Universidad Autónoma de Madrid
Escuela Politécnica Superior

Tesis para obtener el doctorado en
Ingeniería Informática y Telecomunicación
por
Universidad Autónoma de Madrid

Tutor de tesis:
Dr. Luis de Pedro Sánchez

Diseño e implementación de sistemas de computación de alto rendimiento para acelerar algoritmos biomédicos

Germán Retamosa de Ágreda

Esta tesis fue presentada en junio de 2017

Tribunal:

Dr. Francisco Javier Gómez Arribas
Dr. Mikel Izal Azcárate
Dr. Javier Tamemes de la Huerta

Germán Retamosa de Ágreda
Diseño e implementación de sistemas de computación de alto rendimiento para acelerar algoritmos biomédicos

Germán Retamosa de Ágreda
Escuela Politécnica Superior. High Performance Computing and Networking Group

IMPRESO EN ESPAÑA

A mi familia, por ser un ejemplo de lucha y constancia.
A Pilar, por su apoyo sincero e incondicional.

Amor vincit omnia

Evaluadores de tesis:

Dr. Javier Tamames de la Huerta

Dr. Mikel Izal Azcárate

Dr. Francisco Javier Gómez Arribas

Fecha de defensa:

Calificación:

Agradecimientos

Esta tesis doctoral es el fruto del trabajo que, junto con otros muchos proyectos, he estado realizando durante estos últimos cinco años. He de reconocer que ha sido una experiencia única donde he podido aprender una gran cantidad de conocimientos que nunca antes habría imaginado y tener una visión más completa del mundo que nos rodea junto con una filosofía de seguir aprendiendo, seguir investigando y seguir viviendo. Tengo que admitir que ha sido una completa locura, pero después de ver los resultados obtenidos, muy reconfortante y personalmente pienso que, sin la ayuda de muchos/as compañeros/as y amigos/as esto no habría sido posible.

En primer lugar, quiero dar las gracias a mi director de tesis y tutor Luis de Pedro Sánchez por su apoyo y por seguir insistiendo cuando todo estaba prácticamente perdido. Su visión global y experimentada ha sido fundamental para dar a este trabajo un espíritu crítico e inconformista donde todo puede ser útil e importante. Por todo esto y más, muchas gracias.

Asimismo, esta tesis doctoral no hubiera sido posible sin la oportunidad que el grupo de Redes y Computación de Altas Prestaciones de la Universidad Autónoma de Madrid y el Departamento de Tecnología Electrónica y de las Comunicaciones de la Escuela Politécnica Superior me ofreció hace 5 años. Gracias a esta oportunidad, he podido conocer a grandes profesionales y aprender de ellos en temáticas tan diversas como las redes de comunicaciones, la computación de altas prestaciones y el diseño de FPGAs, pero sobre todo he podido conocer a grandes personas. Desconozco a quien se le ocurrió introducirse en la bioinformática de altas prestaciones, pero gracias.

Todo este trabajo no hubiera sido posible sin el apoyo de la empresa a la cual pertenezco actualmente, Naudit High Performance Computing and Networking S.L., y que ha hecho todo lo posible para facilitarme la compatibilidad entre mi

vida profesional y mi vida académica. Además, he de agradecer también los acuerdos de colaboración que ha llevado a cabo con centros de investigación de referencia, como el Centro Nacional de Biotecnología y el Centro Superior de Investigación de Salud Pública de Valencia. Eternamente agradecido con estos centros y las personas implicadas por su dedicación, esfuerzo y por aportar un espíritu crítico, objetivo y realista al trabajo de tesis presentado.

Finalmente, una especial mención a mi familia, mi mujer y mis amigos por ser un ejemplo de apoyo incondicional y por ofrecerme esa vía de escape tan necesaria cuando la cantidad de trabajo te desborda. Muchas gracias.

Índice

Índice de Imágenes

Índice de Tablas

Índice de Ecuaciones

Capítulo 1 Introducción

Este capítulo proporcionará un resumen de la siguiente tesis, describiendo la motivación para su realización y los principales objetivos que se han perseguido y alcanzado durante el proceso de investigación. Finalmente, se detallarán las principales contribuciones junto con el esquema principal del documento.

1.1. Motivación

En los últimos años y de manera cada vez más acusada, la información asociada a nuestro estilo de vida es cada vez mayor en volumen y con mayor relevancia. Son cada vez más los dispositivos que entran en escena con el fin de monitorizar todas nuestras actividades, tanto físicas como mentales, y llegar a niveles de diagnóstico jamás alcanzados, es lo que llamamos la era del *big data* [1], [2].

