TRABAJO FIN DE GRADO

Grado en Ingeniería Informática

UNIVERSIDAD DE ALMERIA

ESCUELA SUPERIOR DE INGENIERÍA

Librería para aplicaciones multihebradas adaptativas.

Curso 2017/2018

Alumno/a:
Alberto Morante Bailón

Director:
Juan Francisco Sanjuan Estrada

A mis padres, que me permitieron tener un futuro mejor

Agradecimientos

Quisiera dedicar estas líneas a todos aquellos que han contribuido a que este trabajo sea posible, en primer lugar agradecer a mi tutor, Juan Francisco Sanjuan Estrada, por haberme apoyado en todo y haberme prestado una gran cantidad de tiempo y una paciencia infinita.

También le dedico estas líneas a mi familia, a mis padres Alberto y Victoria, porque me inculcaron los valores y la educación necesarios para afrontar este trabajo y a mi hermana Laura por haber escuchado mis problemas y haberme dado ánimos cuando más lo necesitaba.

No me olvido de mis compañeros, pero en especial hago una mención a Manolo y Claudio, con los cuáles comparto una amistad que nos unirá durante toda la vida y con los que he pasado los momentos más importantes de esta difícil carrera.

Por último quiero agradecer a varios amigos, a Miguel Ángel, por haber contribuido a esos ratos de ocio y a mitigar el estrés que me generaba la realización de este trabajo y a Daniel, por esas largas charlas sobre no rendirse nunca.

Prólogo

La arquitectura multinúcleo de los procesadores diseñados en la última década ofrece la posibilidad de ejecutar aplicaciones multihebradas, lo que requiere aprovechar eficientemente sus capacidades multihilo. En este sentido, se puede mejorar la eficiencia utilizando un estándar de implementación de hebras de bajo nivel llamado POSIX (Portable Operating System Interface Unix), mediante el cual se pueden gestionar las hebras de forma que no compitan entre sí al acceder a los recursos disponibles, lo que permitirá reducir el tiempo de cómputo tanto en sistemas dedicados como no dedicados.

El presente Trabajo Fin de Grado (TFG) se centra en el ámbito de las APIs (Application Programming Interface) centradas en la programación de aplicaciones multihebradas. La API implementada permite interactuar con un gestor de paralelismo (GP) diseñado en [18], de tal forma, que ofrece a los programadores de aplicaciones paralelas hacer uso de las decisiones del GP en relación al número óptimo de hebras a utilizar en cada instante de tiempo, adaptando el número de hebras activas según el criterio de decisión seleccionado. Es decir, el GP permite determinar en tiempo de ejecución el número óptimo de hebras e informa a la aplicación multihebrada para que adapte, según el criterio de decisión seleccionado, el número de hebras activas en un instante de tiempo. El usuario de la aplicación multihebrada dispone de una amplia variedad de criterios de decisión a utilizar por el GP. Sin embargo, para que una aplicación paralela pueda interactuar con el GP seleccionado, el programador debe utilizar la librería `IGP` (Interfaz de Gestor de Paralelismo).

La API implementada en este TFG facilita la labor al programador de aplicaciones paralelas al hacer uso de la librería `IGP` para que las aplicaciones multihebradas diseñadas puedan interactuar con el GP en tiempo de ejecución. Finalmente, el TFG ha tratado de poner las bases para implementar un nuevo criterio de decisión para el GP basado en el consumo del sistema.

Índice general

Índice de figuras

Índice de tablas

Capítulo 1. Introducción.

La tecnología MIC (Many Integrated Cores) que integra varios núcleos en el mismo chip permite disponer de arquitecturas de procesadores con un elevado número de núcleos, por ejemplo, el coprocesador Xeon Phi. En este sentido, las aplicaciones multihebradas pueden ejecutarse de forma poco eficiente cuando aumenta el número de núcleos en el sistema, si el número de hebras en ejecución no es el adecuado.

Las Figuras 1.1 y 1.2 muestran cuando se produjo el cambio de la época de los procesadores de alta velocidad a los procesadores de alto paralelismo [11]. Alrededor de 2005 los procesadores de alta velocidad se establecieron en 4GHz y la frecuencia de reloj no se ha incrementado desde entonces, esto se debe a que la potencia disipada por unidad de superficie crece con el cuadrado de la frecuencia, por lo que se hacen necesarios sistemas de refrigeración más costosos y complejos. Además, el rendimiento de los procesadores depende de otros factores y puede ocurrir que un procesador que trabaja a una frecuencia elevada obtenga un rendimiento inferior que otro con menor frecuencia.

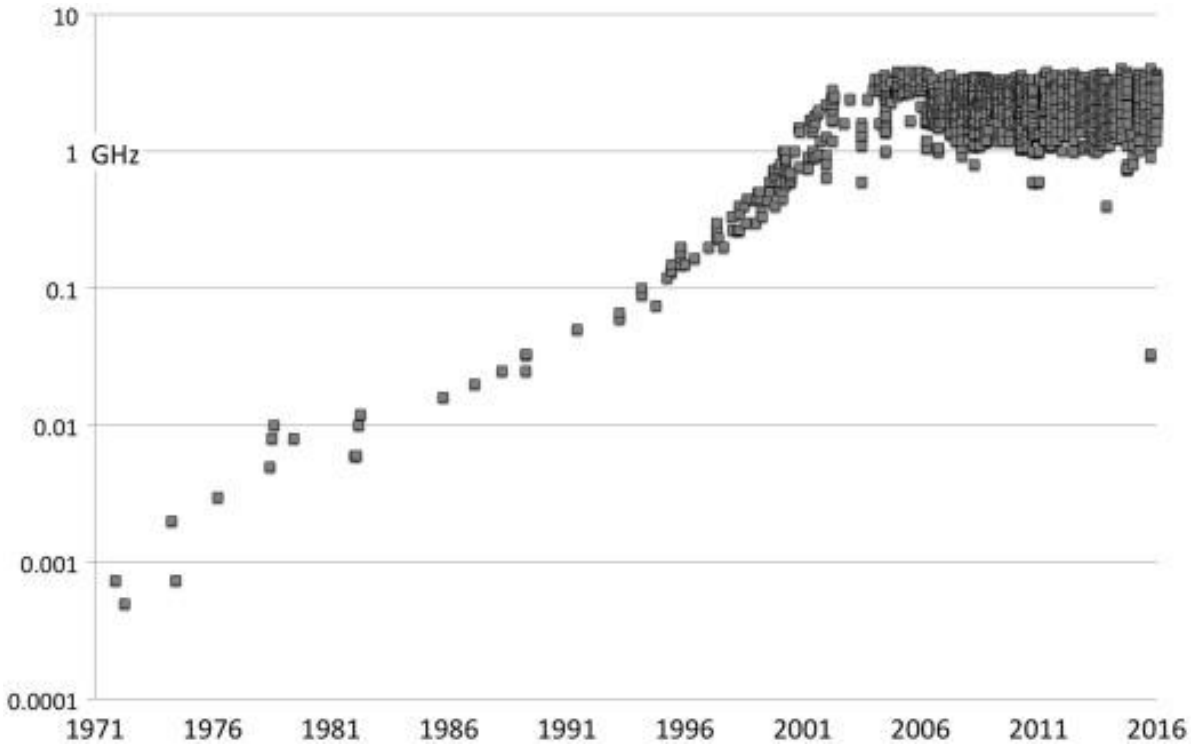

Figura 1.1. Frecuencia de reloj en procesadores/coprocesadores (escala logarítmica). Fuente: [11]

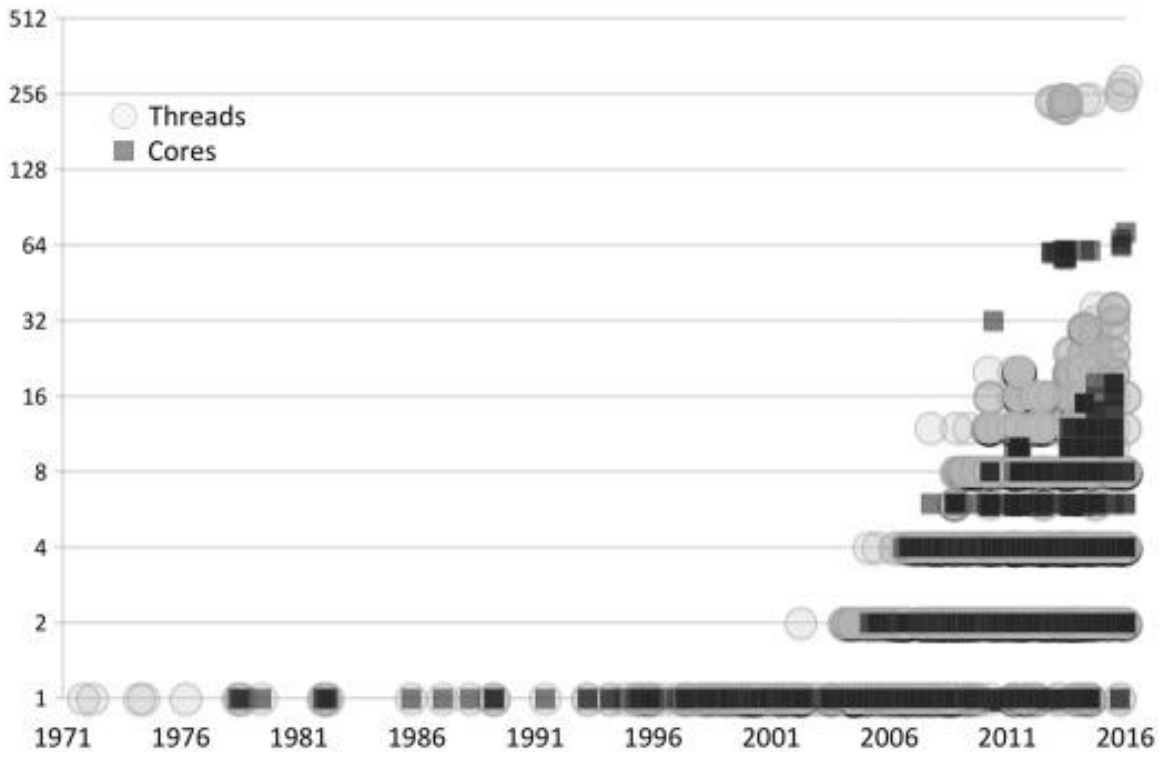

Figura 1.2. Paralelismo por núcleo/hebra (escala logarítmica). Fuente: [11]

Este cambio fue posible debido a que el crecimiento del número de transistores siguió la Ley de Moore [7], por la cual cada año se duplica el número de transistores integrados en un procesador.

Existen aplicaciones multihebradas cuyo rendimiento puede disminuir al utilizar todos los recursos de los sistemas multinúcleo, además, la utilización de estos recursos conlleva también un aumento en el consumo energético. El paralelismo de tareas no funciona bien para todas las aplicaciones multihebradas, un claro ejemplo de ello son los códigos paralelos Branch and Bound (B&B), en los cuales es difícil predecir el trabajo total a realizar ya que el trabajo va surgiendo conforme se va resolviendo el problema, esto conlleva a que el reparto del trabajo entre las distintas hebras activas deba realizarlo en tiempo de ejecución, por tanto surgen irregularidades en la resolución de distintos tipos de problemas [15].

Existen librerías y herramientas que facilitan la programación paralela, sin embargo, estas no terminan de resolver de forma correcta el problema que plantean las aplicaciones multihebradas en sistemas no dedicados, y dejan toda la responsabilidad al sistema operativo (SO) a la hora de repartir los recursos entre las distintas aplicaciones.

Este trabajo se enfoca en el diseño de una API de una librería que interacciona con un gestor de paralelismo, de tal forma que los parámetros iniciales de configuración de la librería se establecen mediante un fichero de configuración. El principal objetivo del TFG es facilitar al programador de aplicaciones paralelas la utilización de los gestores de paralelismo. Esta librería permite que el SO coopere con las aplicaciones multihebradas informándoles del mejor número de hebras a utilizar según el criterio de decisión seleccionado, por ejemplo el rendimiento de la aplicación. En sistemas no dedicados se hace especialmente necesario que las aplicaciones adapten su nivel de paralelismo según el número de recursos disponibles en tiempo de ejecución.

Uno de los factores que afecta directamente al tiempo de ejecución de una aplicación multihebrada es el número de hebras. De manera general, el comportamiento de la aplicación depende directamente de este factor, que normalmente se mantiene invariable a lo largo de la ejecución de la aplicación.

El tiempo de computación de las aplicaciones multihebradas suele ser un factor importante en las aplicaciones High Performance Computing (HPC), para lo cual se propone que el algoritmo multihebrado pueda crear nuevas hebras, o incluso detener alguna hebra activa, de tal forma que el número de hebras activas se adapte dinámicamente a los recursos disponibles en cada momento. Sin embargo, la aplicación debe conocer el número óptimo de hebras en cada instante de tiempo que ofrece el mejor rendimiento posible. El gestor del nivel de paralelismo (GP) determinará periódicamente el número de hebras óptimo e informará a la aplicación para que pueda adaptarse durante la ejecución haciendo frente a las variaciones de las cargas computacionales en el sistema producidas por la propia aplicación [16], o incluso por otras aplicaciones que se ejecutan, en el caso más desfavorable, en un sistema no dedicado [17].

La librería utilizada como interfaz con el Gestor de nivel de Paralelismo (IGP) facilita al programador la implementación de aplicaciones multihebradas capaces de interactuar con distintos tipos de gestores del nivel de paralelismo (GPs). Esta librería proporciona funciones en lenguaje C para comunicar los algoritmos multihebrados con el GP seleccionado, y de esta forma, reducir el trabajo del programador de aplicaciones multihebradas ayudándole en la configuración y manejo del gestor, de tal forma que sea transparente al tipo de gestor utilizado. [10]

1.1. Objetivos.

El objetivo principal del TFG es la creación de una API de la librería IGP que permita a los programadores de aplicaciones paralelas crear aplicaciones multihebradas que puedan interactuar con el gestor de paralelismo a nivel de usuario, de tal forma, que la aplicación de usuario esté informada en cada momento del número óptimo de hebras a ejecutar.

Por otro lado, se publica la librería IGP en código libre y se detalla el manejo de la API diseñada. Además, se ha implementado la utilización de la librería sobre una aplicación multihebrada destinada al alineamiento de las secuencias de ADN. Los experimentos realizados han permitido analizar el comportamiento de la librería IGP al ejecutar la aplicación multihebrada del secuenciamiento del ADN en un coprocesador Xeon Phi. Todas las experimentaciones se realizarán a nivel de usuario, manteniendo el kernel original de la Xeon Phi, delimitando correctamente dos entornos de experimentación, tanto en sistema dedicado, donde una única aplicación multihebrada tenga a disposición todos los recursos del equipo, como en sistema no dedicado, donde la aplicación multihebrada adaptativa compita con otras aplicaciones por los recursos del sistema.

1.2. Fases de realización y cronograma asociado.

Este TFG se estructura en las siguientes fases de desarrollo, habiendo estimado la siguiente temporización de cada parte:

1. Creación de la API para la librería IGP: 10 h

   A partir del código de la librería IGP [10], se adapta la librería para que lea los parámetros de funcionamiento iniciales de un fichero de configuración, de tal forma que permita gestionar el nivel de paralelismo a nivel de usuario.

2. Búsqueda de distintas aplicaciones para Xeon Phi: 10 h

   Búsqueda de código fuente de posibles aplicaciones multihebradas que se puedan ejecutar en el coprocesador Intel Xeon Phi y que utilicen hebras POSIX, así como la selección de una aplicación que abarque un completo abanico de diferentes situaciones, con la posibilidad de programar un benchmark propio que permita analizar de manera más controlada el comportamiento de la librería IGP. En esta fase, se utiliza la aplicación de secuencialización del ADN.

3. Integración de la aplicación seleccionada con la librería IGP: 20 h

   A partir del código fuente de la aplicación seleccionada se realizan las modificaciones necesarias para que utilice la librería IGP, a través de la API implementada anteriormente, de tal forma, que la aplicación pueda adaptar el número de hebras en ejecución.

4. Experimentación del comportamiento de la aplicación con y sin librería: 100 h

   Diseño de distintos entornos de ejecución, tanto en sistemas dedicados, como en sistemas no dedicados, donde se analiza el rendimiento del benchmark, comparando las mejoras obtenidas al utilizar la librería IGP frente a su no utilización. Los parámetros analizados más representativos que definen el comportamiento de cada benchmark han sido el número de hebras, el speed-up y la eficiencia.

5. Modificación de la librería IGP: 100 h

   Análisis del código fuente de la librería IGP para detectar irregularidades o deficiencias en el código, solucionando los problemas y planteando otras alternativas más eficientes, y reevaluando su comportamiento respecto a la librería IGP original. Se añade el código necesario para que la librería permita al proceso principal de la aplicación crear hebras. Además, se propone un método que permite gestionar la creación de hebras en función de su consumo energético.

6. Difusión del TFG: 100 h

   Selección de distintas alternativas que permiten dar a conocer esta librería por Internet, por ejemplo, diseño de página web, comentarios en foros y difusión en portales especializados. Además, se publica en la página web diseñada, tanto el código libre como la ayuda en español e inglés, para su correcta utilización. En este apartado, también se incluye la redacción de la memoria del TFG y la elaboración de presentaciones y videos para la defensa del TFG.

Finalmente, la planificación temporal ha quedado de la siguiente manera, en la Tabla 1.1 se muestra la temporización y en la Figura 1.3 el cronograma asociado. El número total de horas para la realización de este TFG ha sido de 390, ya que se han invertido 50 horas adicionales en la fase de modificación de la librería IGP al añadir el código necesario para que la librería permita al proceso principal de la aplicación crear hebras.

Nombre	Duración	Fecha de inicio	Fecha de fin
1. Creación de la API para la librería IGP.	6 días	lun 12/02/18	dom 18/02/18
2. Búsqueda de distintas aplicaciones para Xeon Phi.	11 días	lun 19/02/18	lun 05/03/18
3. Integración de la librería IGP en aplicación.	20 días	mar 06/03/18	mar 03/04/18
4. Experimentación de aplicación con y sin IGP.	11 días	mié 04/04/18	mié 18/04/18
5. Modificación de la librería IGP.	18 días	jue 19/04/18	lun 14/05/18
6. Difusión del TFG.	35 días	mar 15/05/18	sáb 30/06/18

Tabla 1.1. Temporización en días.

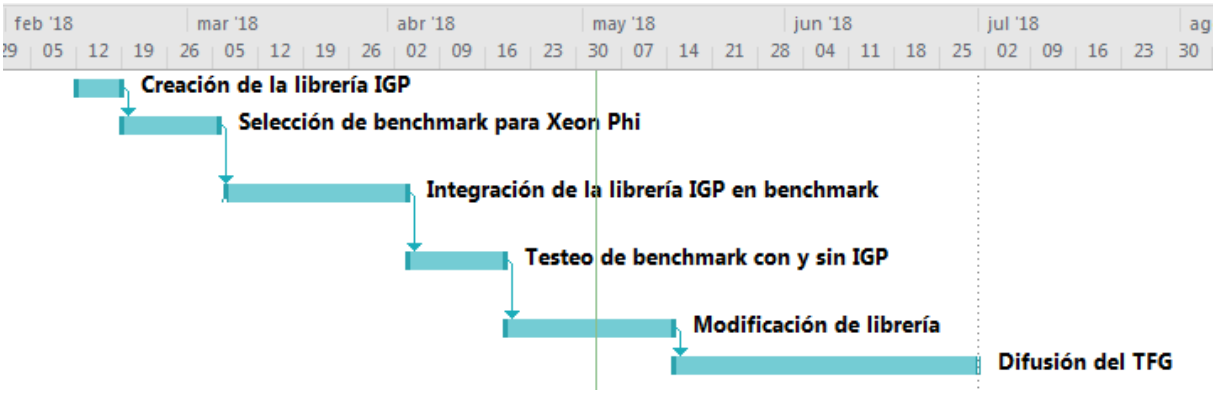

Figura 1.3. Cronograma.

