mauricio Hanzich, "Un Sistema de Planificación Temporal y Espacial para Clusters no Dedicados," M.S. thesis, Universitat Autònoma de Barcelona, 2006.

[2] José R. García, "Planificación de Aplicaciones Best-Efforty Soft Real-Time en NOWs," M.S. thesis, Universitat Autònoma de Barcelona, 2007.

[3] Morris A. Jette, Andy B. Yoo, and Mark Grondona, "SLURM: Simple Linux Utility for Resource Management," , 2002, pp. 44-60.

[4] Michael Litzkow, Miron Livny, and Matthew Mutka, "Condor - A Hunter of Idle Workstations," in *Proceedings of the 8th International Conference of Distributed Computing*, 1988.

[5] Dror G. Feitelson and Larry Rudolph, "Gang Scheduling Performance Benefits for Fine-Grain Synchronization," *Journal of Parallel and Distributed Computing*, vol. 16, pp. 306-318, 1992.

[6] Alexandra Fedorova, Sergey Blagodurov, and Sergey Zhuravlev, "Managing contention for shared resources on multicore processors," *Commun. ACM*, vol. 53, no. 2, pp. 49-57, 2010.

[7] Heng Li and Richard Durbin, "Fast and accurate short read alignment with Burrows-Wheeler transform.," *Bioinformatics (Oxford, England)*, vol. 25, no. 14, pp. 1754-1760, 2009.

[8] G. Tan, L. Xu, Z. Dai, S. Feng, and N. Sun, "A Study of Architectural Optimization Methods in Bioinformatics Applications," *Int. J. High Perform. Comput. Appl.*, vol. 21, no. 3, pp. 371-384, 2007.

[9] T. W. Lam, W. K. Sung, S. L. Tam, C. K. Wong, and S. M. Yiu, "Compressed indexing and local alignment of DNA," *Bioinformatics*, vol. 24, no. 6, pp. 791-797, 2008.

[10] Wing-Kai Hon, Tak-Wah Lam, Kunihiko Sadakane, Wing-Kin Sung, and Siu-Ming Yiu, "A Space and Time Efficient Algorithm for Constructing Compressed Suffix Arrays," *Algorithmica*, vol. 48, no. 1, pp. 23-36, 2007.

[11] Eitan Frachtenberg, "Process scheduling for the parallel desktop," in *Parallel Architectures,Algorithms and Networks, 2005. ISPAN 2005.Proceedings. 8th International Symposium on*, 2005.

[12] James H. Anderson and John M. Calandrino, "Parallel Real-Time Task Scheduling on Multicore Platforms," in *Proceedings of the 27th IEEE International Real-Time Systems Symposium*, Washington, DC, USA, 2006, pp. 89-100.

[13] James H. Anderson, John M. Calandrino, and UmaMaheswari C. Devi, "Real-Time Scheduling on Multicore Platforms," in *Proceedings of the 12th IEEE Real-Time and Embedded Technology and Applications Symposium*, Washington, DC, USA, 2006, pp.

179-190.

[14] John M. Calandrino, Dan Baumberger, Tong Li, Scott Hahn, and James H. Anderson, "Soft Real-Time Scheduling on Performance Asymmetric Multicore Platforms," in *Proceedings of the 13th IEEE Real Time and Embedded Technology and Applications Symposium*, Washington, DC, USA, 2007, pp. 101-112.

[15] Susanne M. Balle and Daniel J. Palermo, "Enhancing an open source resource manager with multi-core/multi-threaded support," in *Proceedings of the 13th international conference on Job scheduling strategies for parallel processing*, Seattle, WA, USA, 2008, pp. 37-50.

[16] Jochen Liedtke, Marcus Volp, and Kevin Elphinstone, "Preliminary thoughts on memory-bus scheduling," in *Proceedings of the 9th workshop on ACM SIGOPS European workshop*, Kolding, Denmark, 2000, pp. 207-210.

[17] Jeffrey Dean and Sanjay Ghemawat, "MapReduce: simplified data processing on large clusters," *Commun. ACM*, vol. 51, no. 1, pp. 107-113, 2008.

[18] Colby Ranger, Ramanan Raghuraman, Arun Penmetsa, Gary Bradski, and Christos Kozyrakis, "Evaluating MapReduce for multi-core and multiprocessor systems," in *Proceedings of the 13th International Symposium on High-Performance Computer Architecture*, 2007, pp. 13-24.

[19] Ben Langmead, Cole Trapnell, Mihai Pop, and Steven L Salzberg, "Ultrafast and memory-efficient alignment of short DNA sequences to the human genome," *GENOME BIOLOGY*, vol. 10, no. 3, 2009.

[20] Giovanni Manzini, "An Analysis of the Burrows-Wheeler Transform," *Journal of the ACM*, vol. 48, p. 2001, 2001.

[21] Juha Karkkainen, Peter Sanders, and Stefan Burkhardt, "Linear work suffix array construction," *J. ACM*, vol. 53, no. 6, pp. 918-936, 2006.

[22] Peter and Rodelsperger, Christian and Jager, Marten and Jostins, Luke and Bauer, Sebastian and Robinson, Peter N. Krawitz, "Microindel detection in short-read sequence data," *Bioinformatics*, vol. 26, no. 6, pp. 722--729, 2010.

[23] Vahid Kazempour, Alexandra Fedorova, and Pouya Alagheband, "Performance Implications of Cache Affinity on Multicore Processors," in *Proceedings of the 14th international Euro-Par conference on Parallel Processing*, Las Palmas de Gran Canaria, Spain, 2008, pp. 151-161.

Índice alfabético