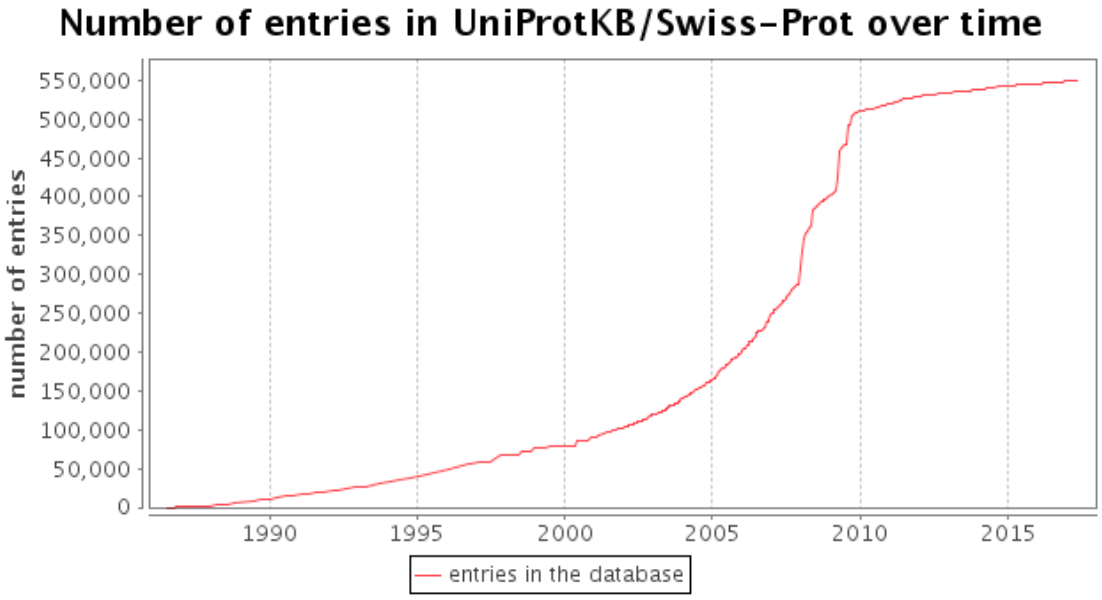

Imagen 1 - Evolución de secuencias en Swiss-Prot

Como se puede observar en la Imagen 1, este rápido crecimiento posee una serie de consecuencias que deben ser tenidas en cuenta con detenimiento y precaución, porque no vale de nada manejar tanta información sino somos capaces de llegar a procesarla ni de analizarla. Abanderado por modelos

predictivos, capaces de alcanzar metas tan lejanas hace unos años como la detección precoz de enfermedades o patologías, la incorporación de tecnologías de altas prestaciones ha sido, es y será la piedra angular de todos estos avances. Sin embargo, todas estas investigaciones y tecnologías funcionarían de manera más eficiente si fuéramos capaces de filtrar toda esta información a aquellos datos que fueran especialmente relevantes para cada persona, es decir, separar el grano de la paja.

Dentro del ámbito de la bioinformática, los procesos de secuenciación genómica, y más concretamente los algoritmos de detección de homologías, han evolucionado vertiginosamente en dos líneas principales, precisión y rendimiento. Históricamente, el primer hito de esta evolución se originó en 1981 con el algoritmo de Smith-Waterman [3]. Este algoritmo tenía por objetivo calcular el factor de similitud entre dos secuencias de aminoácidos mediante técnicas de programación dinámica [4]. Sin embargo, su principal problemática consistía en su alto coste computacional, especialmente en el caso de alineamientos de secuencias de aminoácidos de longitud elevada.

Siguiendo esta progresión de precisión y rendimiento, los algoritmos de detección de homologías incorporaron procesos heurísticos para alcanzar tales objetivos. Algoritmos como BLAST [5] o FASTA [6] fueron desarrollados para cubrir la necesidad de alineamiento de grandes secuencias de aminoácidos. Estos modelos estadísticos eran capaces de obtener dichos alineamientos mucho más rápido y con un alto grado de equivalencia frente a los métodos de programación dinámica [7]. Sin embargo, con el número de secuencias de aminoácidos creciendo mucho más rápido que la capacidad analítica de los algoritmos de detección de homologías existentes, la necesidad de acelerar estos algoritmos se convirtió más que nunca en el tema central de la biología computacional. Por consiguiente y basándose en la idea de acelerar estos algoritmos, varias tecnologías de computación de alto rendimiento, del inglés *High Performance Computing* (HPC), han sido utilizadas hasta la fecha.

Field Programmable Gate Arrays (FPGAs) es el primer elemento de esta colección. Concretamente, esta tecnología está basada en la modificación del código fuente a tratar y la aceleración de los módulos con mayor carga computacional para obtener aceleraciones muy elevadas. Algunos ejemplos de estas aceleraciones están representados en algoritmos como BLAST [8]–[11] o Smith-Waterman [12]–[14]. Sin embargo, todas estas aceleraciones tan significativas tienen una serie de limitaciones a tener en cuenta antes de elegir este tipo de tecnologías. En primer lugar y debido a que el código fuente es traducido para su ejecución en la FPGA dentro de un chip con capacidades reducidas, limita el número de funciones a optimizar dentro de la aplicación. Además, el coste de este tipo de tecnología es muy elevado por lo que no es el mejor candidato para obtener una buena relación rendimiento-precio.

Graphics Processing Units (GPUs) son la segunda y más factible opción frente a las FPGAs. Este tipo de tecnología es capaz de acelerar algoritmos como BLAST [15]–[17] o Smith-Waterman [18]–[20] mediante la modificación del código fuente original (*refactoring*) y el uso de librerías propias. Al contrario que con las FPGAs, no existe una limitación física pero la complejidad computacional de los algoritmos de detección de homologías, el manejo de los diferentes tipos de memoria dentro de una GPU (memoria compartida o global) y las latencias lo convierten en una prometedora alternativa con una buena relación rendimiento-precio.

En la actualidad, existe una nueva línea de investigación basada en la detección remota de homologías mediante técnicas predictivas. Estás técnicas predictivas utilizan algoritmos de aprendizaje automático, como por ejemplo *Support Vector Machines* (SVM) o *Learning To Rank* (LTR) [21]–[23]. Todas estas técnicas son capaces de ayudar a los métodos tradicionales de detección de homologías o aproximaciones basadas en el filtrado de secuencias en su caso de peor rendimiento, es decir, secuencias con similitud muy baja. Asimismo, y a pesar que estos métodos de aprendizaje tienen un impacto significativo en el

rendimiento del sistema global, la utilización de hardware HPC, como por ejemplo FPGAs o GPUs, puede ayudar a paliar este efecto negativo.