1.3. Tecnologías utilizadas.

La realización del presente TFG ha requerido de la utilización de un servidor Dell PowerEdge T620 con procesador Intel Xeon E5-2609 v2 con 4 núcleos a 2,5 GHz de 64 bits y 10 MB de cache por procesador, junto con un coprocesador Intel Xeon Phi 3120A con 57 núcleos de 4 hilos/núcleo, lo que hace un total de 228 hilos a 1,1 GHz (1090 MHz) de 64 bits y una caché L2 de 512 KB en cada procesador, compartidas por todos los núcleos. Además de las instrucciones x86 de 64-bit, los núcleos ofrecen vectores SIMD de 512 bit, que mejoran la ejecución del código a través de vectorización. El coprocesador se conecta al host a través de un bus PCI Express y posee soporte de 8GB de memoria DDR5 [9].

Actualmente, el servidor dispone del sistema operativo Red Hat Enterprise 7.1 sobre un kernel de Linux 3.10.0, mientras que el coprocesador Intel Xeon Phi tiene instalado una Manycore Platform Software Stack (MPSS) versión 3.7.2, que contiene todo el software que se requiere para el funcionamiento de la Xeon Phi.

Intel Xeon Phi es un coprocesador que fue lanzado por Intel en 2012. Se llama coprocesador porque ofrece un servicio extra de computación al procesador, en este caso, puede ejecutar un sistema operativo Linux, también es capaz de utilizar todos los modelos de programación paralela como OpenMP, hebras POSIX u OpenCL.

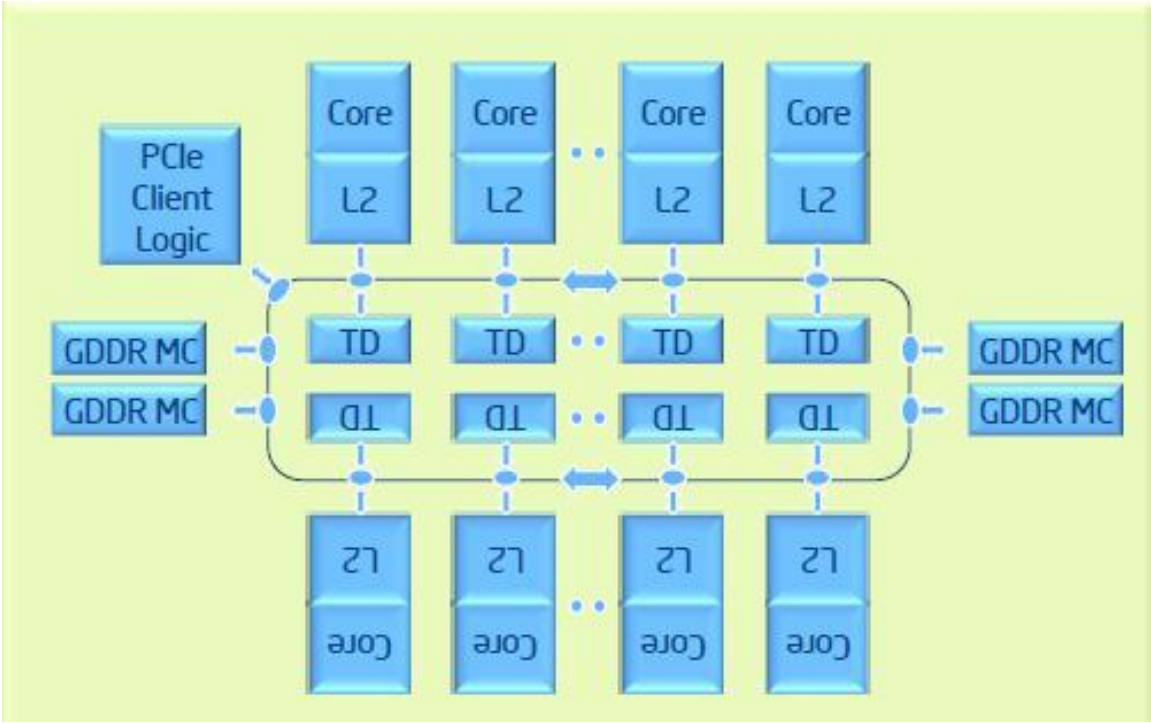

Figura 1.4. Microarquitectura del coprocesador Xeon Phi. Fuente: [8]

Los coprocesadores Intel Xeon Phi han sido diseñados para mejorar el rendimiento de aplicaciones que utilizan todos los recursos de los procesadores Intel Xeon. La mayoría de las aplicaciones no han sido estructuradas para explotar de manera óptima el paralelismo, por tanto, el rendimiento de tales aplicaciones se puede mejorar a través de los coprocesadores. Además, ofrecen la capacidad de hacer que un sistema que ofrece un rendimiento excepcional sea energéticamente eficiente.

La Figura 1.5 muestra los picos de rendimiento comparando el procesador Xeon y el coprocesador Xeon Phi. Los procesadores Xeon ofrecen un buen rendimiento para un amplio abanico de aplicaciones, pero alcanzan un límite. Para medir el rendimiento de sistemas con múltiples núcleos, se utiliza la eficiencia (E):

$$E = \frac{S(n)}{n}$$

Siendo S(n) el speedup o factor de mejora del rendimiento y n el número de hebras ejecutadas:

$$S(n) = \frac{T(1)}{T(n)}$$

Para que el coprocesador Xeon Phi ofrezca buen nivel de rendimiento es necesario un mayor nivel de paralelismo, y por ello se requiere que las aplicaciones multihebradas gestionen de manera eficiente las hebras. [11]

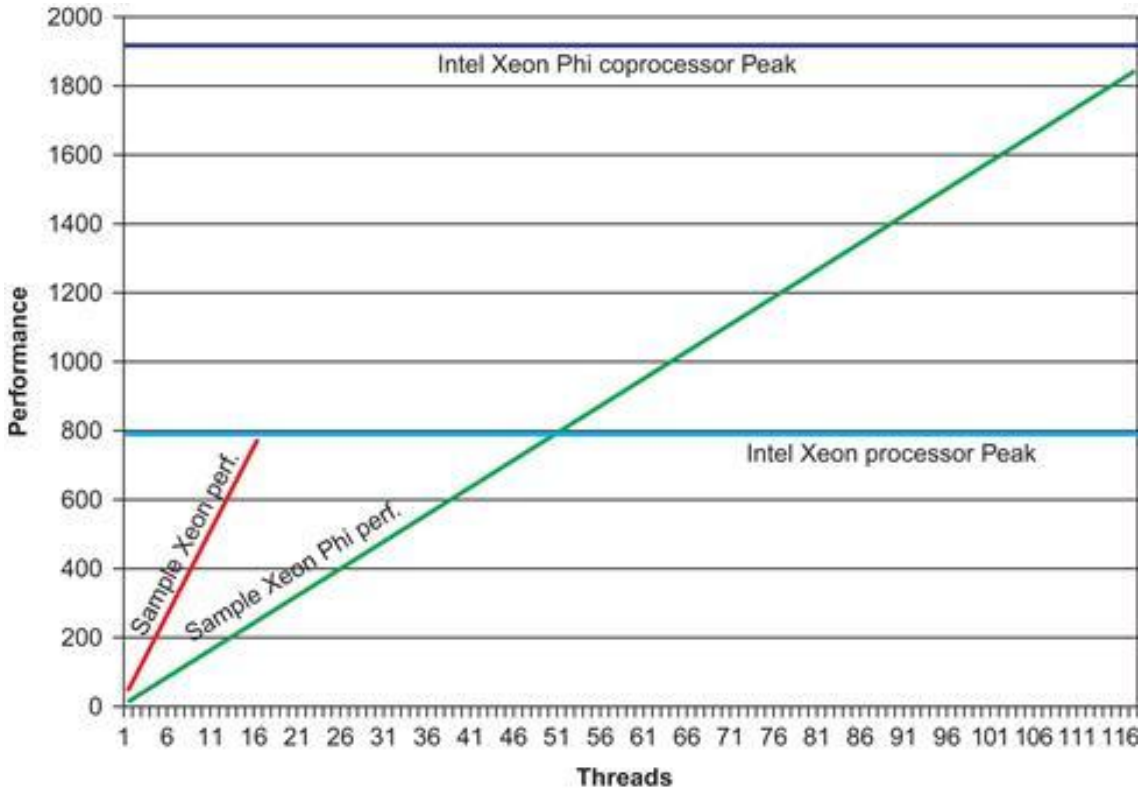

Figura 1.5. Comparativa de rendimiento entre Xeon y Xeon Phi. Fuente: [11]

1.3.1. Compilar y ejecutar aplicaciones en Xeon Phi.

Las interfaces binarias de aplicaciones (ABI) son convenciones entre artefactos de código fuente y definen los mecanismos a través de los cuáles son invocadas las funciones, como se pasan los parámetros y la forma en que son devueltos los valores, o como son implementadas las librerías, una ABI esta forzada por el enlazador (o linker) y en general es un estándar para que los códigos que no

están relacionados sean compatibles, también se encarga de que los procesos puedan coexistir en el mismo sistema, por ejemplo, en un sistema UNIX define como se ejecutan las señales. Las ABI del procesador y el coprocesador difieren entre sí, por este motivo los binarios del procesador no pueden ser ejecutados en el coprocesador, y se hace necesario compilar el código fuente para cada tipo de aplicación (Xeon y Xeon Phi). [10]

Existen dos maneras de ejecutar código paralelo en la Xeon Phi, la ejecución nativa o las extensiones offload del host:

1. La ejecución nativa se realiza en el propio coprocesador, ya que ejecuta un sistema operativo Linux, es posible hacer log en la Xeon Phi utilizando SSH a través de una interfaz de red mic0, añadida al kernel por un módulo proporcionado por Intel, y utilizarla de forma nativa. Por tanto, se pueden ejecutar los típicos comandos y aplicaciones de Linux.

2. Las extensiones offload permiten ejecutar secciones de código (escritas en C o Fortran) o una aplicación completa a la arquitectura de Intel `MIC`. Sin embargo, se obtiene un mejor rendimiento con aplicaciones altamente paralelas y que utilicen operaciones SIMD (operaciones con vectores). El compilador offload produce archivos binarios fat y ficheros .so que contienen código para el host y el coprocesador. Además, produce un código que examina el entorno de ejecución en tiempo real y busca la presencia de un coprocesador, el compilador offload producirá versiones del código para el host y el coprocesador de aquellas regiones que estén marcadas como offload. En resumen, el método offload se ejecuta en el host, y las directivas marcadas como offload se ejecutan en el coprocesador.

Por otra parte, existen varias diferencias entre utilizar el compilador de Intel (`icc`) y utilizar `gcc`. La diferencia más destacable es que al utilizar `icc` se utilizan instrucciones del propio coprocesador, por tanto el código generado se ejecutará en un número menor de ciclos. El compilador `gcc` se considera estándar para los sistemas operativos de UNIX, `gcc` decide qué compilador utilizar para cada archivo y después ejecuta un enlazador para producir un programa completo.

Para probar el funcionamiento de ambos compiladores se han ejecutado varios ejemplos utilizando `gcc` e `icc` en la Xeon Phi, con el objetivo de temporizar la ejecución con ambos compiladores y poder comparar la efectividad.

Se han realizado los siguientes pasos para compilar y ejecutar el código del fichero fuente `hello world` en la Xeon Phi desde la máquina anfitrión, esta aplicación muestra el mensaje *"Hello world"* por pantalla:

```
/opt/mpss/3.5.1/sysroots/x86_64-mpsssdk-linux/usr/bin/k1om-mpss-linux/k1om-mpss-
linux-gcc hello_world.c -o hello_world

scp hello_world mic0:

ssh mic0 /home/alberto/hello_world
```

Como resultado se mostraría *"Hello world"* en la consola de linux.

Para compilar y ejecutar el código del fichero fuente hello world en la Xeon Phi usando Intel C++ Compiler (`icc`) se han ejecutado los siguientes comandos:

```
su

source /opt/intel/parallel_studio_xe_2017.1.043/compilers_and_libraries_2017/linux/
pkg_bin/compilervars.sh intel64

icc -mmic hello_world.c -o hello_world

scp hello_world mic0:

ssh mic0 /home/alberto/hello_world
```

A continuación, se indican los pasos para realizar la compilación con el método offload. Esta instrucción sirve para configurar el entorno de desarrollo para que pueda ser utilizado con las herramientas de Intel.

```
icc -offload hello_offload.c -o hello_offload

./hello_offload
```

Como resultado se mostrará por pantalla el mensaje *"hello_world from offloaded code running on the coprocessor"*.

La tabla 1.2 muestra los tiempos medidos con el comando time para las compilaciones realizadas con `gcc` e `icc` (offload y no offload):

Compilador	Tiempo(s)
gcc	9.129
icc (no-offload)	4.592
icc (offload)	0.753

Tabla 1.2. Tiempos de compilación.

El tiempo de ejecución es menor si el código se compila con `icc` en modo offload, ya que el compilador utiliza instrucciones del propio procesador (ejecución nativa), y además en modo offload se ejecuta en el host pero las directivas marcadas como offload en el código fuente se ejecutan en el coprocesador. Aún así, aunque no se utilicen instrucciones marcadas con offload, el compilador `icc` permite generar un código más optimizado para la arquitectura, por lo que su tiempo de ejecución sigue siendo menor al código generado por el compilador `gcc`.

Por tanto, y tras este análisis previo, se decide utilizar el compilador `icc` para compilar la aplicación que posteriormente se presentará a lo largo del TFG.

1.4. Estructura del documento.

La presente memoria se ha estructurado en seis capítulos, de tal forma que permitan ilustrar las distintas etapas del TFG realizado:

- En el capítulo 2 se realiza una introducción a la secuenciación de ADN, que es el problema a resolver por la librería adaptativa. Se detallan los antecedentes que dieron paso a la secuenciación del genoma humano y los distintos métodos de secuenciación que tuvieron lugar a partir de 1977. A continuación, se explican algunos de los métodos de secuenciación más modernos.

- El capítulo 3 se centra en la API diseñada para la librería `IGP`, describiéndose en detalle su funcionamiento, los gestores disponibles a nivel de usuario, un gestor basado en el consumo energético, así como la publicación y la documentación de la librería.

- El capítulo 4 explica cómo funciona el mapeo del ADN y estudia las distintas herramientas de mapeo, explicando en detalle la herramienta `bwa` y el algoritmo de Burrows-Wheeler.

- En el capítulo 5 se explican los experimentos realizados, que justifican la ejecución de la herramienta `bwa` en un coprocesador, ya que el elevado nivel de paralelismo de la herramienta es idónea para su ejecución en el coprocesador Xeon Phi.

- En el capítulo 6 se exponen los resultados obtenidos al integrar la librería adaptativa `IGP` a la herramienta `bwa`.

Por último se exponen las conclusiones finales, y las futuras ampliaciones de la librería `IGP`.

Capítulo 2. La secuenciación de ADN.

La secuenciación de ADN (Ácido desoxirribonucleico) es el proceso que determina el orden de las bases nitrogenadas (adenina, citosina, timina y guanina) en una cadena de ADN, ya que los biólogos carecen de la tecnología para poder leer secuencialmente un genoma como si se tratase de un libro, en vez de ello, se dividen los fragmentos de ADN en trozos más pequeños llamados lecturas en un alfabeto formado por seis letras {A, T, C, G, N}, el símbolo N se utiliza para representar una ambigüedad.

El método para secuenciar ADN consiste tradicionalmente en extraer una pequeña muestra de sangre que contiene millones de células con ADN idéntico, utilizan métodos bioquímicos para dividir el ADN en trozos más pequeños y secuencian los fragmentos para producir lecturas, este proceso se muestra en la Figura 2.1. La dificultad es que los científicos no conocen la posición del genoma donde se encuentran dichas lecturas, por tanto deben solaparlas para reconstruir el genoma original, lo cuál se conoce como ensamblado de un genoma o ensamblado De Novo, se pueden generar de forma sencilla muchas lecturas, pero la dificultad está en realizar el ensamblado. [4]

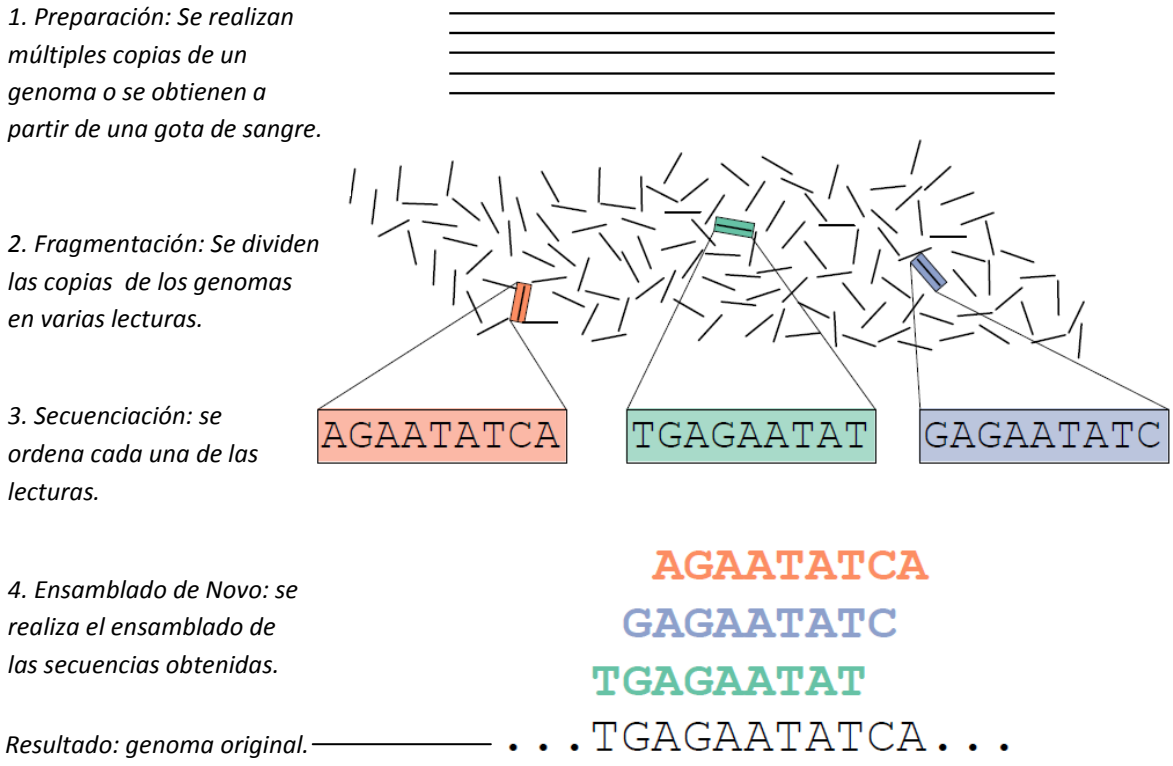

Figura 2.1. Secuenciación de ADN. Fuente: [4]

Por tanto, de manera general se establece el problema de la secuenciación como se muestra a continuación:

- Entradas.

- Reads: Un conjunto de lecturas procedentes de una máquina secuenciadora.

- Salida:

- Genome: El genoma original que componen las lecturas.

En los últimos años se ha conseguido una reducción en el coste de la secuenciación y un incremento en la velocidad, pudiendo secuenciar una gran cantidad de material genético y abriendo grandes oportunidades a la investigación de algunos tipos de cáncer, VIH, aumentando el conocimiento de la estructura celular y de muchos organismos, resistencia a las drogas o predisposición a ciertas enfermedades. El precio de secuenciar un genoma pronto bajará de los mil dólares (838,53€), una vez que caiga por debajo de este precio los investigadores podrán presentar tratamientos específicos para el genoma en particular de cada paciente, pudiendo diagnosticar enfermedades como el Alzheimer y la diabetes, en la Figura 2.2 se muestra como ha ido reduciéndose el coste de la secuenciación desde 2001. [1]

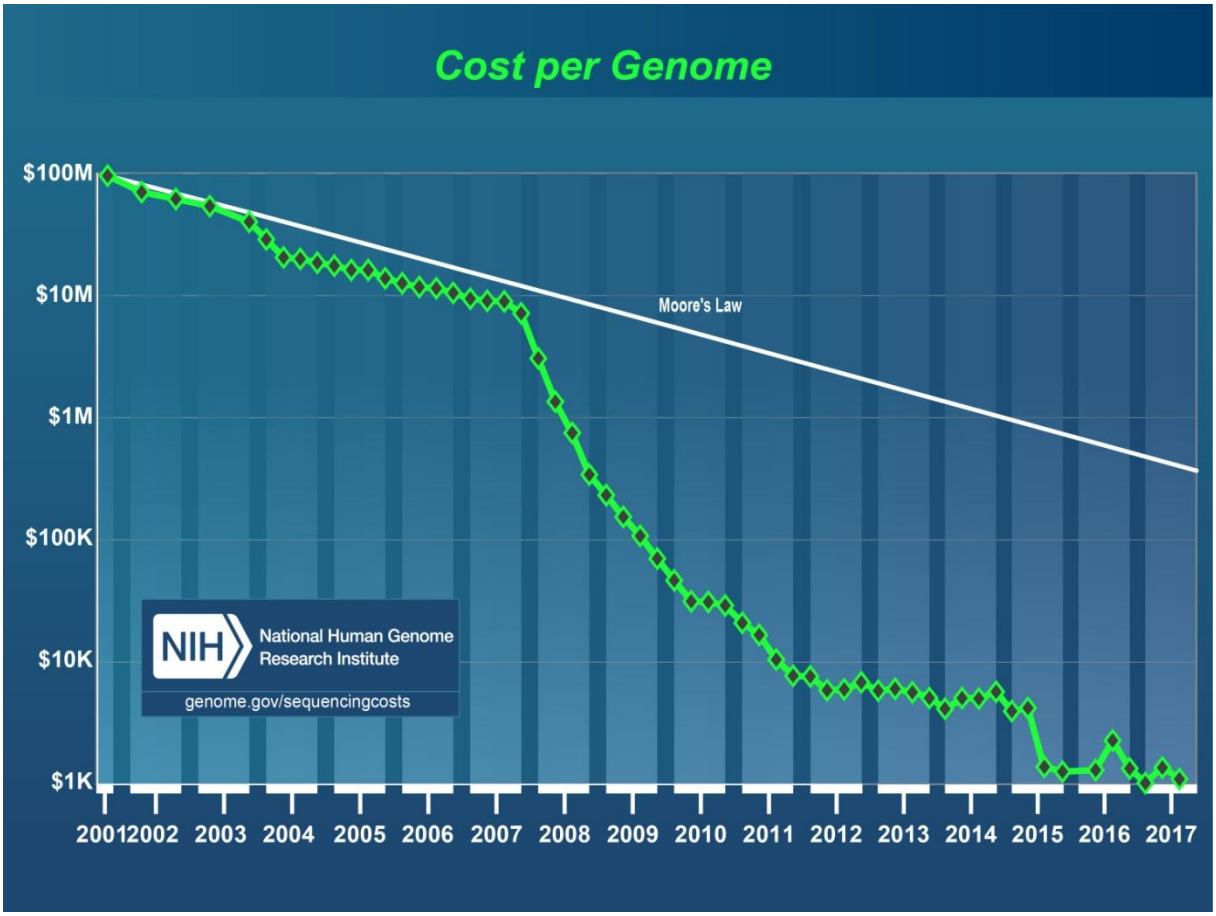

Figura 2.2. Coste de la secuenciación por genoma. Fuente: [13]

2.1. Antecedentes.

Las tecnologías más modernas de secuenciación son capaces de generar millones de lecturas en una sola pasada, sin embargo solo pueden secuenciar fragmentos pequeños de ADN (de entre 35 y 1100 nucleótidos), por lo tanto es necesario dividir la cadena, secuenciar y ensamblar las lecturas que se obtienen, para ello se necesitan programas que se ejecuten en un sistema multinúcleo y permitan realizar el ensamblado.

En 1977 fueron desarrollados los primeros métodos para secuenciar ADN, el primero, llamado método de secuenciación química fue desarrollado en febrero por Maxam y Gilbert de la Universidad de Harvard, y el segundo, llamado método de secuenciación de cadena, fue desarrollado en diciembre por Sanger et al de la Universidad de Cambridge [6, 12].

La técnica desarrollada por Sanger utilizaba menos productos químicos tóxicos e isotopos radioactivos que la técnica de Maxam y Gilbert, como resultado se convirtió en el método predominante durante los siguientes treinta años.

La demanda de un mayor rendimiento conllevó a un paralelismo en los procesos y a una mayor automatización de los laboratorios. Gracias a estos avances la técnica de Sanger hizo posible la secuenciación del genoma humano en 2003.

En 1987 Applied Biosystems introdujo su primera máquina de secuenciación automática, llamada ABI370, adoptando la técnica de electroforesis capilar, la cual incrementaba la velocidad y la precisión del proceso, la ABI370 podía detectar 96 nucleótidos simultáneamente, 500000 nucleótidos diarios, con una longitud de lectura de 600 nucleótidos.

Sin embargo, el genoma humano requirió una alta cantidad de recursos y tiempo ya que está compuesto por 3 billones de nucleótidos y no existían algoritmos de ensamblado lo suficientemente eficientes, se necesitaba tecnología de alto rendimiento, rápida y de bajo coste. Por este motivo, en el año 2004 el Instituto Nacional de Investigación del Genoma Humano (National Human Genome Research Institute, NHGRI) inició un programa de fondos con el objetivo de reducir el coste de la secuenciación del genoma humano a mil dólares en diez años. Por ello, se estimuló el desarrollo y la comercialización de las tecnologías de secuenciación de la próxima generación (Next Generation Sequencing, NGS) en 2005 como contrapartida al método de Sanger, que es considerado un método de primera generación.

2.2. Estado del arte.

El proyecto genoma humano estimuló el desarrollo de nuevos instrumentos de secuenciación para incrementar la precisión y velocidad reduciendo el coste y la mano de obra, en 2005 surgieron las tecnologías de siguiente generación (NGS), las cuales se distinguen del método de Sanger en que se pueden secuenciar en paralelo de millones a billones de lecturas en una sola pasada y el tiempo que se requiere para generar las lecturas de gigas de bases es de solamente días o incluso horas.

Por ejemplo, los 3 billones de pares de bases que posee el genoma humano están distribuidos en varias macromoléculas que varían en tamaño (de 33 a ~247 millones de pares de bp) y se reparten entre los 23 cromosomas, mediante el método de Sanger se tardó 15 años en secuenciar este genoma con un coste de 100 millones de dólares, mientras que utilizando el secuenciador NGS 454 FLX se tardó 2 meses con un coste de 1 millón de dólares.

Las tecnologías NGS continúan mejorando y están divididas en dos tipos:

- Las tecnologías de 2ª generación son aquellas que aparecieron después de la 1ª generación, y se distinguen porque necesitan preparar bancos de ADN antes de comenzar el proceso de secuenciación.
- Las tecnologías de 3ª generación han aparecido recientemente y se distinguen porque no necesitan realizar una preparación de los bancos de ADN y son capaces de generar lecturas más largas a un menor coste.

En el 2005, Life Sciences lanza el secuenciador 454, un año después Solexa desarrolla el secuenciador Genome Analyzer (actualmente HiSeq), a continuación la empresa Agencourt lanza SOLID, Complete Genomics lanza CGA y posteriormente Pacific Biosciences lanza su sistema PacBio RS, estos sistemas son representativos de secuenciación paralela. Posteriormente algunas compañías fueron compradas por otras: Applied Biosystems compró Agencourt en 2006, Roche compró 454 y Solexa fue comprada por Illumina.

El método NGS más utilizado se conoce como secuenciación Shotgun, mediante esta técnica la cadena de ADN se clona mediante una bacteria, el número de copias se conoce como cobertura, la muestra resultante se divide aleatoriamente en pequeños fragmentos y se secuencia desordenadamente utilizando alguna de las tecnologías vistas anteriormente. A través de la división aleatoria se crean fragmentos de diferente tamaño, por lo tanto es necesario elegir las de un tamaño apropiado para la tecnología de secuenciación elegida, por último se ensamblan los fragmentos mediante técnicas de computación. El objetivo es reconstruir la cadena original a partir de los millones de lecturas que se obtienen mediante NGS. La Figura 2.3 representa los pasos del método Shotgun o de escopeta [12].

Otro método de secuenciación muy relevante es el método de Illumina/Solexa, que utiliza un láser con una señal luminosa específica para cada nucleótido, una vez se emite la señal luminosa los distintos nucleótidos, que se encuentran en varios conjuntos (llamados clústeres) emiten varios colores, uno por cada tipo de nucleótido (A, C, T y G), los programas de ordenador se encargan de traducir las señales luminosas en las diferentes cadenas de ADN.

La secuenciación de PacBio Sciences utiliza el mismo proceso que Illumina pero no requiere de una preparación previa de los clústeres, sino que se los nucleótidos se van coloreando a la vez que se va evaluando cada uno en tiempo real.

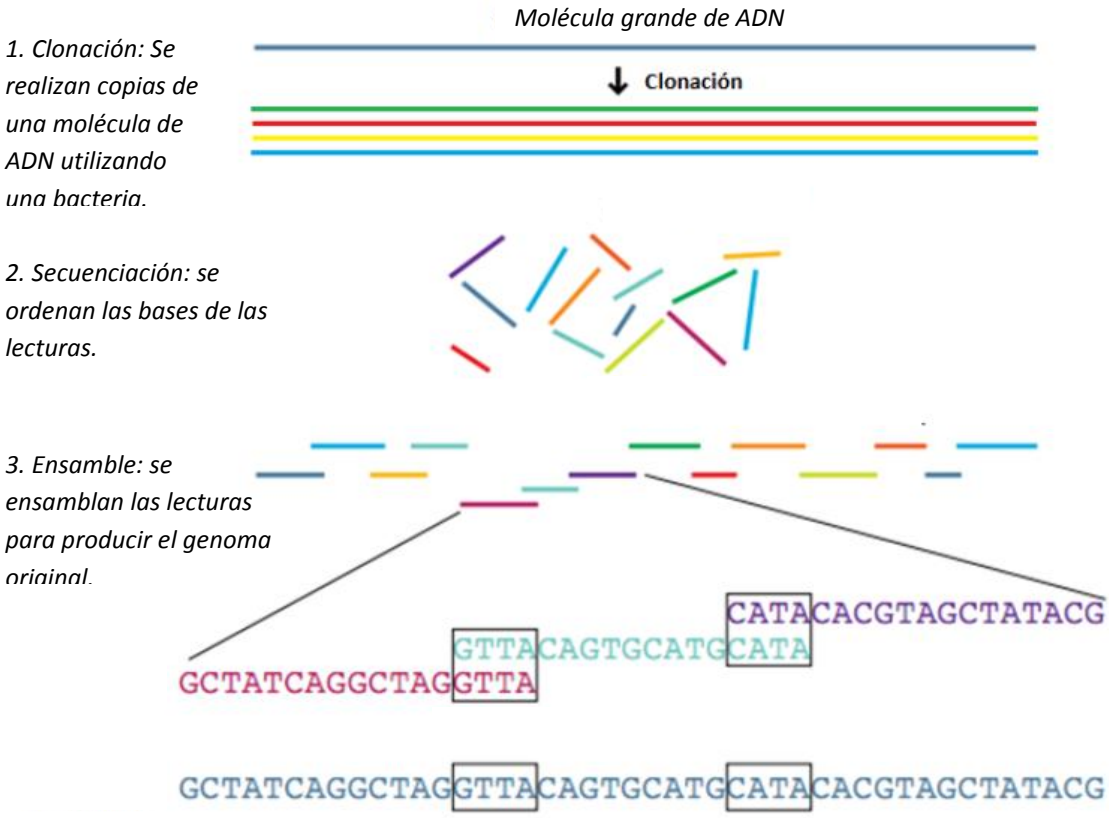

Figura 2.3. Secuenciación Shotgun. Fuente: [1]

En la tabla 2.1 se muestra las principales características de los sistemas de secuenciación NGS, destacando el número de lecturas por ejecución (pasada) de la secuenciadora, la tasa de error, el tiempo que tarda cada ejecución así como la cantidad de datos generada por pasada.

Plataforma	Instrumento	Lecturas/pasada	Datos generados/pasada	Tasa de error (%)	Año	Tiempo de pasada
			1ª generación			
Sanger ABI 3730XL	3730xl	96	0,00069 - 0,0021	0,3	2002	3h
			2ª generación			
454	GS20	200	0,02 Gb	1	2005	23h
454	GS FLX	400	0,1 Gb	1	2007	25h
Illumina	HiSeq 2500	5×10^{12}	1,5 Tb	0,1	2012	3-10 días
SOLiD	5500xl W	6×10^{12}	320 Gb	0,1	2013	7-14 días
			3ª generación			
PacBio	RS	432	0,5-1 Gb	15	2014	0,5-2h

Tabla 2.1. Características de los sistemas de secuenciación NGS. Fuente: [1]

Este repaso de las distintas tecnologías utilizadas en el secuenciamiento del ADN nos permite resaltar que algunas de las tecnologías superan el Terabyte de datos por pasada, no obstante la longitud de cada lectura sigue estando limitada a 1000 nucleótidos.

En general las lecturas que se obtienen de los instrumentos de secuenciación de cualquier generación son demasiado cortas, esto motivó el desarrollo de métodos que permitieran secuenciar lecturas más largas de ADN, incluso de genomas completos (*WGS*, del inglés Whole Genome Sequencing).

La Figura 2.4 representa las mejoras respecto a la carga de trabajo y la longitud de lectura de las tecnologías de tercera generación.

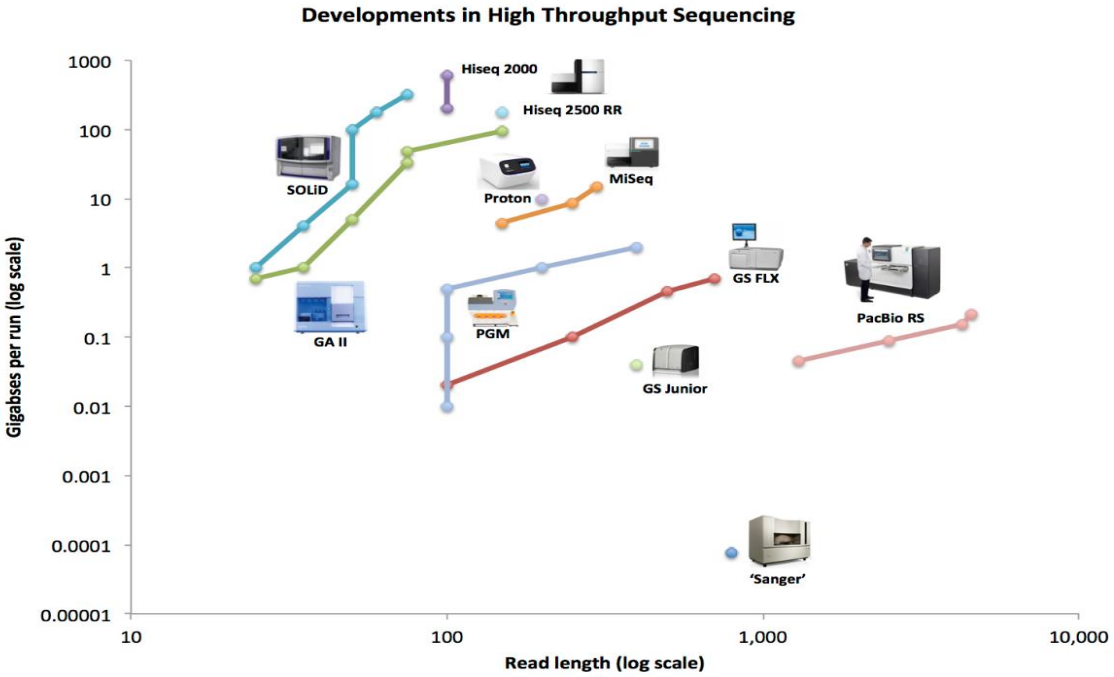

Figura 2.4. Mejoras en la carga de trabajo de las tecnologías NGS. Fuente: [14]

La máquina PacBio RS genera lecturas de mayor tamaño y una menor cantidad por pasada, no obstante se siguen necesitando herramientas computacionales capaces de procesar todas las lecturas.

La reconstrucción se puede llevar a cabo mediante dos formas:

1. Utilizando un genoma de referencia secuenciado previamente, esta técnica se denomina alineación o mapeo. En el apartado 4.1 se explica en detalle el método de alineación.
2. A partir de las lecturas secuenciadas, esta técnica se denomina ensamble De Novo.

Los principales ficheros de salida obtenidos en cada pasada de NGS son los ficheros FASTQ, es un formato de fichero en texto plano que contiene los resultados del secuenciador en forma de datos crudos (raw data), cada plataforma extrae los datos de formas distintas, por ejemplo Ion Torrent obtiene los datos a través de electroquímica (iones de hidrogeno) [3].

Este fichero se compone de secuencias de cuatro líneas:

- *Título*: Normalmente es el identificador de secuencia encabezado por el carácter `@` y no tiene límite de longitud.

- *Secuencia biológica*: Los espacios y tabulaciones no se permiten y está compuesta por la combinación de los nucleótidos (A, C, T y G), si existe una ambigüedad el secuenciador asigna el carácter N.
- *Indicador de fin de secuencia*: El carácter `+`.
- *Valores de calidad*: Un conjunto de caracteres ASCII con la misma longitud que la secuencia biológica.

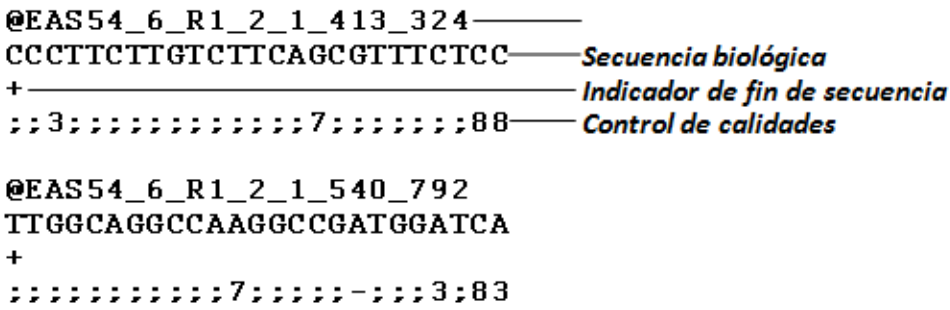

Figura 2.5. Ejemplo de fichero FASTQ.

En la figura 2.5 se aprecia un ejemplo del fichero FASTQ que representa la entrada estándar utilizada por las aplicaciones software de alineación.

Capítulo 3. Utilización de la librería IGP.

La librería IGP (Interfaz de Gestor de Paralelismo) se utiliza para gestionar de manera más eficiente las aplicaciones multihebradas y se compone de distintos métodos que ayudan al programador a su integración en una aplicación que utilice la interfaz multihilo de bajo nivel POSIX threads (pthreads), el uso de este esquema permite tener un mayor control sobre cada hebra ya que se puede elegir cuales lanzar al tener controlado el identificador de cada una de ellas, al contrario que la interfaz MPI (Message Passing Interface), en el cuál se lanzan todas las hebras y una barrera se encarga de que finalicen pero no nos permite controlarlas individualmente. En los siguientes apartados se explicará cómo se utiliza IGP y el funcionamiento de los distintos criterios de decisión del gestor de paralelismo.

En la Figura 3.1 se muestra la estructura de la librería.

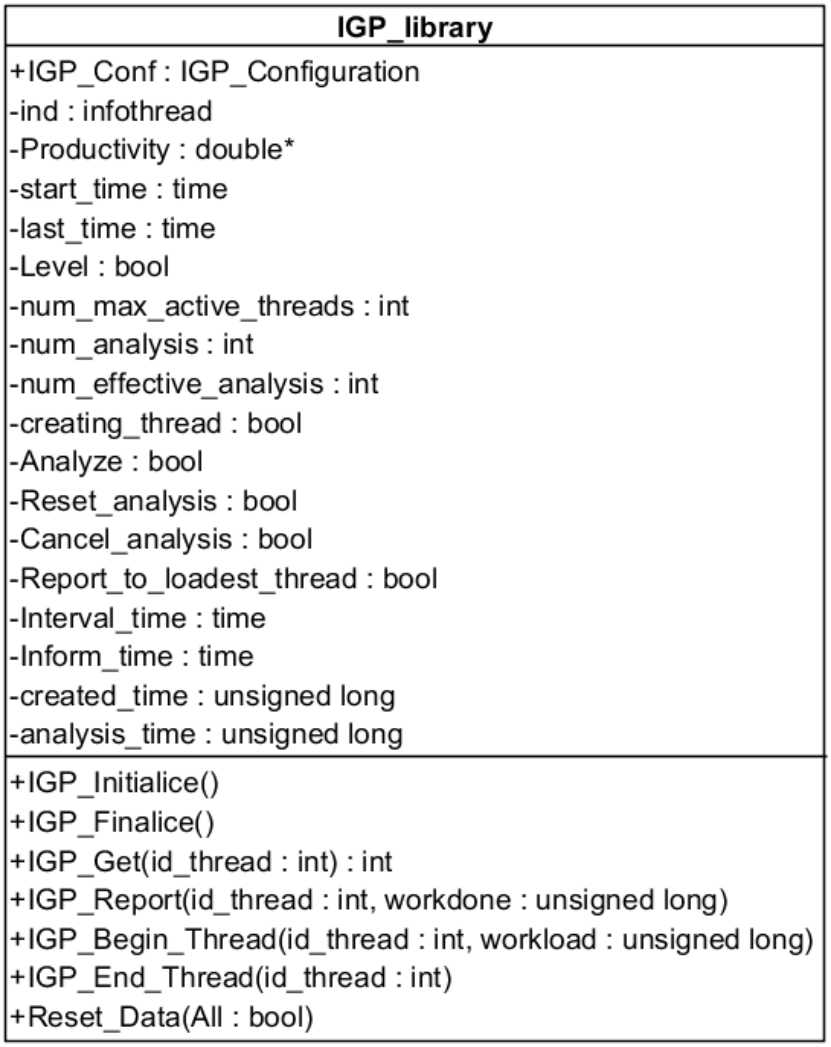

Figura 3.1. Diseño de IGP.