Finalmente, existe una última alternativa basada en el filtrado de secuencias. La idea principal se basa en reducir rápidamente el número de secuencias de una base de datos basándose en criterios de relevancia de las secuencias dado un modelado teórico previo [24], [25]. Esta será la aproximación seguida dentro del trabajo de tesis propuesto.

1.2. Objetivos

El objetivo de la presente tesis está basado en el diseño e implementación de una solución de filtrado de secuencias mediante GP-GPU [26][27]. Esta propuesta difiere de otras alternativas analizadas en que es completamente independiente del algoritmo de secuenciación utilizado y no requiere de la re-implementación de su código fuente. Además, aprovecha todo el potencial del procesamiento paralelo sobre GP-GPU, por lo que puede ser utilizado conjuntamente con cualquier algoritmo de detección de homologías y llevar a cabo secuenciación de aminoácidos o nucleótidos en menos tiempo, menor coste y con mayor precisión en función del algoritmo utilizado.

Este último punto ha sido recientemente investigado en el que se relaciona la precisión del algoritmo BLAST en términos de su espacio de búsqueda [28]. De esta manera, la investigación concluye que la precisión del algoritmo es inversamente proporcional al número de secuencias dentro de la base de datos. Esta es una técnica de programación muy común utilizada para evitar largas ejecuciones provocadas por grandes cantidades de datos.

Con el objetivo de establecer una línea de diseño e implementación sólida, se ha utilizado el algoritmo BLAST como algoritmo de referencia debido a su uso extendido en el ámbito de la bioinformática, su arquitectura modular y naturaleza paralelizable [5]. Además, ha sido de especial importancia durante la fase de

evaluación y pruebas del sistema propuesto, ya que ha permitido establecer la línea base sobre la cual determinar los porcentajes correspondientes de mejora, tanto a nivel de rendimiento como de precisión. Por consiguiente, el principal resultado obtenido es un sistema de filtrado que, combinado junto con la versión del algoritmo NCBI BLASTP, mejora mediante el uso de GPU su rendimiento y precisión iniciales, alcanzando resultados prácticamente idénticos que el algoritmo original, con un 70% de precisión en el caso peor y un 100% en el caso mejor, y un factor de aceleración hasta cinco veces mayor (5x).

Este proceso de evaluación fue llevado a cabo por diferentes investigadores del Centro Nacional de Biotecnología (CNB-CSIC) y el Centro Superior de Investigación y Salud Pública de Valencia (CSISP) donde se evaluaron 20.000 secuencias de entrada procedentes de diferentes tipos de genomas, sesgados o no sesgados frente a bases de datos de referencia como GenBank [26][27][29].

A continuación, se enumeran los conceptos más relevantes del trabajo propuesto:

- El sistema de filtrado propuesto requiere únicamente un hardware GP-GPU para su funcionamiento, siendo ésta la solución con mejor relación coste-rendimiento y uso más extendido.
- El sistema de filtrado propuesto está basado en modelos heurísticos y puede ser fácilmente modificado por cualquier otro sistema de filtrado o modelo heurístico existente.
- El sistema de filtrado propuesto es completamente transparente y compatible con cualquier algoritmo de detección de homologías, como por ejemplo NCBI BLAST.

Todo el código fuente, documentación e instrucciones de instalación están disponibles en Github [30] como dominio público y bajo licenciamiento MIT.

Capítulo 2 Estructura de la tesis

La presente tesis tiene por objetivo describir el proceso de diseño, implementación y evaluación de un modelo de pre-filtrado basado en GPU y aplicado a algoritmos de detección de homologías tradicionales, como por ejemplo BLAST, para su optimización en términos de precisión y rendimiento.

Los dos capítulos siguientes formarán parte del estado del arte llevado a cabo para realizar la presente tesis y está compuesto por las siguientes categorías:

- **Detección de homologías:** A lo largo de este capítulo se hará un recorrido cronológico por los diferentes algoritmos de detección de homologías que permita estudiar la evolución que estos algoritmos han experimentado a lo largo del tiempo, determinar sus principales ventajas e inconvenientes y analizar las tendencias de secuenciación más actuales. Además, este capítulo detallará el modelo teórico-estadístico de Karlin-Altschul y que ha sido utilizado por múltiples algoritmos de detección de homologías, como por ejemplo BLAST, a lo largo de los últimos años.
- **Sistemas de computación de altas prestaciones:** En este capítulo se analizarán los diferentes sistemas de computación de altas prestaciones utilizados dentro del ámbito de la bioinformática y aquellas aproximaciones que se han tomado para fusionar estos sistemas con los algoritmos de detección de homologías existentes.

Por consiguiente, el presente estado del arte tiene por objetivo determinar el mejor método de detección de homologías según diferentes factores evaluados, el sistema de computación de altas prestaciones más adecuado para implementar dicho algoritmo y con la base teórica de Karlin-Altschul. Este proceso de análisis ha dado como resultado la solución propuesta que se detallará en los capítulos posteriores.

A continuación, el siguiente capítulo definirá la metodología del sistema de filtrado, siendo ésta la base para el diseño de la solución propuesta y desembocando en la descripción de la implementación de la solución propuesta, detallando las diferentes consideraciones adicionales que han sido tenidas en cuenta para optimizar el desarrollo en el sistema de computación de altas prestaciones seleccionado. Una vez explicadas tanto la metodología como la implementación del sistema propuesto, el siguiente capítulo describirá el proceso de evaluación llevado a cabo, clasificando las diferentes pruebas realizadas en términos de precisión de los resultados y rendimiento de la solución propuesta e implementada.

Finalmente, los cuatro últimos capítulos de la presente tesis describirán las conclusiones al trabajo de investigación realizado, la divulgación de los resultados obtenidos en diferentes revistas y congresos internacionales de alto impacto que han servido para corroborar la relevancia de la solución propuesta desde un punto de vista académico, las aplicaciones industriales que ha tenido la solución propuesta dentro de centros de referencia en biotecnologías y los diferentes trabajos futuros que den continuidad al trabajo de tesis presentado.

Capítulo 3 Detección de homologías

Este capítulo presenta la primera parte el estado del arte de la presente tesis doctoral, revisando cronológicamente los algoritmos y técnicas de secuenciación más relevantes y con mayor proyección. Asimismo, se evaluarán sus principales ventajas e inconvenientes respecto al sistema propuesto.

La estructura de dicho capítulo comenzará con los orígenes de la detección de homologías, basada en alineamientos globales y locales, con los algoritmos de Needleman-Wunsch [31] y Smith-Waterman [3] como principales actores respectivamente. A continuación, se dará paso a los principales algoritmos de detección de homologías actuales, basados en alineamientos locales, y su evolución a lo largo del tiempo. Además, se realizará una mención especial a uno de las aplicaciones más utilizadas y extendidos en la actualidad, BLAST [5], así como la motivación existente para la implementación y diseño en el sistema propuesto. Finalmente, se concluirá con las últimas tendencias que están surgiendo en la actualidad, basadas en modelos predictivos y aprendizaje automático (*machine learning*).

3.1. Needleman-Wunsch

Los algoritmos de detección de homologías existentes en la actualidad se basan en las técnicas de programación dinámica de los años 70. Este modelo de programación, creado en 1953 por Richard Bellman, tiene como foco central la optimización de problemas complejos que pueden ser discretizados y secuencializados. Concretamente, dentro del ámbito de la bioinformática, este modelo de programación toma un enfoque *top-down* donde el proceso de secuenciación se divide en pequeños subproblemas superpuestos, y éstos se resuelven recordando su solución para un uso posterior, a semejanza de un modelo de programación recursiva.

Dicho algoritmo, creado por Saul Needleman y Christian Wunsch [31] es utilizado actualmente para realizar alineamientos globales entre secuencias de aminoácidos y funciona con total independencia a la complejidad o longitud de las secuencias, obteniendo así la solución óptima para cada alineamiento.

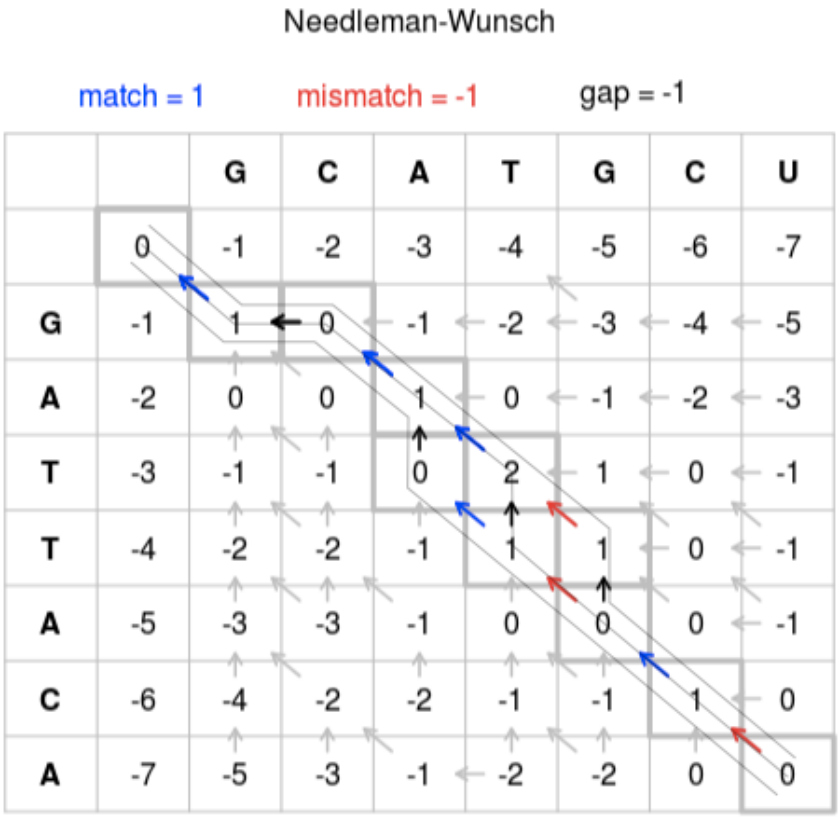

Imagen 2 - Ejemplo de algoritmo Needleman-Wunsch

Tal como se observa en la Imagen 2, con el objetivo de alcanzar el alineamiento global óptimo entre dos secuencias, el algoritmo representará dentro de una matriz bidimensional las dos secuencias de aminoácidos a alinear globalmente y definirá tres parámetros correspondientes por coincidencia, fallo o *indel* (*insertion* o *deletion*), es decir, se produce un alineamiento introduciendo huecos entre los propios aminoácidos. Dichos parámetros serán de especial relevancia en el algoritmo ya que determinarán su dirección a través de los diferentes aminoácidos, tanto en la primera fase de asignación de valores como en la fase de vuelta atrás (*trace-back*) y pueden seguir diferentes esquemas de puntuación:

- Modelo básico
    - Este modelo se basa únicamente en los parámetros de coincidencia y fallo y asigna valores fijos a cada uno de estos parámetros en base a la experiencia adquirida previa.
- Modelo de matrices de verosimilitud
    - De la misma manera que en el caso anterior, este modelo se basa únicamente en los parámetros de coincidencia y fallo, aunque en este caso de manera unificada. Esto es debido a que dichos valores son obtenidos a través de una matriz bidimensional de verosimilitud obtenida mediante la combinación de procesos estadísticos y experiencia previa [32]. Los ejemplos más conocidos de matrices de substitución son BLOSUM (*Blocks Substitution Matrix*) y PAM (*Point Accepted Mutation*).
- Modelo de penalización por huecos (*gaps*)
    - Este modelo se basa en la consideración que cuando alineamos secuencias de aminoácidos ocurren frecuentemente huecos que pueden llegar a generar un alineamiento de mayor longitud y, por tanto, con mayor probabilidad de relevancia. Sin embargo y con el fin de penalizar alineamientos con grandes huecos, se toman valores de puntuación distintos a nivel de aperturas de huecos o huecos adicionales a otros ya existentes.