3.1. Funcionamiento.

A continuación se explicará cómo se realizan llamadas a la librería desde cualquier aplicación. Es necesario que el código este escrito en C/C++ y que la implementación se adhiera al estándar POSIX (Portable Operating System Interface).

Los pthreads o hilos POSIX se definen como un conjunto de tipos y llamadas a procedimientos implementados con una cabecera `pthread.h` y una librería, han sido desarrollados por el IEEE [2].

La principal ventaja que ofrecen los pthreads con respecto a otros esquemas multihebra (como OpenMP) es que se puede mejorar el rendimiento de la aplicación. Mientras que OpenMP necesita tener al menos una copia en memoria de sus operaciones, pthreads no necesita esta copia ya que las hebras comparten un espacio de memoria común. Además este esquema nos permite tener un mayor control de las hebras. Una de las principales desventajas es que al implementar operaciones de bajo nivel la programación con pthreads es más complicada que si se utilizasen otros esquemas.

En aquellos ficheros fuente en los que se vaya a utilizar la librería es necesario importar la cabecera mediante la sentencia `#include "IGP_library.h"`. En la Figura 3.4 se muestra el comportamiento de la librería en cualquier aplicación multihebrada, que se divide en tres fases: fase de inicialización, fase de evaluación y fase de finalización.

3.2. Compilación de la aplicación con la librería IGP.

Para instalar la librería es necesario tener instalado `Intel C++ Compiler`, ya que se utilizará para realizar la compilación de la librería.

Primero, se añadirán las siguientes sentencias al Makefile de la aplicación:

```
icc -mmic -c -o IGP_library.o IGP_library.c

ar rcs libIGP.a IGP_library.o
```

En la Figura 3.2 se muestra un ejemplo de Makefile con el objetivo `lib` el cual nos permite construir la librería mediante el comando `make lib`.

```
ALL_PROGRAMS = ray_tracing_1 ray_tracing

all: $(ALL_PROGRAMS)
	@echo
	@echo
	@echo $(ALL_PROGRAMS) "is (now) up-to-date"
	@echo
	@echo

clean:
	rm -rf $(ALL_PROGRAMS) $(OBJS) *~ IGP_library.o libIGP.a IGP_library_threadalloc.o
lib:
	icc -mmic -c -o IGP_library.o IGP_library.c
	ar rcs libIGP.a IGP_library.o
	icc -mmic -c -o IGP_library_threadalloc.o IGP_library.c -DUSE_THREAD_ALLOC
	ar rcs libIGP_threadalloc.a IGP_library_threadalloc.o
```

Figura 3.2. Ejemplo de Makefile.

Mediante el comando `icc` se compila IGP_library.c a un fichero objeto (con extensión .o) llamado IGP_library.o, el parámetro `mmic` permite que la librería se puede ejecutar en la Xeon Phi.

El comando `ar` permite empaquetar el fichero objeto generado en un único archivo llamado `libIGP.a`, que se utilizará posteriormente para compilarlo junto con el programa principal y poder realizar llamadas a las funciones de la librería. Por convenio los nombres de todas las librerías estáticas comienzan por `lib` y tienen .a por extensión.

Una vez generado el fichero `libIGP.a`, para integrar la librería junto con cualquier programa es necesario indicar el directorio donde se encuentra dicho fichero y cuál es su nombre mediante el siguiente comando:

```
icc -mmic -lpthread aplicacion.c IGP_library.c -o ejecutable_aplicacion
```

La Figura 3.2 muestra un ejemplo de integración de la librería en la aplicación *bwa*.

```
bwa:libbwa.a $(AOBJS) main.o
		$(CC) $(CFLAGS) $(DFLAGS) IGP_library.c $(AOBJS) main.o -o $@
		-Wl,-rpath,/opt/intel/composer_xe_2015.7.235/compiler/lib/mic -L. -lbwa $(LIBS)
```

Figura 3.3. Integración con IGP.

En este caso no se especifica el nombre del ejecutable, por defecto se llama *bwa*.

La variable `LIBS` almacena los enlaces a las librerías que se utilizarán en la aplicación, en este caso la librería `IGP` se encuentra en el directorio actual (.), indicándose a través del parámetro `L.`

```
LIBS = -lm -lz -L/home/zlib/lib -lpthread -lIGP -L.
```

A través del parámetro `lpthread` se enlaza con la librería pthreads, a continuación se explican las ventajas de utilizar pthreads sobre otros esquemas.

La Figura 3.4 muestra las distintas fases que componen `IGP` (fase de inicialización, fase de evaluación y fase de finalización), así como la interacción entre una aplicación real y la librería. También se detallan las distintas llamadas a las funciones así como el lanzamiento de las hebras.

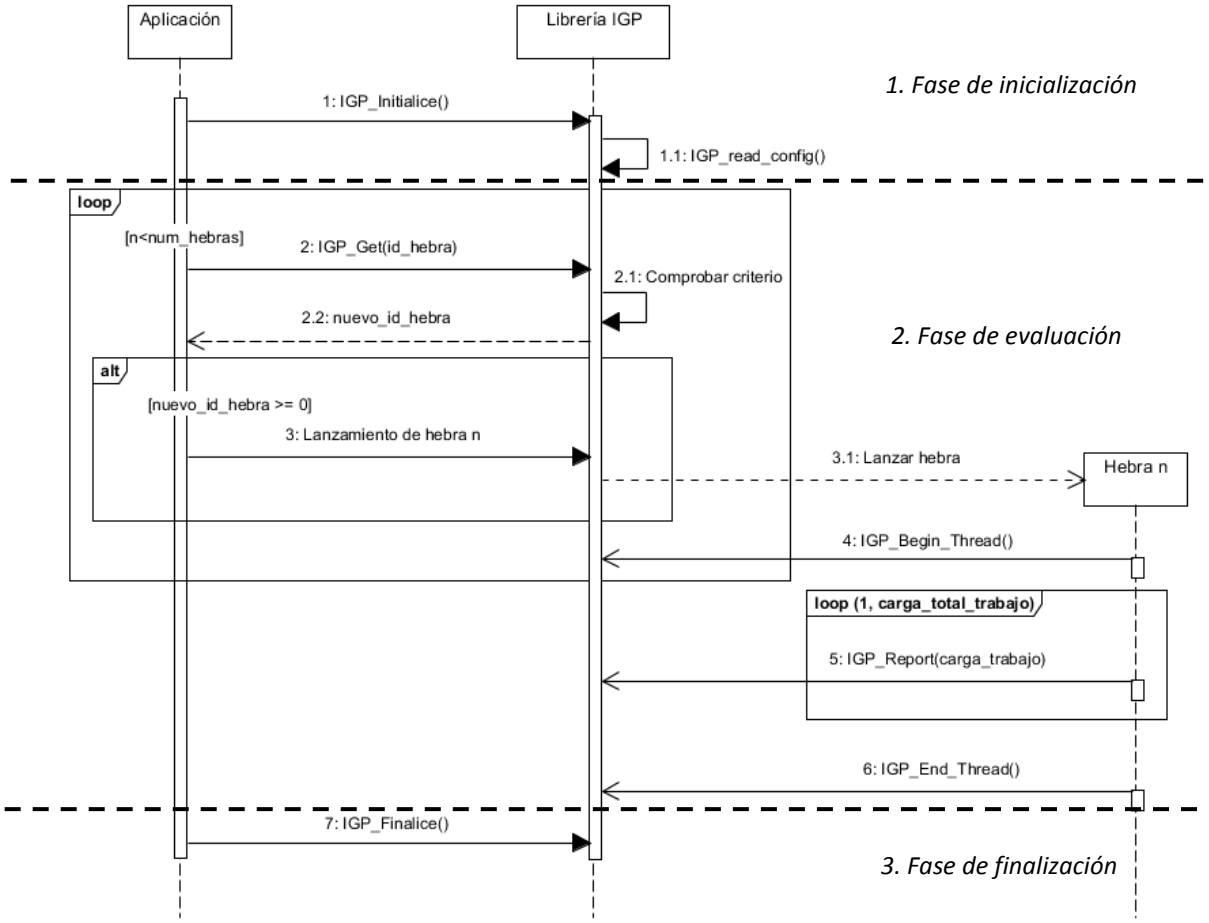

Figura 3.4. Comportamiento general de IGP.

1. Fase de inicialización.

En primer lugar se realiza una llamada a *IGP_Initialice*, que se encarga de inicializar los parámetros de configuración de la librería leyéndolos directamente de un fichero y se informa al GP de que va a ser gestionado.

En la Figura 3.5 se muestra el fichero de configuración con los valores por defecto indicando donde se encuentra cada parámetro, estos se pueden establecer en cualquier orden y si faltase alguno la librería lanzaría un error, también es necesario que los parámetros tengan los valores correctos, ya que los valores enteros y flotantes se encuentran en un intervalo. Los valores de los parámetros Timing, Detention, Process_is_a_thread y Maxload pueden tener los valores *yes* o *no*.

```
//                                          Configuration file of the IGP library.
//                                         IGP (Interfaz del Gestor de Paralelismo)
// The method used by parallelism manager to decide when multithreaded application has to create a new thread.
// The following methods have been built, where the parallelism manager decides to create news threads,
// METHOD / APPROACH: NON ADAPTIVE METHOD
//   0    /    0    : if total number of active threads is less than MAX_THREADS.
// METHOD / APPROACH: ADAPTIVE METHODS AT APPLICATION LEVEL
//   1    /    0    : if the application performance with n threads measured by parallelism manager in last time between the
//                  : application performance with n-1 threads is better than a THRESHOLD chosen and total number of active
//                  : threads is less than MAX_THREADS. The THRESHOLD value is between 0 and 1.
//   2    /    0    : if last PERIOD between idle average time of all cpus in this interval time is higher than THRESHOLD
//                  : chosen and total number of active threads is less than MAX_THREADS. The idle time of cpus are reading
//                  : from /proc/stat file. The THRESHOLD value is between 0 and 100.
//   2    /    1    : if sleeping time of all threads in the last PERIOD is null and total number of active threads is less than
//                  : MAX_THREADS. The sleeping time of threads are reading form /proc/[id_pid]/task/[id_tid]/stat files.
//   2    /    2    : if Phi temperature less fan-in temperature doesn`t increase a THRESHOLD chosen and total number
//                  : of active threads is less than MAX_THREADS. The Phi and fan-in temperatures are reading from /sys/class/
//                  : micras/temp file. The THRESHOLD value is between 0 and 100.
//   2    /    3    : if Phi power running multithreaded application less phi power in idle state doesn't increase a THRESHOLD
//                  : chosen and total number of active threads is less than MAX_THREADS. The Phi power is reading from /sys/
//                  : class/micras/temp file. The THRESHOLD value is between 0 and 100.
//   3    /    0    : if the application power is under a POWER_THRESHOLD and the total_work hasn't decreased
//                  : The THRESHOLD value is between 0 and 1.
// METHOD / APPROACH: ADAPTIVE METHODS AT KERNEL LEVEL
// 3,4,5  /   0-5   : Kernel-level methods.
METHOD=3
APROACH=0
// The Max_Threads is a maximum number of running threads at the same time.
MAX_THREADS=256
// Threshold used in application performance method.
THRESHOLD=0.5
// Period is a time interval (ms) between two consecutive samples to read the manager decision
PERIOD=0.00
// Manager shows times and information by console.
TIMING=yes
// Manager sleeps running threads if it´s neccesary.
DETENTION=no
// Main process resolves workload like a thread
PROCESS_IS_A_THREAD=no
// Main process uses loadmax as approach or the first thread
MAXLOAD=yes
```

Figura 3.5. Fichero de configuración de IGP.

Los parámetros de configuración de la librería son los siguientes:

- *Max_Threads*: Se trata del número máximo de hebras que va a gestionar la librería. Se puede seleccionar un número de hebras en el rango de 1 a 1000.

- *Method*: Indica el GP a utilizar por el planificador de hebras de la librería para decidir si crea o no una nueva hebra, posteriormente se explicarán en detalle cada uno de los métodos. Los valores permitidos oscilan en el rango de 0 a 1, el valor 0 permite seleccionar el método no adaptable y el valor 1 permite escoger el gestor ACW.

- *Work_Threshold*: Establece un umbral mínimo de eficiencia que se utiliza para decidir si se lanza una hebra cuando la productividad en un intervalo de tiempo supera dicho umbral. Se puede seleccionar una carga de trabajo de 0 a 300.

- *Timing*: Sirve para que el planificador muestre información sobre varios instantes de tiempo, en concreto el instante en que se inicia y finaliza el tiempo total de ejecución de la librería, el instante en que comienza y finaliza un análisis, y los instantes de tiempo en que comienzan y finalizan las hebras.

- *Period*: Establece un intervalo de tiempo (ms) entre dos análisis consecutivos del GP, oscila en el rango de 0 a 10000 (10s).

- *Detention*: Permite la detención temporal de una hebra en un momento determinado. Puede tomar los valores *yes* o *no*.

- *Process_is_a_thread*: Permite decidir entre utilizar un esquema en el que un proceso crea hebras o un esquema en el que las hebras se dividen unas a otras.

2. Fase de evaluación.

Una vez inicializada la librería las hebras informan periódicamente de la carga de trabajo y de su comienzo mediante *IGP_Begin_Thread* o finalización mediante *IGP_End_Thread*.

A través de la llamada a *IGP_Get* se decide si se crea o no una hebra basándose en un criterio de decisión, los cuales se explican en detalle en el apartado 3.2.

El algoritmo 3.1.2.1 muestra el esqueleto de una aplicación utilizando la librería *IGP* y destacando el funcionamiento de *IGP_Get*.

Algoritmo 3.1.2.1 : IGP_Get.

```
(1) include "IGP_library.h"
(2) func main(){
(3)    IGP_Initialice();                                  1. Fase de inicialización
(4)    int i=0, num_threads=500, id_nueva_hebra=0;
(5)    while (i < num_threads){
(6)           id_nueva_hebra = IGP_Get(0);                2. Fase de evaluación
(7)           if (id_nueva_hebra >= 0){
(8)                  i++;
(9)                  rc = pthread_create(&tid[i], &attr, worker, data+i);
(10)          }
(11)   }
(12)  for (i=0; i < num_threads; i++){                     Espera a que todas las hebras
                                                           terminen de ejecutarse
(13)          pthread_join(i);
(14)  }
(15)   IGP_Finalice();                                     3. Fase de finalización
```

El algoritmo 3.1.2.2 se centra en la función que ejecuta cada hebra en particular haciendo énfasis en las funciones *IGP_Begin_Thread*, *IGP_End_Thread* e *IGP_Report*.

Algoritmo 3.1.2.2 : calculate_thread.

```
(1) func calculate_thread(){                          2. Fase de evaluación
(2)     IGP_Begin_Thread(id, total_workload);
(3)     for (i=0; i < total_workload; i++){
(4)             IGP_Report(1);                        Se informa al GP de que la hebra ha ejecutado 1 unidad de trabajo
(5)             thread_calculate();                   La hebra realiza sus cálculos
(6)     }
(7)     IGP_End_Thread(id);
```

Se realiza a continuación una explicación más detallada de las funciones mostradas:

- `IGP_Begin_Thread(id, total_workload)`: Esta función se encarga de informar al GP de que la aplicación acaba de crear una hebra con el identificador *id* y con una carga total de trabajo inicial *total_workload*.

- `IGP_Report(workload)`: Esta función informa al GP de que la hebra está ejecutando la carga de trabajo *workload*.

- `IGP_End_Thread(id)`: Informa al GP de que la hebra *id* ha terminado de ejecutarse.

3. Fase de finalización.

Una vez que todas las hebras han finalizado su ejecución la aplicación informa al *GP* de que ha finalizado mediante la llamada a `IGP_Finalice` y el GP muestra los resultados de las distintas estadísticas:

- *Analysis time*: Tiempo total de los análisis realizados por la librería.

- *Executions of the decision model*: Número de evaluaciones realizadas.

- *Evaluations of the decision rule*: Número de veces que el gestor ha decidido si crear o no una hebra.

- *Positive analysis*: Número de veces que el gestor ha decidido crear una hebra.

- *Maximum number of threads*: El máximo número de hebras activas durante la ejecución de la aplicación.

3.3. Gestores de paralelismo.

Para tomar la decisión de crear una hebra la librería `IGP` se basa en dos gestores de decisión, los cuales chequea continuamente a través de `IGP_Get`:

- **NONADAPTABLE (No adaptable).**

Este gestor se basa en el valor del parámetro MAX_THREADS, que es el número máximo de hebras que se permiten crear simultáneamente, suponiendo que el usuario lanza un número de hebras mayor al número de núcleos, se produciría concurrencia en los núcleos, por lo que las hebras podrían entrar en conflicto entre sí al compartir los recursos, por ello se necesita un mecanismo que controle las hebras haciendo que no sobrepasen el número de núcleos. Para ello el gestor evalúa continuamente que se cumpla la siguiente condición:

$$n < Nº\ máximo\ de\ hebras$$

El número de hebras activas (n) en un instante de tiempo no debe sobrepasar el número máximo de hebras que se establece en la librería.

- **ACW (Method decides based on work).**

Este método decide si crear hebras evaluando el rendimiento de la aplicación, comparando la productividad de n hebras con la obtenida con n-1 hebras. La productividad *Pr* de la hebra actual se representa como la media de la carga total de trabajo en un intervalo de tiempo *dif_time*:

$$\Pr[n-1] = \frac{total\_work}{dif_{time} \times n}$$

- *total_work*: Representa el trabajo realizado por todas las hebras en cada intervalo de tiempo.

- *dif_time*: Representa un intervalo de tiempo entre la creación/finalización de una hebra y la siguiente.

- *n*: El número total de hebras activas en el intervalo *dif_time*.

Y la eficiencia *E* se calcula comparando la productividad actual con la productividad obtenida anteriormente con una hebra menos:

$$E = \frac{\Pr[n-2]}{\Pr[n-1]}$$

Por tanto, el gestor evalúa si la eficiencia no supera un umbral *Work_Threshold* establecido por el usuario en el fichero de configuración mediante la siguiente condición.

$$E > WORK\_THRESHOLD$$

La primera hebra siempre se lanzará, y a partir de la segunda comenzará a compararse la productividad.

La figura 3.7 representa un array de tamaño N hebras en el que cada elemento representa la productividad de n hebras.

Pr_1	Pr_2	Pr_3	Pr_4	Pr_5	Pr_6	Pr_7	Pr_8	...	Pr_N

La eficiencia se calcula a partir de 2 hebras, como Pr_{n-2}/Pr_{n-1}

Figura 3.6. Productividad de n hebras.

Capítulo 4. El mapeo del ADN.

El desarrollo de las tecnologías NGS ha hecho posible el surgimiento de una gran cantidad de software de alineación, no se puede decir que un programa sea mejor que otro, sino que estos programas muestran características particulares que los adecúan a una plataforma en particular (Illumina, 454, Roche, etc.) o a un tipo de dato específico (ADN, ARN, bisulfito, etc.). También se distinguen en la forma en que informan de los resultados de la alineación, las inserciones y supresiones permitidas (indels), si se permiten indels consecutivos, si indexan el genoma de referencia o no o la capacidad de mapear pares de bases [4]. La tabla 4.2.1 muestra las características de los programas de alineación más representativos actualmente. La columna 2 representa las plataformas de secuenciación que soporta el alineador: Illumina (I), ABI Solid (So), Roche 454 (4), ABI Sanger (Sa), Helicos (Hel), Ion Torrent (Ion) y PacBio(P).