Uno de los principales problemas del algoritmo Needleman-Wunsch es su fuerte dependencia con la longitud de las secuencias a alinear y el rendimiento obtenido en términos de su complejidad algorítmica $O(mn)$ [33]. Concretamente, m y n se corresponden con las longitudes de cada una de las secuencias alineadas y que, en escenarios con secuencias de longitud elevado, tanto su espacio de memoria como su tiempo de ejecución se hacen inasumibles para cualquier trabajo de secuenciación. No obstante, existen alternativas optimizadas con complejidad algorítmica $O(m+n)$ pero a costa de incrementar el tiempo de computación [34].

3.2. Longest Common Sequence (LCS)

Este algoritmo, creado por Dan Hirschberg en 1975, fue utilizado en bioinformática para reducir la complejidad computacional y espacio en memoria de algoritmos como Needleman-Wunsch [34]. Su objetivo principal está centrado en encontrar una sub-secuencia más larga, es decir el alineamiento óptimo, que es común al resto de secuencias.

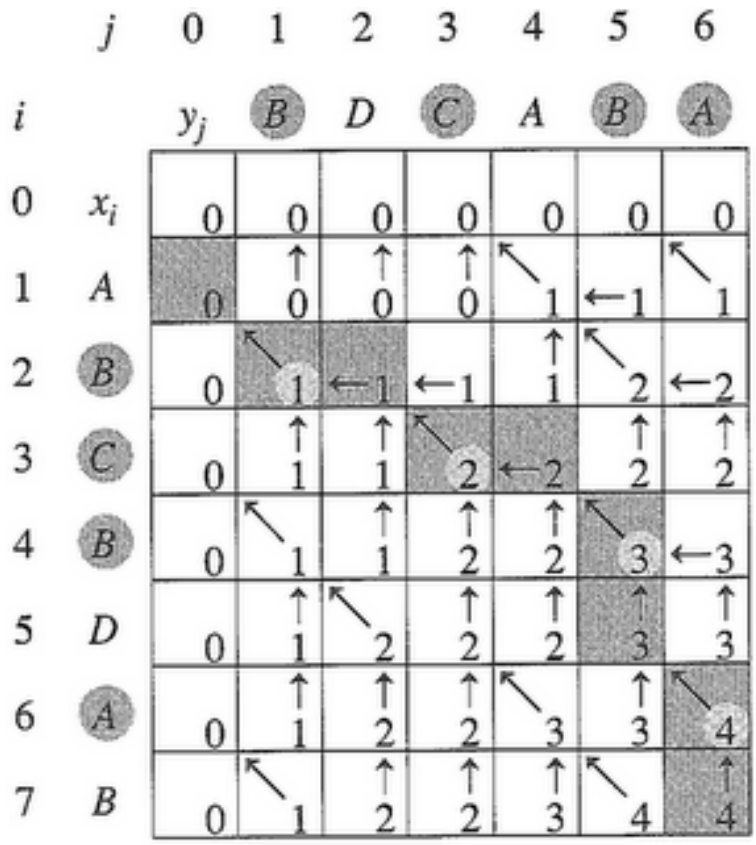

Imagen 3 - Ejemplo de algoritmo LCS

Tal como se puede observar en la Imagen 3, dadas dos secuencias de aminoácidos a alinear x_i y y_j, se compone una matriz bidimensional con valores basados en el esquema de puntuación definido en el algoritmo de Needleman-Wunsch [31] y con la particularidad que, tanto la primera fila como la primera columna tiene valores auto-asignados a cero.

Una vez establecidos los valores entre aminoácidos, se procederá a obtener la longitud de la secuencia más larga que servirá de punto de partida para que, mediante el mecanismo de marcha atrás (*trace-back*), se obtenga el alineamiento óptimo entre ambas secuencias.

Dicho cálculo seguirá la siguiente notación matemática:

$$SCL(X[0..m], Y[0..n]) = \begin{cases} 0 & si\ m = 0\ o\ n = 0 \\ SCL(X[0..m-1], Y[0..n-1]) + 1 & si\ X[m] = Y[n] \\ \max\big(SCL(X[0..m], Y[0..n-1]), SCL(X[0..m-1], Y[0..n])\big) & resto \end{cases}$$

Ecuación 1 - Algoritmo LCS

Como se puede observar en la Ecuación 1 y al igual que con el algoritmo Needleman-Wunsch [31], el tiempo de procesamiento se hace excesivamente costoso para el alineamiento entre secuencias de longitud elevada [35].

3.3. Smith-Waterman

Con anterioridad a la aparición del algoritmo Smith-Waterman, todos los algoritmos creados hasta el momento estaban basados en técnicas de programación dinámica que tenían por objetivo encontrar el alineamiento global óptimo entre dos secuencias de aminoácidos o nucleótidos. Pongamos por ejemplo un alineamiento entre secuencias de genomas con bajo porcentaje de verosimilitud. En este caso, esperaríamos que un conjunto reducido de secuencias permaneciera inalterable dada la propia naturaleza evolutiva del genoma y otro conjunto mayor fuera completamente irreconocible debido a las diferentes inserciones y borrados introducidos durante el alineamiento. Por consiguiente, en este caso particular, aquellos algoritmos tradicionales basados en alineamientos globales aportarían resultados poco concluyentes respecto al alineamiento. Sin embargo, en caso de aplicar algoritmos basados en alineamientos locales, se podrían llegar a obtener alineamientos de sub-

secuencias cuyo valor de relevancia sea lo suficientemente significativo para establecer relaciones locales entre ambas.

La entrada en escena de algoritmos como Smith-Waterman supuso el comienzo del paradigma entre alineamientos locales y su comparación respecto a los alineamientos globales ya que, aunque ambos mecanismos son prácticamente iguales, poseen propiedades diferenciadoras a tener en cuenta.

Este algoritmo, creado por Temple F. Smith y Michael S. Waterman en 1981, asegura la obtención del alineamiento local óptimo basado en una matriz de substitución y un esquema de puntuación de huecos [3]. La principal diferencia respecto a los algoritmos de Needleman-Wunsch [31] y LCS [35] es que, aquellos alineamientos negativos serán fijados al valor cero, teniendo únicamente valores positivos y dividiéndose en las siguientes tareas:

1. Se determina la matriz de substitución *s(a,b)* y el esquema de penalización por huecos W_k

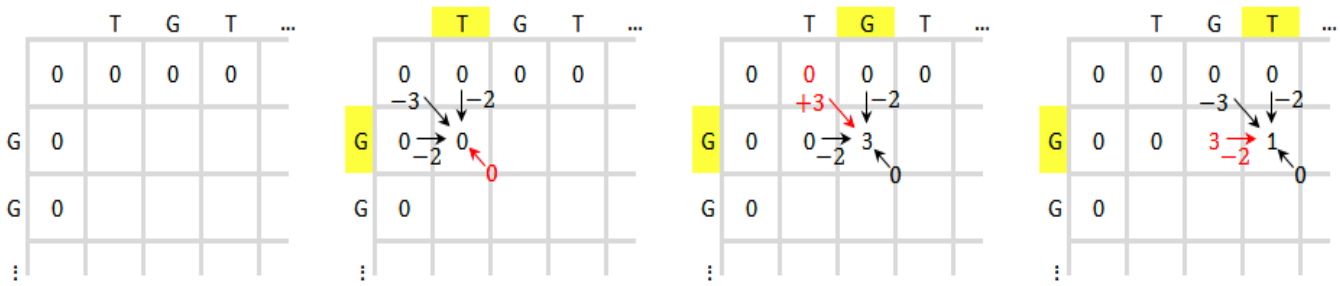

Imagen 4 - Inicialización de la matriz de puntuación

2. Se construye la matriz de puntuación H con tamaño *(n+1)(m+1)* y completar la primera fila y la primera columna con ceros.

$$H_{k0} = H_{0l} = 0$$

Ecuación 2 - Inicialización matriz de puntuación H

3. Se rellenan el resto de celdas de la matriz de acuerdo con la matriz de substitución y el esquema de penalización por huecos.

$$H_{ij} = max \begin{cases} H_{i-1,j-1} + s(a_i, b_j) \\ max_{k \geq 1}\{H_{i-k,j} - W_k\} \\ max_{l \geq 1}\{H_{i,j-l} - W_l\} \\ 0 \end{cases}$$

Ecuación 3 - Obtención matriz de puntuación H

		T	G	T	T	A	C	G	G
	0	0	0	0	0	0	0	0	0
G	0	0	3	1	0	0	0	3	3
G	0	0	3	1	0	0	0	3	6
T	0	3	1	6	4	2	0	1	4
T	0	3	1	4	9	7	5	3	2
G	0	1	6	4	7	6	4	8	6
A	0	0	4	3	5	10	8	6	5
C	0	0	2	1	3	8	13	11	9
T	0	3	1	5	4	6	11	10	8
A	0	1	0	3	2	7	9	8	7

Imagen 5 - Asignación de valores a la matriz de puntuación

4. Se procede a la marcha atrás (*trace-back*) comenzando por el valor máximo de la matriz de puntuación y finalizando cuando se encuentre el primer valor cero.