Programa de alineación	Plataforma de secuenciación	Longitud de lectura min/max (nucleótidos)	Inserciones y borrados permitidos	Indels consecutivos	Bases simples	Pares de bases
Bfast	I, So, 4, Hel	11/--	S	S	N	S
Bowtie	I, So, 4, Sa, P	4/1K	S	N	S	S
Bwa	I, So, 4, Sa, P	4/200	8	S	S	S
GASSST	I, So, 4, Sa, P	50/500	S	N	-	-
Gmap	I, 4, Sa, Hel, Ion, P	8/--	S	S	N	N
Maq	I, So	8/63	S	N	S	S
Novoaling	I, So, 4, Ion, P	30/300	2	N	S	S
Pass	I, So, 4	23/1K	S	S	S	S
Rmap	I, So, 4	11/10K	0	N	S	S
Seqmap	I	15/500	3	N	N	N
Shrimp2	I, So, 4	30/1K	S	N	N	S
Soap	I	7/60	3	N	N	S
Soap2	I	27/1K	0	S	N	S
Ssaha2	I, 4, Sa	15/48K	S	N	N	S
Zoom	I, So, 4	12/240	S	N	S	S

Tabla 4.1. Programas de alineación más representativos [1].

La mayoría de los programas de alineación de lecturas cortas se basan en dos estrategias algorítmicas:

1. *Algoritmos basados en tablas hash*: Con el objetivo de lograr eficiencia estos algoritmos utilizan una lista de todas las palabras de longitud *n* y determinan una sola vez sus posiciones en el genoma de referencia. Al utilizar tablas hash esta estrategia consume mucha memoria y el esquema básico no considera alineaciones inexactas, una posible solución sería dividir el genoma de referencia en k-mers (subcadenas de longitud k), siendo k mucho menor que n, estos k-mers se pueden almacenar en una lista, tal y como se muestra en la Figura 4.1.

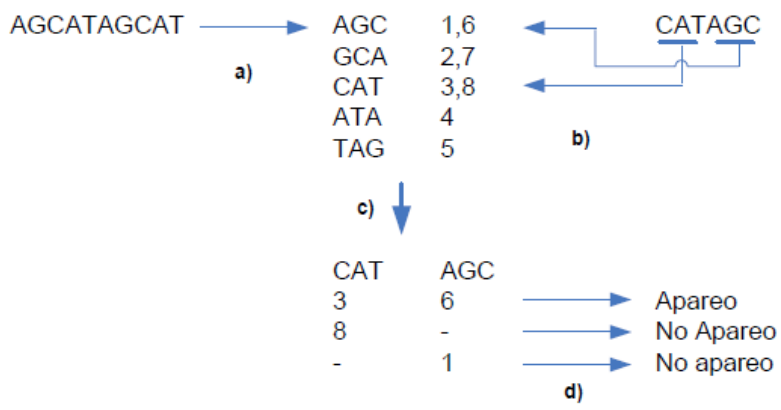

Figura 4.1. Algoritmo de hasheo.

2. *Algoritmos basados en la Transformada de Burrows-Wheeler (TBW)*: La transformada de Burrows-Wheeler realiza una compresión del genoma de referencia de tal forma que es mucho más rápido encontrar las lecturas al realizar el mapeo. Hoy en día muchos de los programas de alineación utilizan esta estrategia, como `SOAP2`, `Bowtie` y `BWA`.

   En los siguientes apartados se explica en detalle el mapeo del ADN y el algoritmo de Burrows-Wheeler, que se utilizará como ejemplo para la integración de la librería `IGP` en el capítulo 5.

   La herramienta utilizada como ejemplo para integrar la librería `IGP` ha sido `bwa` (Burrows-Wheeler Aligner), que implementa una variación del algoritmo de Burrows-Wheeler, por ello se explica con más detalle en el apartado 4.3.

4.1. Definición del problema.

Una vez obtenidas las lecturas de un secuenciador de ADN estas pueden ser utilizadas en proyectos de re-secuenciación, en los que se obtiene el código genético a partir del genoma de una especie que ha sido secuenciada previamente mediante el método De Novo. Uno de los ejemplos más destacados es el del genoma humano, el cual sirve para secuenciar el ADN de individuos en particular mediante la comparación del genoma secuenciado mediante De Novo (genoma de referencia) y el genoma específico de un individuo, esto permite detectar mutaciones o enfermedades genéticas.

El genoma de referencia es fijo y se conoce de antemano, su tamaño es de unas decenas a miles de millones de bases (nucleótidos), por tanto puede indexarse una sola vez para agilizar la búsqueda de bases y ser reutilizado en cada proyecto de re-secuenciación de una sola especie.

La elevada cantidad de lecturas que genera un secuenciador hace necesario utilizar un programa de alineación para detectar variaciones mínimas entre las lecturas y la referencia, con el objetivo de dimensionar el problema, en el genoma humano el número de lecturas es de 10^7-10^8, la longitud de una lectura es de 35-1.100 nucleótidos, por tanto la longitud del genoma será de $3x10^9$ nucleótidos y las variaciones entre un ser humano y otro es del 0,1%. En el siguiente apartado se explicará detalladamente el funcionamiento de estos programas y como detectan las variaciones entre dos genomas [4].

Las lecturas tienen una longitud fija, entre 35 y 1.100 bases generadas por las máquinas NGS, y la cantidad de lecturas puede ser del orden de millones por cada pasada, además, la similitud entre el genoma de referencia y el genoma re-secuenciado es de aproximadamente un 99,9%, por tanto se establece el problema del mapeo de manera general como se muestra a continuación:

Entradas:

- Un conjunto de lecturas cortas procedentes del genoma de cualquier individuo, r_1,.....,r_m
- Una secuencia de referencia R de tamaño |R| que procede de un individuo sano.
- Un grupo de restricciones.

Salida:

- Las posiciones de R donde cada lectura r se mapea de forma exacta o aproximada.

El grupo de restricciones varía de una plataforma NGS a otra y del tipo de datos que se procesa, es decir, si los datos generados son bases sencillas o pares de bases, también depende de las restricciones impuestas por el usuario. Es importante que la definición de mapeo permita alineaciones aproximadas debido a algunas particularidades que ocurren en el genoma, el proceso se muestra en la Figura 4.2.

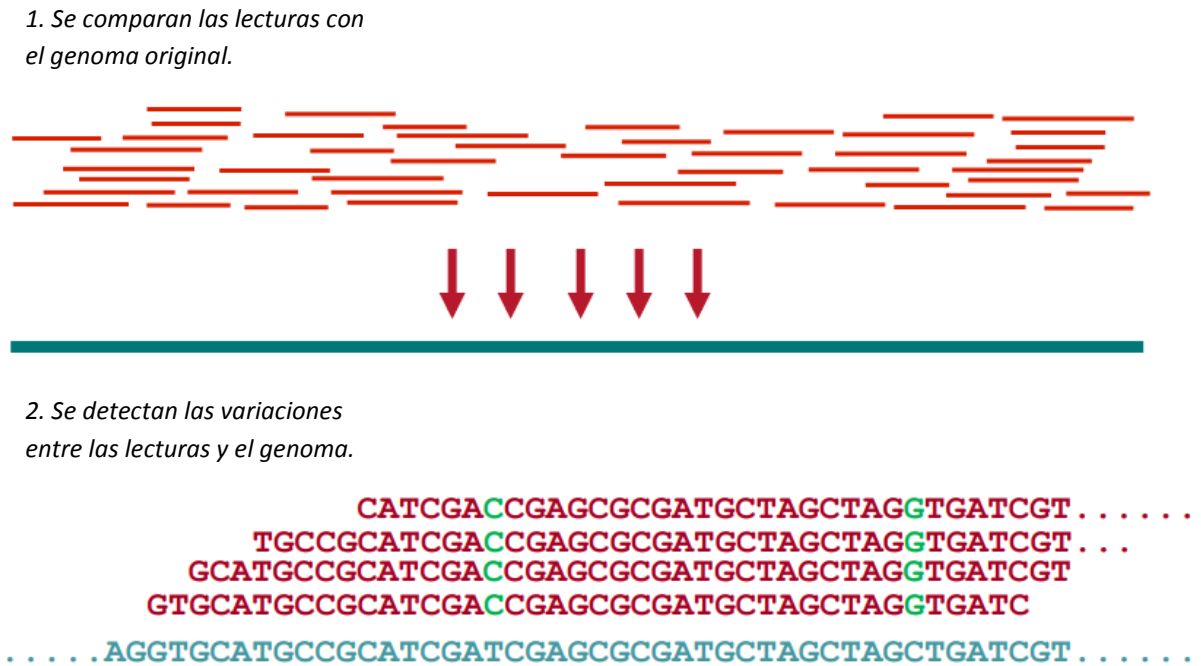

Figura 4.2. Mapeo de ADN.

4.2. La transformada de Burrows-Wheeler.

Surgió en 1994 y fue inventada por David Wheeler y Michael Burrows, consiste en la conversión de la cadena original en otra mediante el uso de todas las posibles rotaciones de la cadena [1, 5], sus objetivos son los siguientes:

1. *Compresión de un genoma*: Se puede utilizar en algoritmos como bzip2 para comprimir el genoma de referencia. El uso de memoria de este algoritmo es de O(|R|), por tanto almacenar la TBW requiere el mismo espacio que el genoma de referencia, ya que solo es una permutación de dicho genoma. Como el alfabeto consiste de 4 letras, se puede almacenar en tan solo dos bits. Por ejemplo, para almacenar la TBW del genoma humano se requieren $2 \times 3 \times 10^9$ bits $\simeq$ 715Mb.

2. *Búsqueda de patrones*: El objetivo principal de la transformada de Burrows-Wheeler es generar una estructura de datos llamada FM-Index que nos permite buscar una secuencia de lectura corta generada por un secuenciador (lectura) en un genoma de referencia, este es el problema a resolver por las herramientas de alineación, que se encargan de alinear las lecturas cortas con el genoma de referencia.

4.2.1. La compresión de Burrows-Wheeler.

Existen varios métodos para comprimir un genoma, el primero de ellos se basa en la idea de que el genoma tiene muchas bases de ADN repetidas, tal como se muestra en la Figura 4.3.

GGGGGGGGGGCCCCCCCCCCCCAAAAAAATTTTTTTTTTTTTTTTCCCCCG

Figura 4.3. Genoma con repeticiones de aminoácidos. Fuente: [5].

Pero en realidad un genoma posee repeticiones de cadenas de caracteres, como se muestra en la Figura 4.4.

GACGACGACGACCATTCATTCATCATTACGTAGACGTAGCACCCC

Figura 4.4. Genoma con repeticiones de cadenas.

Por ello, se proponen los siguientes métodos de compresión [5].

1. *Compresión por longitud de pasada*. Comprime conjuntos de pasadas de n símbolos idénticos como se muestra en la Figura 4.5, pero los genomas no tienen en realidad tantas pasadas.

Genoma

GGGGGGGGGGCCCCCCCCCCCCAAAAAAATTTTTTTTTTTTTTTTCCCCCG

10G11C7A15T5C1G

Codificación de bases simples repetidas

Figura 4.5. Compresión por longitud de pasada.

2. *Compresión de Burrows Wheeler*.

Los genomas tienen muchas repeticiones de conjuntos de aminoácidos, por ejemplo, en la Figura 4.6 se muestran las repeticiones del genoma mostrado en la Figura 4.4.

Figura 4.6. Repeticiones en genoma.

La compresión de Burrows-Wheeler convierte repeticiones a conjuntos de bases simples y aplica la compresión por longitud de pasada, el esquema general del algoritmo se muestra en la Figura 4.7.

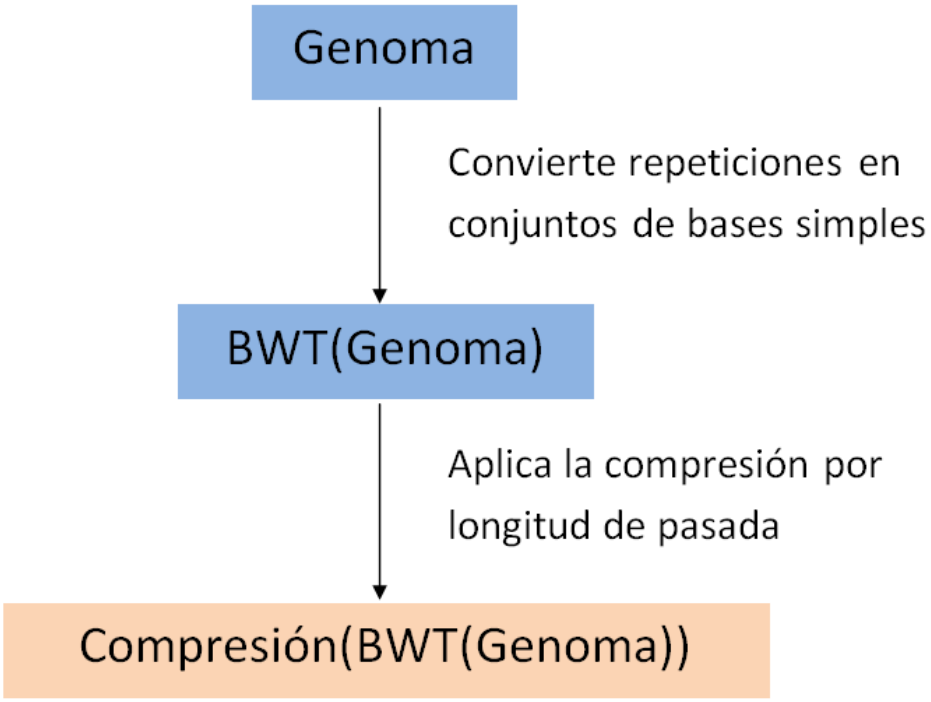

Figura 4.7. Compresión de Burrows-Wheeler.

Se explican a continuación los pasos para generar la transformada BWT a través de un ejemplo. Teniendo los siguientes datos de entrada:

Datos de entrada

Genoma de referencia: R = GCTGAGCTTCT.

Alfabeto: ∑ = {A, C, G, T}

1. El primer paso es agregar el carácter de terminación $ a la cadena R, ya que es lexicográficamente menor que los caracteres del alfabeto, se obtiene como resultado la secuencia R'.

R' = GCTGAGCTTCT**$**

Mediante este carácter es posible revertir la transformación.

2. A continuación, se generan todas las rotaciones a la derecha de R' y se colocan en una matriz M representada en la Tabla 4.2. La columna I_M representa los índices de la matriz.

I_M												
0	G	C	T	G	A	G	C	T	T	C	T	$
1	$	G	C	T	G	A	G	C	T	T	C	T
2	T	$	G	C	T	G	A	G	C	T	T	C
3	C	T	$	G	C	T	G	A	G	C	T	T
4	T	C	T	$	G	C	T	G	A	G	C	T
5	T	T	C	T	$	G	C	T	G	A	G	C
6	C	T	T	C	T	$	G	C	T	G	A	G
7	G	C	T	T	C	T	$	G	C	T	G	A
8	A	G	C	T	T	C	T	$	G	C	T	G
9	G	A	G	C	T	T	C	T	$	G	C	T
10	T	G	A	G	C	T	T	C	T	$	G	C
11	C	T	G	A	G	C	T	T	C	T	$	G

Tabla 4.2. Rotaciones de R' representadas en la matriz M.

3. Se ordenan lexicográficamente (en orden alfabético, teniendo en cuenta que el carácter de terminación es el menor) las filas de M, dando como resultado M' en la Tabla 4.3. Se añaden tanto los índices de M' como la correspondencia con su índice de M para mayor claridad. La última columna de M' es la transformada de Burrows-Wheeler (BWT), representada como L (Last).

I_M	$I_{M'}$											L
0	1	$	G	C	T	G	A	G	C	T	T	C	T
1	8	A	G	C	T	T	C	T	$	G	C	T	G
2	3	C	T	$	G	C	T	G	A	G	C	T	T
3	11	C	T	G	A	G	C	T	T	C	T	$	G
4	6	C	T	T	C	T	$	G	C	T	G	A	G
5	9	G	A	G	C	T	T	C	T	$	G	C	T
6	0	G	C	T	G	A	G	C	T	T	C	T	$
7	7	G	C	T	T	C	T	$	G	C	T	G	A
8	2	T	$	G	C	T	G	A	G	C	T	T	C
9	4	T	C	T	$	G	C	T	G	A	G	C	T
10	10	T	G	A	G	C	T	T	C	T	$	G	C
11	5	T	T	C	T	$	G	C	T	G	A	G	C

Tabla 4.3. Ordenación lexicográfica de M.

Como resultado tenemos:

Resultado

BWT(R) = TGTGGT$ACTCC

Esta columna tiene muchas repeticiones de bases simples las cuales son muy útiles para aplicar la compresión por longitud de pasada.

Compresión por longitud de ejecución de R

Compresión(BWT(R)) = TGTG2T$ACT2C

4.2.1. Descompresión de Burrows-Wheeler.

En el año 2000, Paolo Ferragina y Geovanni Manzini propusieron algunas estructuras de datos auxiliares a BWA que podrían usarse como un índice de R eficiente en espacio, a lo que llamaron índices FM (FM index), que se basan en la propiedad del mapeo Último-Primero:

Lema: Propiedad del mapeo Último-Primero

La n-esima ocurrencia de una letra en particular C en la última columna (L) de la matriz M' es la misma letra como la n-esima ocurrencia de C en la primera columna (F).

En la Tabla 4.4 se muestra una representación de esta propiedad en la matriz M'.

I_M	F										L
0	$\$_1$	G	C	T	G	A	G	C	T	T	C	T_1
1	A_1	G	C	T	T	C	T	$	G	C	T	G_1
2	C_1	T	$	G	C	T	G	A	G	C	T	T_2
3	C_2	T	G	A	G	C	T	T	C	T	$	G_2
4	C_3	T	T	C	T	$	G	C	T	G	A	G_3
5	G_1	A	G	C	T	T	C	T	$	G	C	T_3
6	G_2	C	T	G	A	G	C	T	T	C	T	$\$_1$
7	G_3	C	T	T	C	T	$	G	C	T	G	A_1
8	T_1	$	G	C	T	G	A	G	C	T	T	C_1
9	T_2	C	T	$	G	C	T	G	A	G	C	T_4
10	T_3	G	A	G	C	T	T	C	T	$	G	C_2
11	T_4	T	C	T	$	G	C	T	G	A	G	C_3

Tabla 4.4. Propiedad del mapeo Último-Primero.

Se utilizarán los índices FM del ejemplo anterior para descomprimir el genoma comprimido de R. El esquema general de descompresión se muestra en la Figura 4.8.

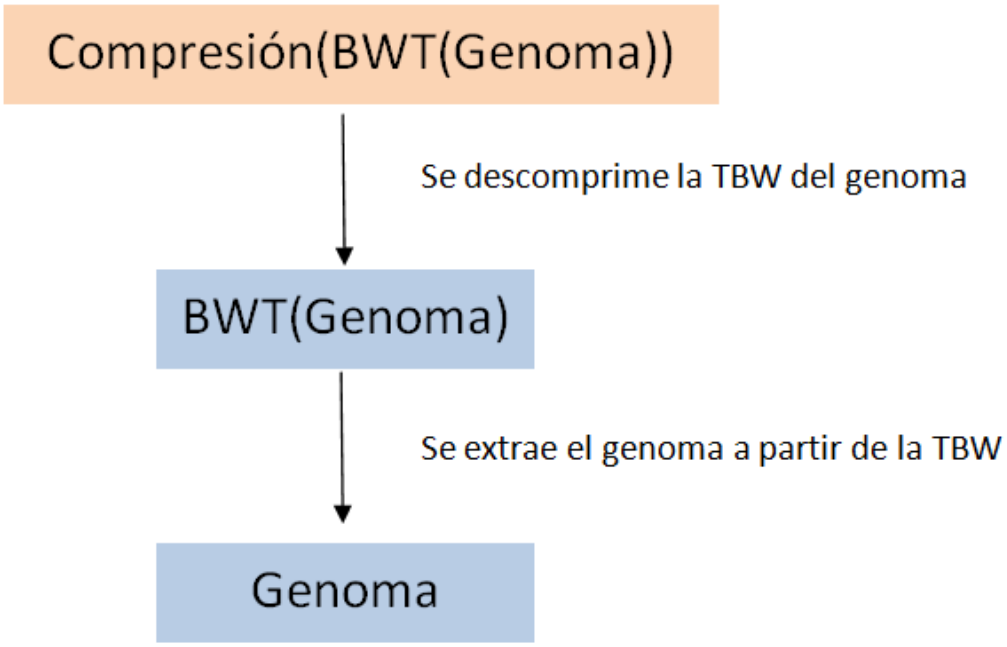

Figura 4.8. Descompresión de Burrows-Wheeler.

Por tanto, los pasos a realizar para descomprimir R serían:

1. Descomprimir R.

Compresión por longitud de ejecución de R

Descompresión(TGTG2T$ACT2C) = BWT(R) = TGTGGT$ACTTC

2. Extraer el genoma a partir de la BWT de R.

Para obtener la cadena original a partir del genoma original se utilizarán los índices FM, realizándose mapeos sucesivos comenzando por el carácter de terminación ($) en la primera columna. Para representar el mapeo se utilizará una matriz de seguimiento en la tabla 4.5.

Iteración	F	V	Resultado
0	Se añade el carácter $		$
1	$_1	T_1	$T
2	T_1	C_1	$TC
3	C_1	T_2	$TCT
4	T_2	T_4	$TCTT
5	T_4	C_3	$TCTTC
6	C_3	G_3	$TCTTCG
7	G_3	A_1	$TCTTCGA
8	A_1	G_1	$TCTTCGAG
9	G_1	T_3	$TCTTCGAGT
10	T_3	C_2	$TCTTCGAGTC
11	C_2	G_2	$TCTTCGAGTCG

Tabla 4.5. Descompresión de Burrows-Wheeler.

El genoma de referencia será la cadena inversa del resultado obtenido a través los mapeos.

Resultado

BWT(R) = GCTGAGCTTCT$

4.2.2. Búsqueda de patrones mediante Burrows-Wheeler.

A continuación se explicará cómo realizar una búsqueda de lecturas en un genoma de referencia, que es el segundo objetivo de la transformada de Burrows-Wheeler.

Siguiendo con el ejemplo anterior, se realizará una búsqueda de las ocurrencias del patrón P = GCT en la cadena de referencia R, ya que el objetivo principal es encontrar todas las lecturas del genoma de un individuo generadas por el secuenciador de ADN en el genoma de referencia. Supongamos que una de esas lecturas es GCT y es crítica para detectar una mutación, por ejemplo en el gen que causa la diabetes.

Para ello se utilizará como entrada la matriz M', la búsqueda está basada en los índices de la primera columna (F) y de la última (L), y un vector que contiene en cada posición *i* el número de veces que el carácter *L(i)* ha aparecido desde el principio de la fila hasta la posición actual de la fila incluida, se denomina vector de rango.

Datos de entrada

Matriz BWT: M'

Patrón de búsqueda: "GCT"

Columnas primera y última de M': F y L.

Vector de posición: R.

1. Se inicia la búsqueda encontrando las filas que comienzan con el sufijo más corto de P (S_1) en M', que es el carácter T (Tabla 4.6).

S_1 (GCT) = ´T´.

Al ser la primera columna parte del índice el carácter T se encuentra en el intervalo [8, 11].

I_M	F											L	R
0	**$\$_1$**	G	C	T	G	A	G	C	T	T	C	**T_1**	**1**
1	**A_1**	G	C	T	T	C	T	$	G	C	T	**G_1**	**1**
2	**C_1**	T	$	G	C	T	G	A	G	C	T	**T_2**	**2**
3	**C_2**	T	G	A	G	C	T	T	C	T	$	**G_2**	**2**
4	**C_3**	T	T	C	T	$	G	C	T	G	A	**G_3**	**3**
5	**G_1**	A	G	C	T	T	C	T	$	G	C	**T_3**	**3**
6	**G_2**	C	T	G	A	G	C	T	T	C	T	**$\$_1$**	**1**
7	**G_3**	C	T	T	C	T	$	G	C	T	G	**A_1**	**1**
8	**T_1**	$	G	C	T	G	A	G	C	T	T	**C_1**	**1**
9	**T_2**	C	T	$	G	C	T	G	A	G	C	**T_4**	**4**
10	**T_3**	G	A	G	C	T	T	C	T	$	G	**C_2**	**2**
11	**T_4**	T	C	T	$	G	C	T	G	A	G	**C_3**	**3**

Tabla 4.6. Búsqueda del primer sufijo.

2. La búsqueda continúa encontrando las filas que comienzan con el segundo sufijo más corto de P (S_1) en M', que es la cadena CT. El espacio de búsqueda actual es [8, 11], en la tabla 4.7 se observa que C precede a T en las filas 8, 10 y 11, por tanto, mapeando el índice de C a la primera columna nos restringimos al intervalo [2, 3, 4].

I_M	F											L	R
0	**$\$_1$**	G	C	T	G	A	G	C	T	T	C	**T_1**	**1**
1	**A_1**	G	C	T	T	C	T	$	G	C	T	**G_1**	**1**
2	**C_1**	T	$	G	C	T	G	A	G	C	T	**T_2**	**2**
3	**C_2**	T	G	A	G	C	T	T	C	T	$	**G_2**	**2**
4	**C_3**	T	T	C	T	$	G	C	T	G	A	**G_3**	**3**
5	**G_1**	A	G	C	T	T	C	T	$	G	C	**T_3**	**3**
6	**G_2**	C	T	G	A	G	C	T	T	C	T	**$\$_1$**	**1**
7	**G_3**	C	T	T	C	T	$	G	C	T	G	**A_1**	**1**
8	**T_1**	$	G	C	T	G	A	G	C	T	T	**C_1**	**1**
9	**T_2**	C	T	$	G	C	T	G	A	G	C	**T_4**	**4**
10	**T_3**	G	A	G	C	T	T	C	T	$	G	**C_2**	**2**
11	**T_4**	T	C	T	$	G	C	T	G	A	G	**C_3**	**3**

Tabla 4.7. Búsqueda del segundo sufijo.

3. Por último, se buscan aquellas filas que comienzan con el tercer sufijo más corto de P (S_1), que coindice con P, por tanto es la última búsqueda en M', que es la cadena GCT. El espacio de búsqueda actual es [2, 3, 4], en la tabla 4.8 se observa que G precede a C en las filas 3 y 4, por tanto, mapeando el índice de G en L a la primera columna encontramos que el rango buscado es [6, 7].

I_M	F											L	R
0	$\$_1$	G	C	T	G	A	G	C	T	T	C	T_1	1
1	A_1	G	C	T	T	C	T	$	G	C	T	G_1	1
2	C_1	T	$	G	C	T	G	A	G	C	T	T_2	2
3	C_2	T	G	A	G	C	T	T	C	T	$	G_2	2
4	C_3	T	T	C	T	$	G	C	T	G	A	G_3	3
5	G_1	A	G	C	T	T	C	T	$	G	C	T_3	3
6	G_2	C	T	G	A	G	C	T	T	C	T	$\$_1$	1
7	G_3	C	T	T	C	T	$	G	C	T	G	A_1	1
8	T_1	$	G	C	T	G	A	G	C	T	T	C_1	1
9	T_2	C	T	$	G	C	T	G	A	G	C	T_4	4
10	T_3	G	A	G	C	T	T	C	T	$	G	C_2	2
11	T_4	T	C	T	$	G	C	T	G	A	G	C_3	3

Tabla 4.8. Búsqueda de la ocurrencia GCT.

El resultado es el número de ocurrencias de GCT en el genoma de referencia.

Para conocer en qué posición del genoma se encuentra la ocurrencia GCT es necesario almacenar las posiciones de comienzo de cada sufijo de las filas, mediante el carácter de terminación de cadena se puede averiguar cuáles son los sufijos. A continuación se muestra un ejemplo en el que se calculan los sufijos de M' en la tabla 4.9. La coordenadas del sufijo en el genoma de referencia (P) se calcula como ***P=longitud(R) - longitud(sufijo).***

I_M	F											L	Sufijo	Longitud(Sufijo)	P
0	$	G	C	T	G	A	G	C	T	T	C	T	$	1	11
1	A	G	C	T	T	C	T	$	G	C	T	G	AGCTTCT$	8	4
2	C	T	$	G	C	T	G	A	G	C	T	T	CT$	3	9
3	C	T	G	A	G	C	T	T	C	T	$	G	CTGAGCTTCT$	11	1
4	C	T	T	C	T	$	G	C	T	G	A	G	CTTCT$	6	6
5	G	A	G	C	T	T	C	T	$	G	C	T	GAGCTTCT$	9	3
6	G	C	T	G	A	G	C	T	T	C	T	$	GCTGAGCTTCT$	12	0
7	G	C	T	T	C	T	$	G	C	T	G	A	GCCTTCT$	7	5
8	T	$	G	C	T	G	A	G	C	T	T	C	T$	2	10
9	T	C	T	$	G	C	T	G	A	G	C	T	TCT$	4	8
10	T	G	A	G	C	T	T	C	T	$	G	C	TGAGCTTCT$	10	2
11	T	T	C	T	$	G	C	T	G	A	G	C	TTCT$	5	7

Tabla 4.9. Construcción del array de sufijos de M'.

Como salida se obtiene el array de coordenadas de sufijos P = [11, 4, 9, 1, 6, 3, 0, 5, 10, 8, 2, 7].

Una vez que se obtiene el array de sufijos y aplicando el algoritmo de Burrows-Wheeler se pueden conocer las posiciones en el genoma de las ocurrencias en el intervalo que se obtiene, en la Tabla 4.10 se muestran las posiciones de GCT en el genoma de referencia a través del array de sufijos.

P	I_M	F										L	R
11	0	$\$_1$	G	C	T	G	A	G	C	T	T	C	T_1	1
4	1	A_1	G	C	T	T	C	T	$	G	C	T	G_1	1
9	2	C_1	T	$	G	C	T	G	A	G	C	T	T_2	2
1	3	C_2	T	G	A	G	C	T	T	C	T	$	G_2	2
6	4	C_3	T	T	C	T	$	G	C	T	G	A	G_3	3
3	5	G_1	A	G	C	T	T	C	T	$	G	C	T_3	3
0	6	G_2	C	T	G	A	G	C	T	T	C	T	$\$_1$	1
5	7	G_3	C	T	T	C	T	$	G	C	T	G	A_1	1
10	8	T_1	$	G	C	T	G	A	G	C	T	T	C_1	1
8	9	T_2	C	T	$	G	C	T	G	A	G	C	T_4	4
2	10	T_3	G	A	G	C	T	T	C	T	$	G	C_2	2
7	11	T_4	T	C	T	$	G	C	T	G	A	G	C_3	3

Tabla 4.10. Posiciones de GCT en R.

En la Tabla 4.11 se comprueba que las posiciones obtenidas en el array de sufijos se corresponden con las posiciones de GCT en el genoma de referencia.

i	0	1	2	3	4	5	6	7	8	9	10	11
Carácter R_i	G	C	T	G	A	G	C	T	T	C	T	$

Tabla 4.11. Posiciones de GCT en R.

A continuación se explica en detalle la herramienta Burrows-Wheeler Aligner (*BWA*), que aplica la transformada de Burrows-Wheeler para mapear secuencias, se hará hincapié en el paso de obtención de las coordenadas de las lecturas en el genoma de referencia.

Siendo el resultado final del proceso de búsqueda el siguiente:

Resultado

COORDENADAS DE P EN R = {0, 5}

4.3. El alineador de Burrows-Wheeler.

Un alineador de secuencias mapea los datos en crudo almacenados en formato FASTQ contra un genoma de referencia ya conocido y previamente indexado para acelerar la búsqueda del algoritmo de alineamiento.

BWA es un paquete de software que usa la transformada de Burrows-Wheeler para indexar el genoma de referencia y mapear secuencias muy parecidas entre sí [1], siguiendo el esquema mostrado en la Figura 4.9.

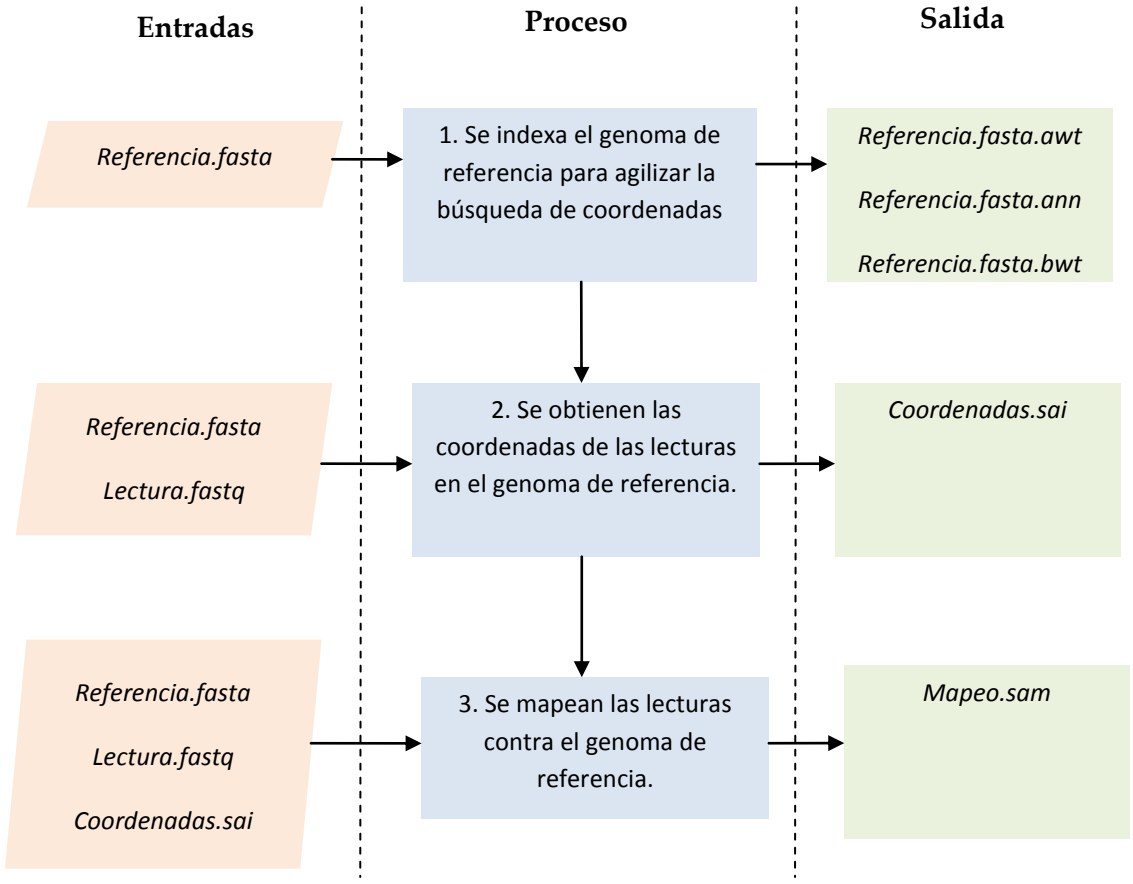

Figura 4.9. Proceso de mapeo en BWA.

Para realizar estos pasos se han utilizado los siguientes comandos:

1. Indexación: `bwa index FR853083.fasta`

2. Búsqueda de coordenadas: `bwa aln -t <nº de hebras> FR853083.fasta ERR002717_1.fastq`

3. Alineación: `bwa samse FR853083.fasta reads.sai ERR002717_1.fastq`

El presente trabajo se centra en la mejora del paralelismo de la utilidad de búsqueda de coordenadas en el genoma de referencia, `bwa aln`.

La difusión de bwa con la librería IGP se ha realizado en Github, el código se puede descargar del siguiente enlace: *https://github.com/amb690/bwa-igp*.

Capítulo 5. Experimentación y pruebas.

En el presente capítulo se detalla el procedimiento seguido para realizar la mejora del paralelismo de la herramienta `BWA ALN` a través de la librería `IGP`.

5.1. Integración de IGP en BWA.

Se han seguido los siguientes pasos para realizar la implementación y pruebas de `IGP` en `BWA` y por ende, mejorar el paralelismo de la herramienta, se muestran de manera resumida en la Figura 5.1.

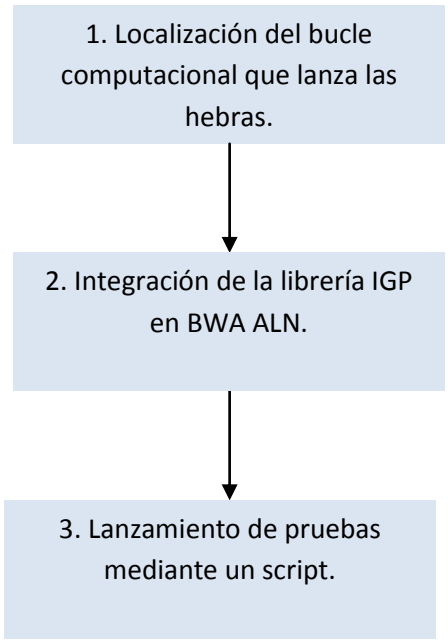

Figura 5.1. Pasos de integración de IGP en BWA ALN.

1. Localización del bucle computacional que lanza las hebras.

El primer paso es analizar la aplicación `bwa aln` y encontrar el bucle que lanza los pthreads. Tras un proceso exhaustivo de análisis se ha detectado que el bucle computacional se encuentra dentro de la función `bwa_aln_core`, el funcionamiento del bucle se muestra en el esquema de la Figura 5.2.

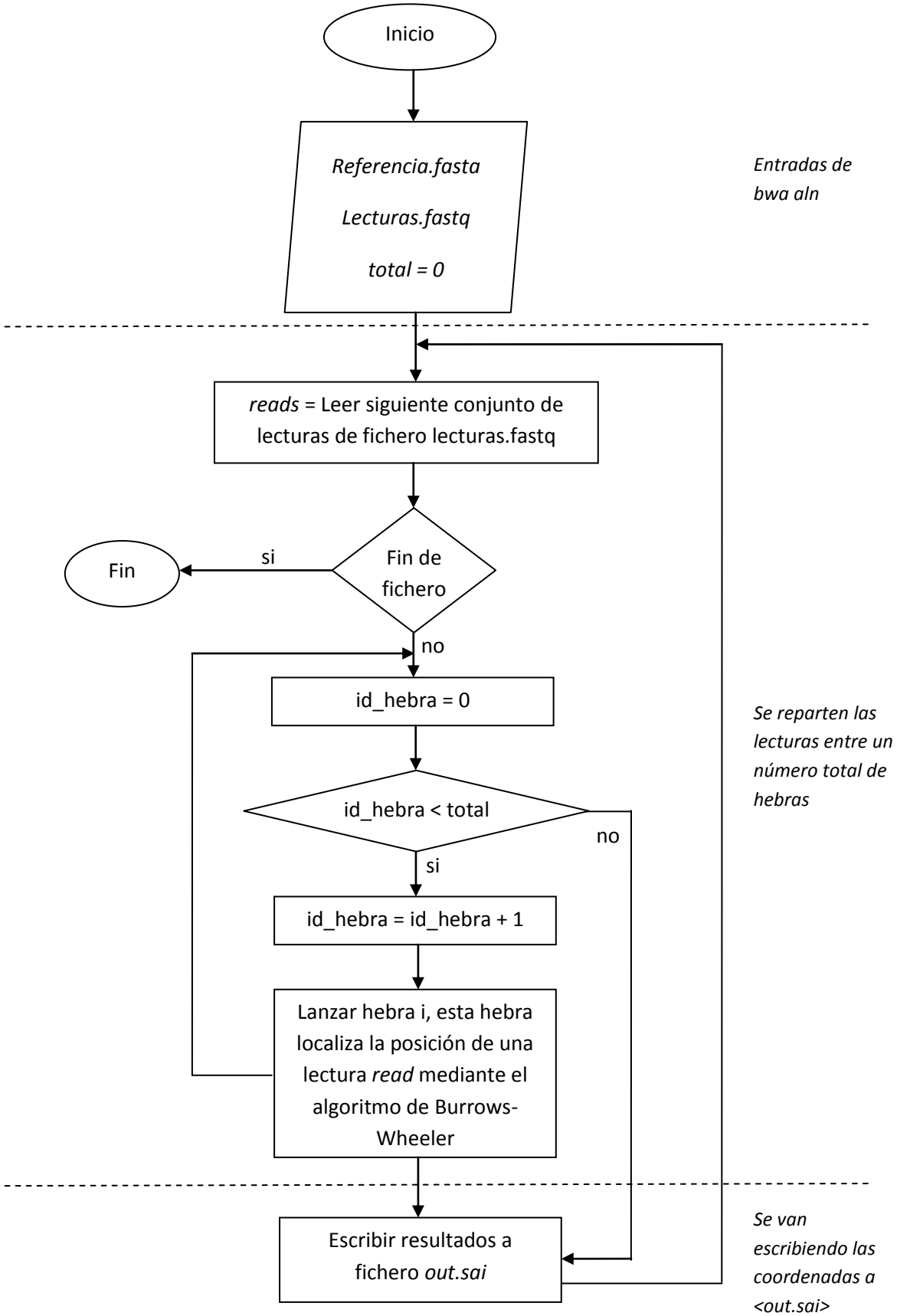

Figura 5.2. Lanzamiento de hebras en bwa aln.