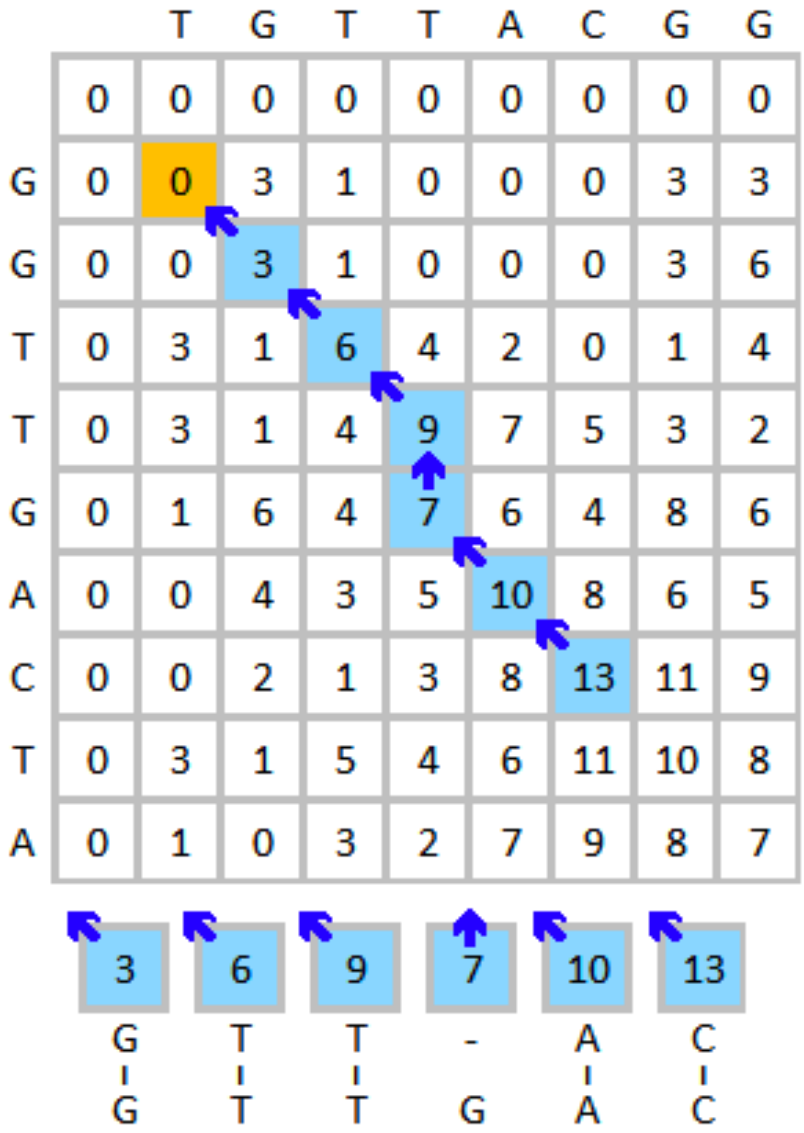

Imagen 6 - Trace-back sobre la matriz de puntuación

Como se ha podido observar en las diferentes etapas del algoritmo, la complejidad algorítmica de Smith-Waterman es $O(m^2n)$ siendo un valor de orden excesivo para un algoritmo de detección de homologías [36]. Esta problemática propició el diseño de algoritmos basados en heurística tales como BLAST.

3.4. Modelo estadístico Karlin-Altschul

Durante la década de los años 90, Samuel Karlin y Stephen Altschul publicaron lo que es hoy, la teoría estadística de alineamientos locales más utilizada en el ámbito de la bioinformática. Esta teoría está fundamentada en los siguientes pilares:

- Los valores de puntuación entre aminoácidos deben ser positivos.
- El valor de relevancia esperado sobre el alineamiento debe ser negativo.
- Cada una de las letras que componen la secuencia son independientes entre sí e idénticamente distribuidas (IID).
- No existe una limitación de tamaño para las secuencias.
- Los alineamientos existentes no deben contener huecos.

Las dos primeras consideraciones se cumplen para cualquier matriz de puntuación estimada a partir de valores reales y tal como describimos en secciones anteriores. Sin embargo, las tres siguientes presentan una serie de problemáticas debido a las diferentes dependencias biológicas inherentes a la propia secuencia. Concretamente, las secuencias tienen limitaciones de tamaño y frecuentemente son alineadas con huecos.

$$E = \frac{Km'n'}{e^{\lambda s}}$$

Ecuación 4 - Ecuación de Karlin-Altschul

Tal como indica la Ecuación 4, el número de alineamientos esperados por casualidad (*E*) durante el proceso de búsqueda en una base de datos de secuencias es una función compuesta por el espacio de búsqueda (*mn*), un valor de puntuación normalizado (λS) y una constante menor (*k*). En primer lugar, el concepto del espacio de búsqueda se define como el área comprendida por la longitud de la secuencia o secuencias de entrada (*m*) por la longitud de las secuencias que forman la base de datos (*n*). En segundo lugar, un pequeño

factor de ajuste (*k*) debe ser tenido en cuenta con el objetivo de obtener el valor de puntuación óptimo sobre alineamientos que no empiezan en las mismas posiciones. Además, partiendo de esta ecuación se pueden observar dos comportamientos claramente identificados:

- La relación existente entre el espacio de búsqueda y el número de alineamientos esperados es lineal.
- La relación entre el número de alineamientos esperado y el valor de puntuación es exponencial.

Desde un punto de vista objetivo, los alineamientos con huecos proporcionan un mayor rigor en el esquema de puntuación y ocurren con mayor frecuencia respecto a aquellos sin huecos. No obstante, este valor dependerá de las penalizaciones existentes por apertura y extensión de huecos respecto al esquema de puntuación existente. Por consiguiente, los valores de los parámetros λ, *K* y *H* estarán vinculados a su esquema de puntuación y la asociación correspondiente con las penalizaciones por apertura y extensión de hueco. Asimismo, este proceso no puede ser obtenido mediante procesos analíticos, sino que debe ser estimado empíricamente.

Como parte de la teoría de Karlin-Altschul, la distribución de puntuaciones dentro de un alineamiento sigue una distribución del valor extremo (EVD) y con un parecido bastante razonable respecto a una distribución uniforme. La Imagen 7 muestra el modelo de distribución de valor extremo o distribución de Gumbel.