2. Integración de la librería IGP en BWA ALN.

El segundo paso es realizar la integración de la librería dentro del bucle computacional tal y como se explica en el apartado 3.1.2. Se muestra a continuación el pseudocódigo del bucle multihebra utilizando la librería.

Integración de IGP en BWA ALN

```
(1)  include "IGP_library.h"
(2)  func bwa_aln_core(){
(3)      IGP_Initialice();                                   1. Fase de inicialización
(4)      total_hebras=200;
(4)      Mientras leer_secuencia("read.sai");
(5)          id_nueva_hebra=0;
(6)          Mientras (id_nueva_hebra < total_hebras){
(7)              id_nueva_hebra = IGP_Get(0);                2. Fase de evaluación
(8)              Si (id_nueva_hebra >= 0){
(9)                  lanzar_Hebra(id_nueva_hebra, secuencia(id_nueva_hebra));
(10)                 id_nueva_hebra = id_nueva_hebra + 1;
(11)             }
(12)         }
(14)         Escribir_Resultados("fichero.sai");
(13)     }
(14)     IGP_Finalice();                                     3. Fase de finalización
```

3. Lanzamiento de pruebas mediante un script.

Con el objetivo de testear bwa aln en la Xeon Phi se ha implementado un script de pruebas (Figura 5.3) lanzando la aplicación con un número distinto de bloques de secuencias, tomando tres tiempos y haciendo la media entre los dos tiempos más parecidos, ya que debido a varios factores (otros usuarios utilizando la máquina, la propia red...) pueden diferir bastante entre sí.

Para simular un entorno no dedicado se ha utilizado una aplicación que estresa los núcleos de la Xeon Phi, llamada stress. Las pruebas se han realizado tanto en un entorno sin IGP como con la librería ya integrada. A partir de los tiempos tomados se calculan tanto el speedup como la eficiencia.

Al indicarle al programa un número de bloques de secuencias como entrada el número de hebras activas puede ser menor al número de bloques, ya que las hebras no tienen suficiente capacidad de cómputo y finalizan sus cálculos en poco tiempo.

5.2. Resultados.

En el presente apartado se comentan los resultados de la experimentación tanto en sistemas dedicados como no dedicados. En todos los experimentos se han ejecutado (n = 1, 2, 4, 8, 16, 32, 48, 57, 64, 72, 96, 108, 114, 128, 144, 146, 171, 179, 184, 192, 212, 224, 228, 230, 240, 248, 252, 256 bloques de secuencias) y los métodos de la librería que se han testeado han sido NONADAPTABLE (no adaptativo) y ACW (Application decides based on work).

5.2.1. Sistemas dedicados.

En un sistema dedicado los recursos no se comparten entre distintas aplicaciones, sino que están disponibles en exclusiva para la aplicación que los necesita, en este caso `bwa`. La Figura 5.3 muestra los tiempos consumidos por la ejecución en un entorno dedicado a partir de 32 bloques de secuencias, se concluye que utilizando `IGP` los tiempos de ejecución son superiores a la versión de bwa sin utilizar `IGP`, esto se debe a que ejecuta tantas hebras como bloques de secuencias se le indican, mientras que la librería `IGP` solo permite crear hebras si se verifica el criterio seleccionado.

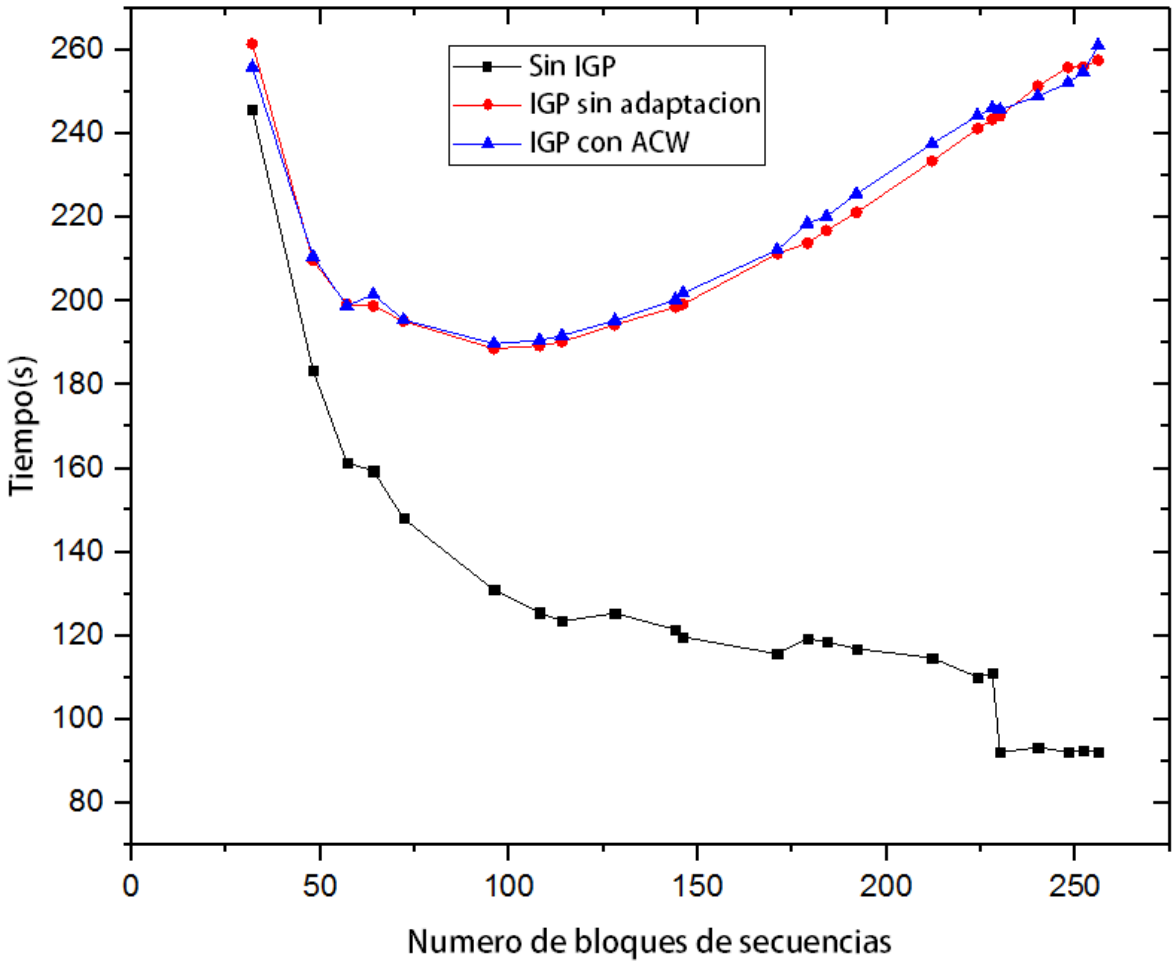

Figura 5.3. Tiempos de BWA en un entorno dedicado

Sin embargo, en la Figura 5.4 la aplicación `bwa` muestra una mayor eficiencia utilizando la librería a partir de 128 bloques de secuencias realizando una evaluación con el método no adaptativo y a partir de 96 bloques de secuencias realizando una evaluación mediante el método ACW, esto significa que varias aplicaciones pueden hacer uso del sistema, suponiendo una ventaja en grandes empresas donde muchos empleados utilizan un solo sistema. Por ejemplo, en el campo de la bioinformática existen muchos técnicos que lanzan varias instancias de la aplicación en el mismo sistema con el objetivo de mapear muchas secuencias de ADN. Además, el aumento de eficiencia permite un mayor aprovechamiento del sistema al haber más procesadores siendo utilizados durante más tiempo de computación.

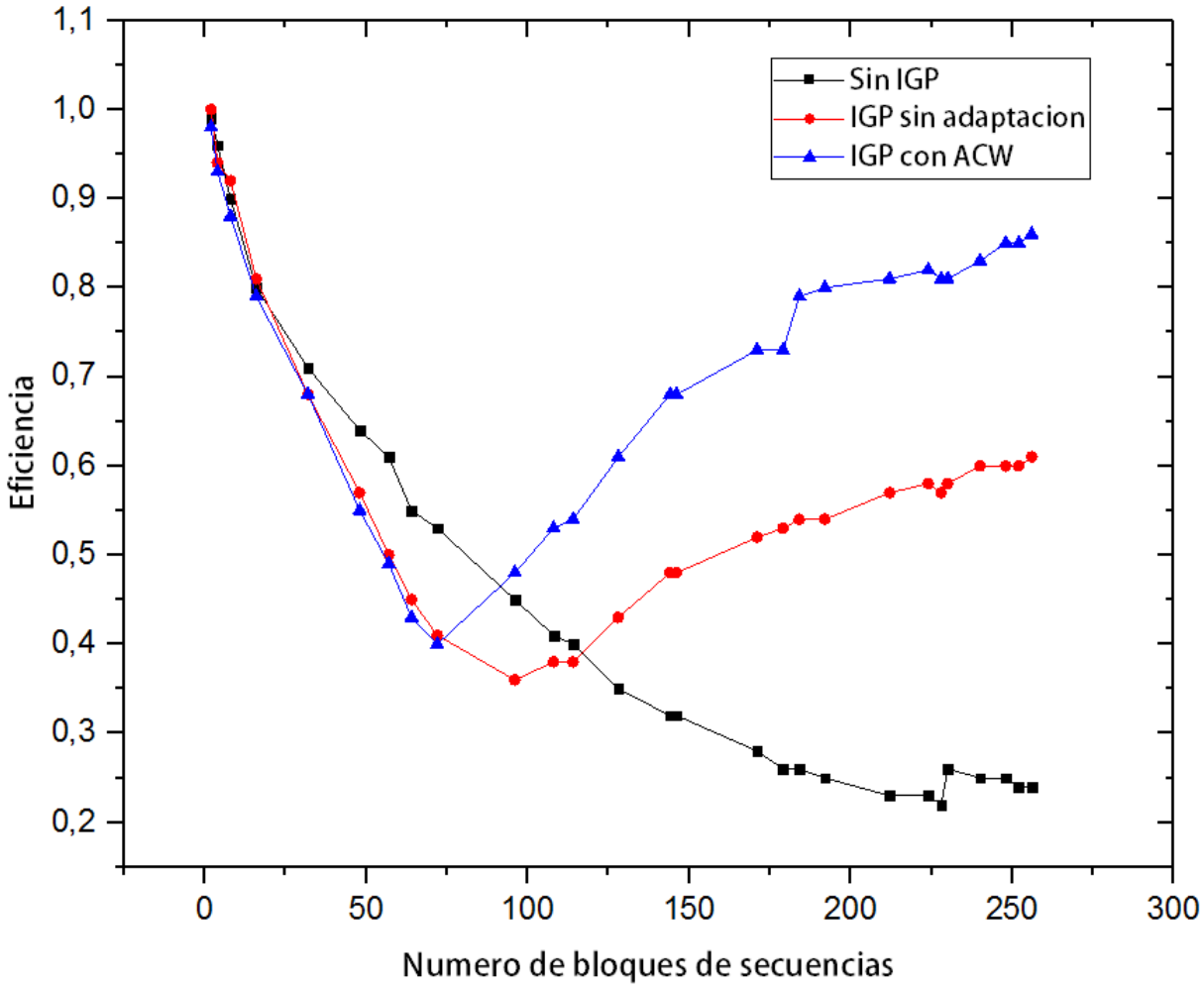

Figura 5.4. Eficiencia de BWA en un entorno dedicado.

El método no adaptativo comienza a ser más eficiente a partir de 96 bloques de secuencias, mientras que el método ACW comienza a superar en eficiencia al método no adaptativo a partir de 68 bloques, ya que las hebras se están creando teniendo en cuenta el rendimiento de la aplicación, mientras que en el método no adaptativo se crean hebras si el número de hebras activas es inferior al número máximo de hebras permitidas.

La Figura 5.5 muestra el número de hebras activas que se lanzan frente al número de bloques de secuencias lanzando IGP mediante el método no adaptativo. El número de bloques de secuencias representa el número de hebras indicado por el usuario pero no el que se lanza realmente, el

número de hebras activas es el número máximo de hebras que se ha lanzado durante el tiempo en que se ha ejecutado la aplicación. Y La Figura 5.6 muestra la eficiencia de bwa con IGP no adaptativo, indicando con etiquetas el número máximo de hebras creadas durante la ejecución. Por ejemplo, la combinación menos eficiente se consigue para 96 bloques de secuencias el Gestor de Paralelismo permite crear 85 hebras como máximo y a partir de ahí comienza a aumentar la eficiencia. Para el resto de situaciones se consigue mejorar la eficiencia reduciendo el número de hebras máximas.

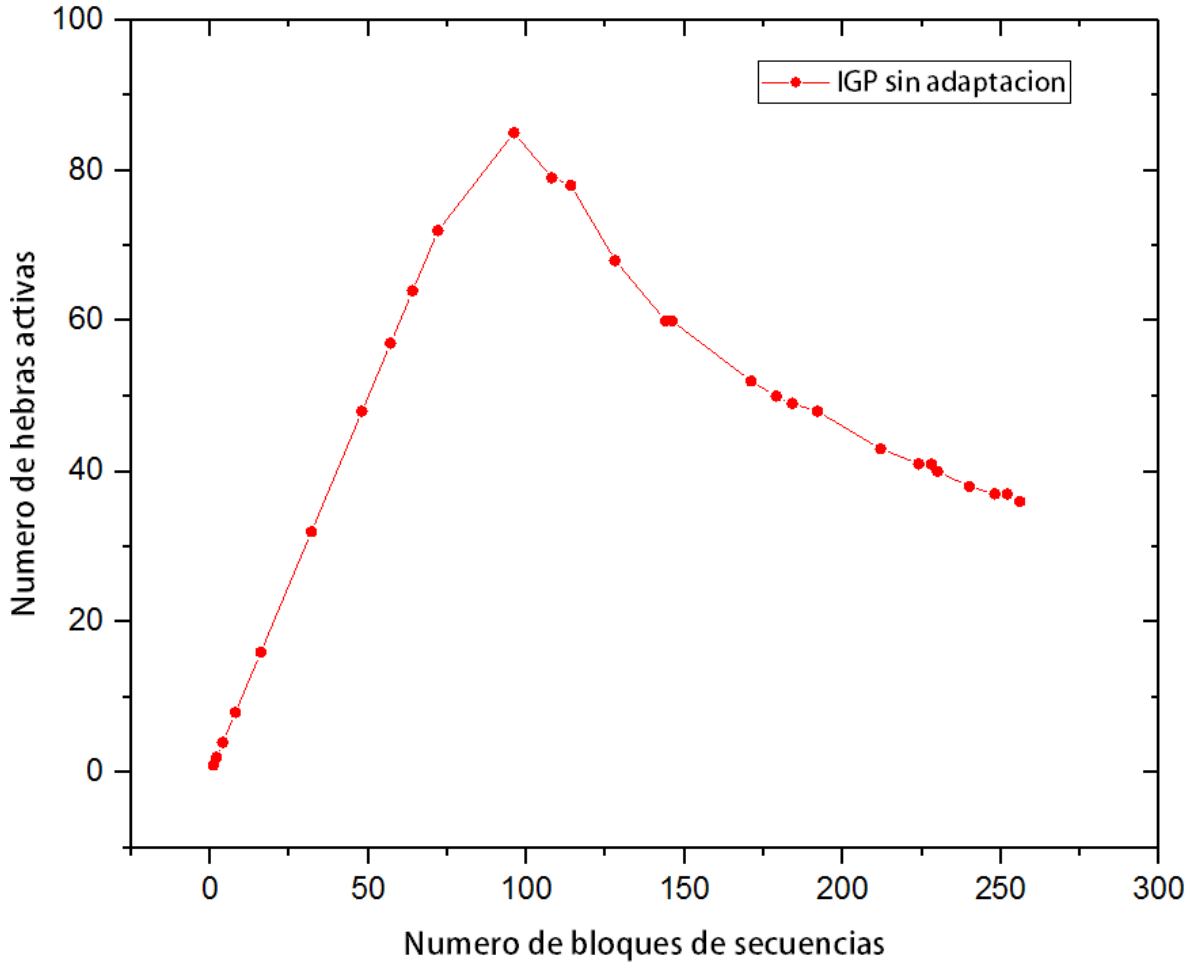

Figura 5.5. Bloques de secuencias frente a hebras activas en IGP no adaptativo.

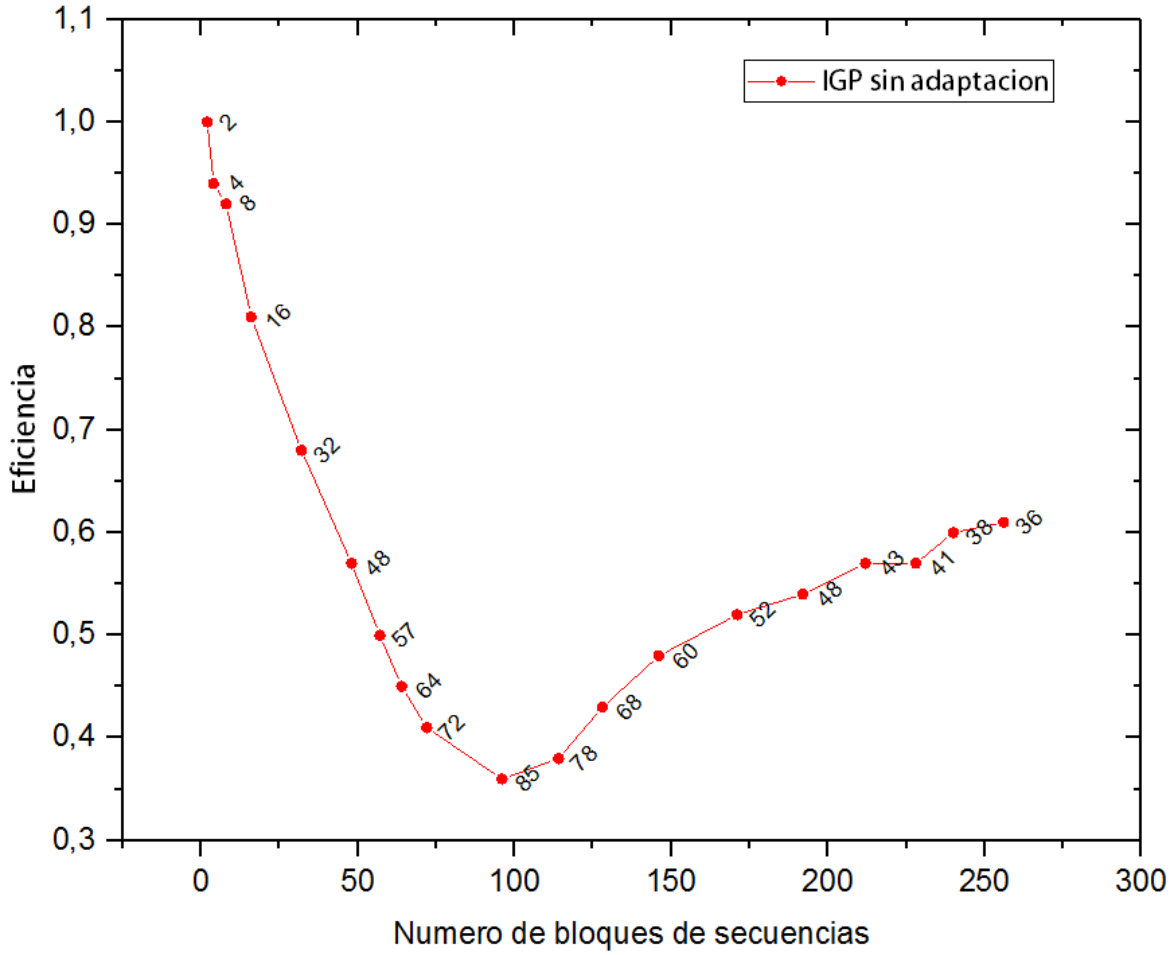

Figura 5.6. Eficiencia de bwa no adaptativo mostrando el número de hebras activas.

Y la Figura 5.7 muestra la eficiencia de bwa con IGP ACW añadiendo el número de hebras activas. A partir de 57 bloques de secuencias y 72 hebras activas comienza a aumentar la eficiencia, lo que significa que el Gestor de Paralelismo ha decidido el número de hebras adecuado que permita mejorar el rendimiento de la aplicación para cada número de bloques de secuencias.

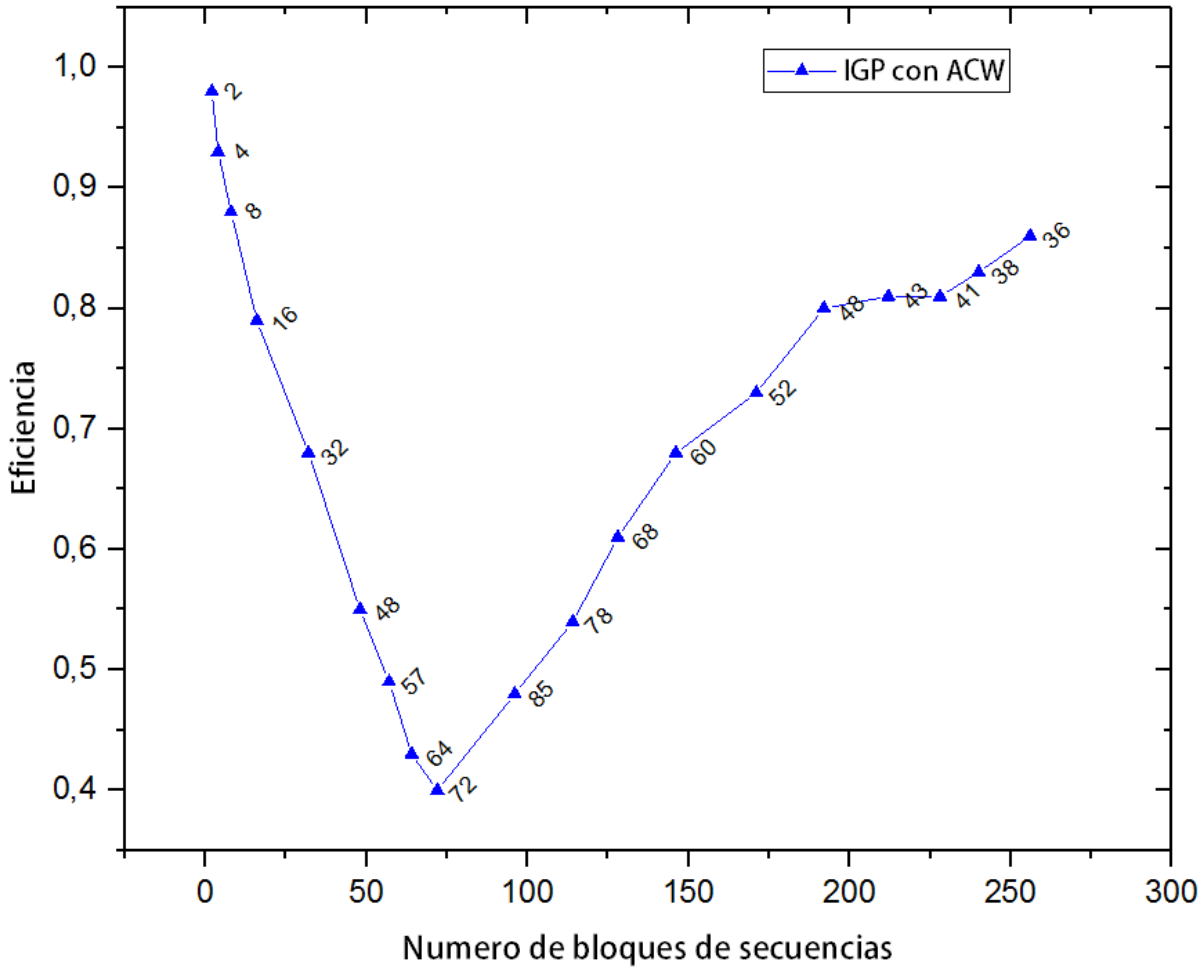

Figura 5.7. Eficiencia de bwa ACW mostrando el número de hebras activas.

5.2.2. Sistemas no dedicados.

En el presente apartado se mostrarán los resultados ejecutando la aplicación `bwa` en sistemas no dedicados, en este tipo de sistemas los recursos no están disponibles en exclusiva para la aplicación, sino que se comparten entre distintas aplicaciones multihebradas.

Para simular la ejecución de otras aplicaciones multihebradas consumiendo recursos se ha estresado el sistema al (25%, 50% y 75%, respectivamente), esto significa, que si la Xeon Phi permite la ejecución simultánea de 228 hilos, se han lanzado utilizando la aplicación stress (57, 117 y 171 hebras, respectivamente) para consumir recursos. El comando para ejecutar la aplicación stress es:

```
stress -c n
```

donde n es el número de hebras ejecutadas, en nuestro caso, sus valores son 57, 114 y 171 hebras respectivamente.

Al igual que en un entorno dedicado, la aplicación que hace uso de la librería `IGP` consume un tiempo mayor ya que permite crear menos hebras, pero permite que se distribuyan mejor los recursos entre

las distintas hebras. En la Figura 5.8 se muestra la eficiencia lanzando la aplicación en la Xeon Phi en un sistema no dedicado utilizando el método no adaptativo. Con estrés 57 la eficiencia comienza a aumentar a partir de 96 bloques de secuencias, mientras que con estrés 114 aumenta a partir de 128 bloques, y con estrés 171 se produce un incremento de la eficiencia a partir de 114 bloques.

Y en la Figura 5.9 se muestra la eficiencia lanzando la aplicación en la Xeon Phi en un sistema no dedicado utilizando el método ACW. Con estrés 57 la eficiencia comienza a aumentar a partir de 96 bloques de secuencias, mientras que con estrés 114 aumenta a partir de 128 bloques, y con estrés 171 se produce un incremento de la eficiencia a partir de 114 bloques.

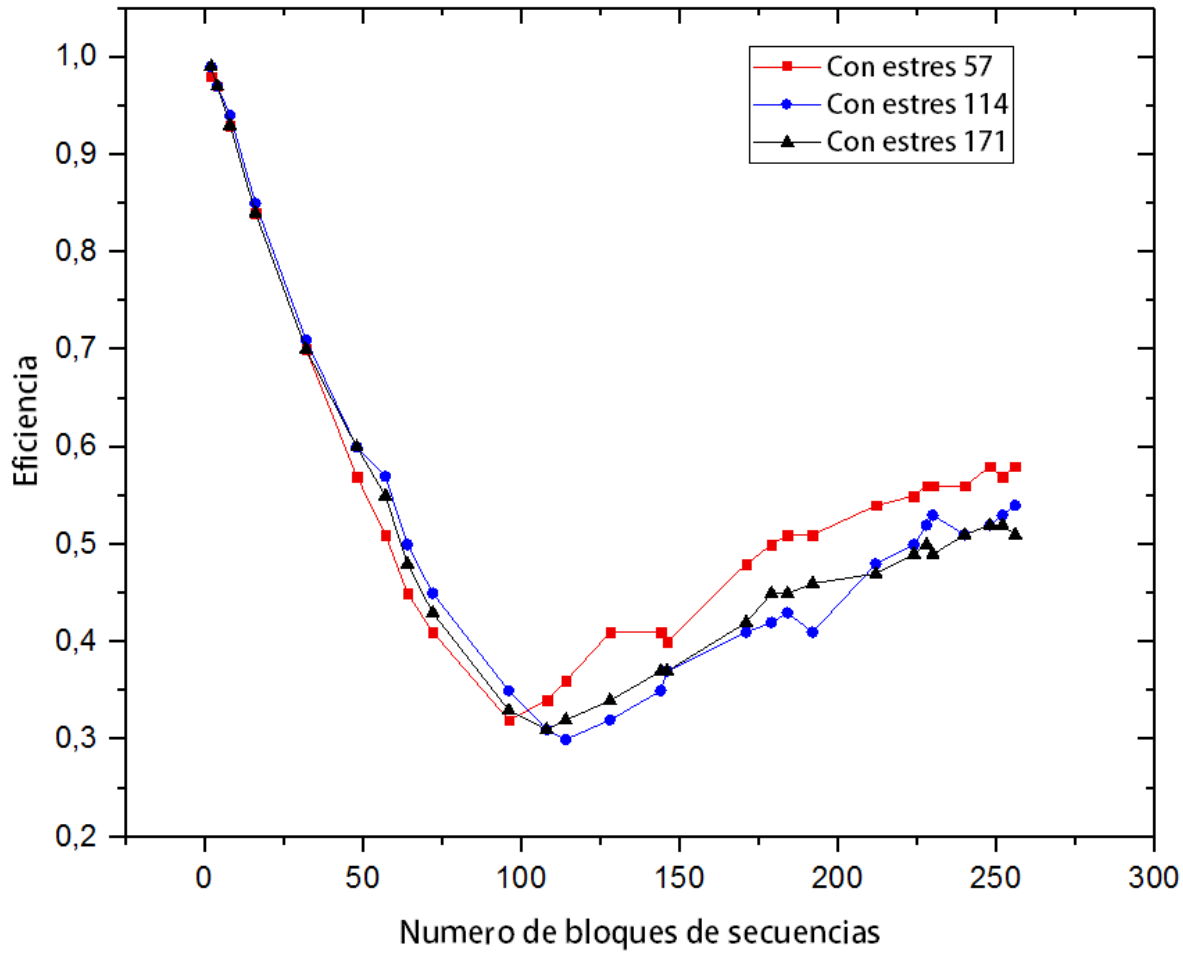

Figura 5.8. Eficiencia de BWA con IGP no adaptativo en un entorno no dedicado.

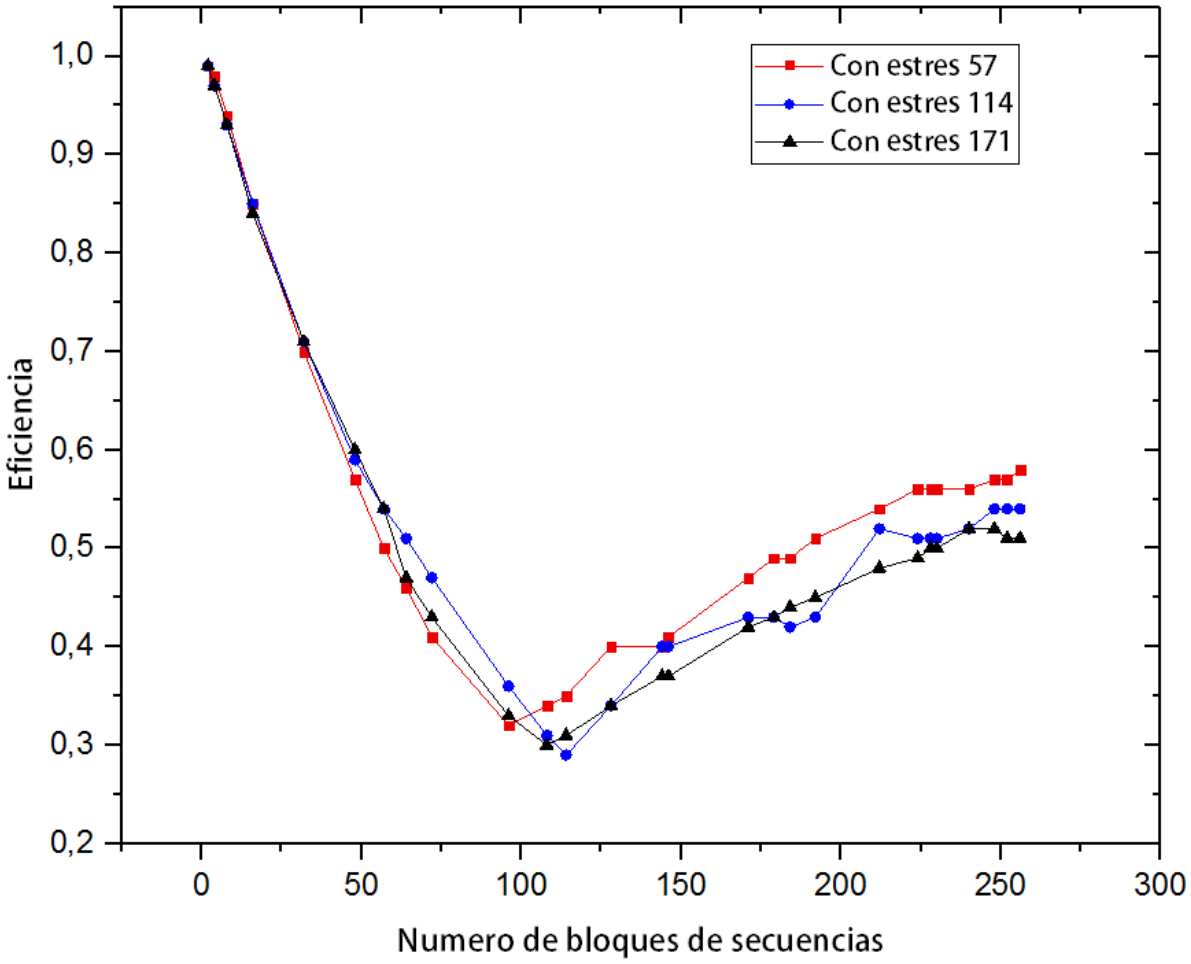

Figura 5.9. Eficiencia de BWA con IGP ACW en un entorno no dedicado.

Capítulo 6. Conclusiones y trabajo futuro.

En el capítulo anterior se han realizado experimentos con diferentes criterios de decisión tanto en entornos dedicados como no dedicados.

Se concluye que el tiempo de ejecución utilizando IGP en sistemas dedicados y no dedicados es superior, debido a que `IGP` decide lanzar menos hebras, no obstante, la eficiencia aumenta con la utilización de la librería, esto quiere decir que durante todo el tiempo en que se está ejecutando la aplicación hay un mayor aprovechamiento de los recursos, y existen más núcleos ocupados, por tanto la carga de trabajo está mejor repartida.

En el caso de un entorno dedicado la eficiencia si se elige el método no adaptativo es inferior a la eficiencia aportada por el método ACW, esto se debe a que mediante el criterio de decisión basado en el rendimiento hay un mayor número de hebras activas. El método no adaptativo obtiene su menor nivel de eficiencia con 96 bloques de secuencias, mientras que el método ACW se produce con 57 bloques. A partir, de sus respectivos mínimos el Gestor de Paralelismo reduce el número máximo de hebras creadas al aumentar el número de bloques, consiguiendo mejorar la eficiencia.

En entornos no dedicados, los niveles de eficiencia conseguidos usando el método no adaptativo no difieren sustancialmente respecto a los obtenidos con el método ACW, debido a que el sistema operativo distribuye las primeras 57 hebras ocupando todos los núcleos de la Xeon Phi y el resto de hebras activas se ejecutan dentro de los hilos de ejecución adicionales de cada núcleo (multithreading).

Como ampliaciones futuras se propone la implementación de un método basado en el consumo energético, así como una herramienta adicional que muestre varias estadísticas finales en una gráfica, como la creación y finalización de hebras en cada instante de tiempo.

Bibliografía

[1] P. Bautista, M. González, and A. Badillo. De la secuenciación a la aceleración hardware de los programas de alineación de ADN, una revisión integral. *In Revista mexicana de ingeniería biomédica*, volume 36, pages 257-275, Tlaxcala, December 2015.

[2] B. Blaise. POSIX Threads Programming. Link: https://computing.llnl.gov/tutorials/pthreads/. 2017.

[3] P. Cock and C.J. Fields, "The Sanger FASTQ file format for sequences with quality scores, and the Solexa/Illumina FASTQ variants ," in *Nucleic Acids Research*, vol. 38, no. 6, pp. 1767-1771, 2010.

[4] P. Compeau and P. Pevzner. "How do we assemble genome?, " in *Bioinformatics Algorithms: An active learning approach 2nd edition*, Vol. 1, Ed. California: Active Learning Publishers, 2015, pages 115-120.

[5] P. Compeau and P. Pevzner. "How do we locate diseases causing mutations?," in *Bioinformatics Algorithms: An active learning approach 2nd edition*, Vol. 2, Ed. California: Active Learning Publishers, 2015, pages 136-157.

[6] M. Guzvic. The history of DNA Sequencing. *In Journal of Medical Biochemistry*, volume 32, pages 301-312, Regensburg, September 2013.

[7] Intel. Advancing Moore's Law in 2014 - The road to 14 nm.
Link: https://www.intel.es/content/www/es/es/silicon-innovations/advancing-moores-law-in-2014-presentation.html. 2014.

[8] Intel. Intel Manycore Platform Software Stack (Intel MPSS).
Link: https://software.intel.com/sites/default/files/managed/09/07/xeon-phi-coprocessor-system software-developers-guide.pdf. 2014.

[9] Intel. Intel Xeon Phi coprocessor developer's quick start guide. Link:
https://software.intel.com/sites/default/files/managed/ee/4e/intel-xeon-phi-coprocessor-quick-start-developers-guide.pdf. 2014.

[10] Intel. Intel Xeon Phi coprocessor system software developers guide.
Link: https://software.intel.com/sites/default/files/managed/09/07/xeon-phi-coprocessor-system software-developers-guide.pdf. 2014.

[11] J. James and J. Reinders. Introduction. In T. Green, editor, *Intel Xeon Phi Coprocessor High-Performance Programming*, chapter 1, pages 1-3. Morgan Kaufmann, 2013.

[12] M. Kchouk, J.F. Gibrat, and M. Elloumi. Generations of sequencing technologies: from first to next generation. *In Biology and Medicine*, volume 9, pages 1-8, Tunis, January 2017.

[13] National Human Genome Research Institute. The cost of sequencing a human genome.
Link: https://www.genome.gov/27565109/the-cost-of-sequencing-a-human-genome/

[14] L. Nederbragt. Developments in next generation sequencing.
Link: https://github.com/lexnederbragt/developments-in-next-generation-sequencing

[15] J.F. Sanjuan-Estrada, L.G. Casado, and I. García. Dynamic number of threads based on application performance and computational resources at run-time for interval branch and bound algorithms. *In Proceedings of CMMSE'10*, volume 3, pages 828-839, Almería, June 2010.

[16] J.F. Sanjuan-Estrada, L.G. Casado, and I. García. Adaptive parallel interval branch and bound algorithms based on their performance for multicore architectures. *The Journal of Supercomputing*, pages 376-384, 2011.

[17] J.F. Sanjuan-Estrada, L.G. Casado, and I. García. Adaptive parallel interval global optimization algorithms based on their performance for non-dedicated multicore architectures. *In Proceedings of PDP 2011 - The 19th Euromicro International Conference on Parallel, Distributed and Network-Based Computing*, pages 252-256, Cyprus, February 2011. IEEE.

[18] J.F. Sanjuan-Estrada. Algoritmos multihebrados adaptativos en sistemas no dedicados. *Tesis Doctoral en la Universidad de Almería*. Enero 2015.

Glosario de términos

ADN - Ácido desoxirribonucleico

ABI - Application Binary Interface

API - Application Programming Interface

BWA - Burrows Wheeler Aligner

BWT - Burrows Wheeler Transform

GP - Gestor de paralelismo

HPC - High Performance Computing

ICC - Intel C++ Compiler

IGP - Interfaz de gestor de paralelismo

MIC - Many Integrated Cores

MPSS - Manycore Platform Software Stack

NGS - Next Generation Sequencing

PCI - Peripheral Component Interconnect

POSIX - Portable Operating System Interface Unix

TBW - Transformada de Burrows Wheeler

SIMD - Single Instruction, Multiple Data

WGS - Whole Genome Sequencing