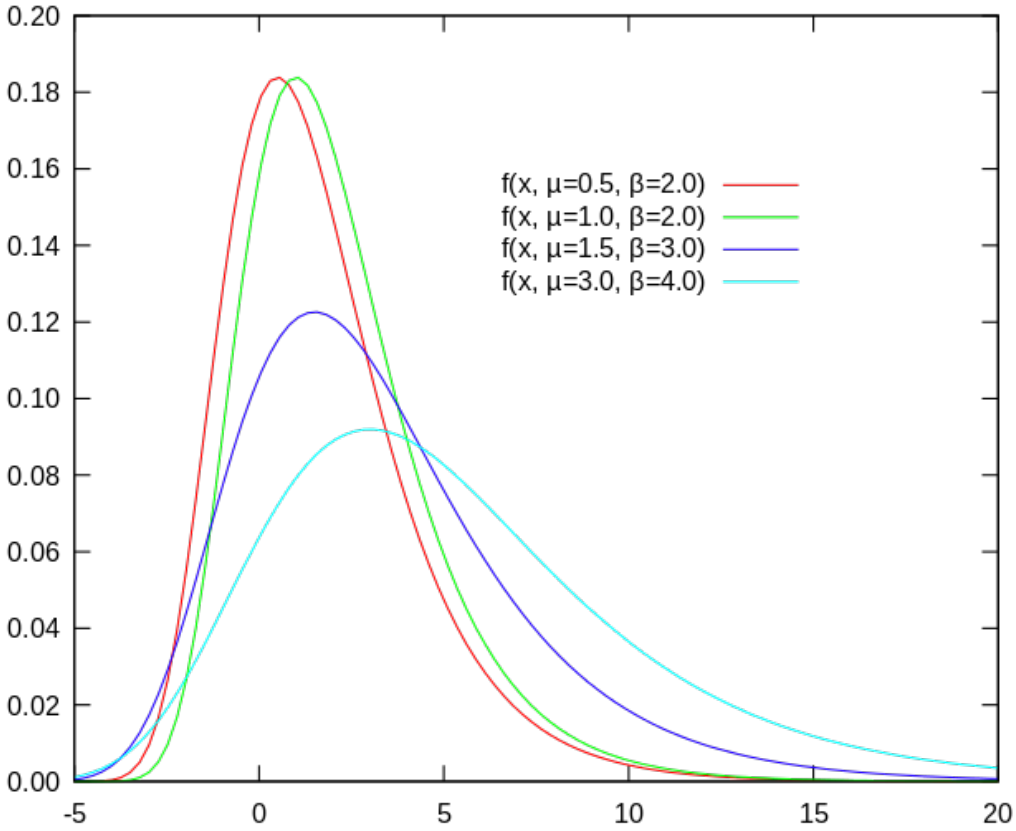

Imagen 7 - Distribución de valor extremo o Gumbel

De esta manera, el valor de la variable *H* seguirá la misma distribución mediante la siguiente ecuación de cálculo de entropía:

$$H = -\sum_{q_{ij}}^{\lambda} S_{ij}$$

Ecuación 5 - Cálculo de la entropía

Dentro de la Ecuación 5, la variable λ es inversamente proporcional al factor de escalado y las variables q_{ij} y S_{ij} son las frecuencias del par *i,j* y el valor del par *i,j* respectivamente.

En lo que respecta al espacio de búsqueda, es importante tener en cuenta que no todo puede ser analizado de manera efectiva, ya que hay una parte de dicho espacio que corresponde con el final de las secuencias. Este área, tal y como se

verá en la fase de extensión de BLAST, excede siempre el área de relevancia de la secuencia. Por consiguiente, este modelo estadístico provee una manera de calcular la longitud efectiva de cada secuencia (HSP) y representado a través de la siguiente ecuación:

$$l = \frac{\ln(Kmn)}{H}$$

Ecuación 6 - Cálculo del espacio de búsqueda

Como se observa en la Ecuación 6, la longitud esperada del HSP es dependiente del espacio de búsqueda (*mn*) y la entropía relativa al esquema de puntuación (*H*) por lo que puede variar entre búsquedas. Además, una consideración a tener en cuenta dentro de estos cálculos es la existencia de un límite inferior respecto a la longitud efectiva y fijado a *1/k*. Finalmente, el espacio de búsqueda representado mediante las variables *m'* y *n'* quedan representadas mediante las siguientes ecuaciones:

$$m' = \begin{cases} m - l & si\ l > {}^{1}/_{k} \\ m - {}^{1}/_{k} & si\ 0 \leq l \leq {}^{1}/_{k} \end{cases}$$

Ecuación 7 - Cálculo longitud efectiva secuencia entrada

$$n' = \begin{cases} n - (l * dbnumseq) & si\ l > {}^{1}/_{k} \\ n - \left({}^{1}/_{k} * dbnumseq\right) & si\ 0 \leq l \leq {}^{1}/_{k} \end{cases}$$

Ecuación 8 - Cálculo longitud efectiva base de datos

Una vez obtenidos los diferentes alineamientos, BLAST calculará un valor de puntuación normalizado para cada uno de los HSP obtenidos y mediante la siguiente ecuación:

$$S'_{nats} = \lambda S - \ln k$$

Ecuación 9 - Cálculo del valor de puntuación normalizado

Sin embargo, en el caso de existir un grupo de HSPs a agrupar y normalizar se deberán llevar a cabo otros cálculos categorizados por la posición de los HSPs dentro del espacio de búsqueda, desordenados y ordenados. La Ecuación 10 y Ecuación 11 reflejan ambas situaciones respectivamente:

$$S'_{sum} = \lambda \sum_{i=1}^{r} S_r - r \ln(kmn)$$

Ecuación 10 - Cálculo de agrupación de puntuaciones desordenadas

$$S'_{sum} = \lambda \sum_{i=1}^{r} S_r - r \ln(kmn) + \ln(r!)$$

Ecuación 11 - Cálculo de agrupación de puntuaciones ordenadas

Finalmente, y como se vio al inicio del capítulo, el valor obtenido de la Ecuación 4 se corresponde con el número de alineamientos con un determinado valor de puntuación que son esperados por casualidad. Sin embargo, existen en la actualidad otras soluciones, como por ejemplo WU-BLAST, que emplean este mismo modelo teórico pero otro tipo de retornos (*P-value*). Este retorno indica la frecuencia esperada de aparición de un determinado alineamiento y queda representado mediante la siguiente ecuación:

$$P = 1 - e^{-E}$$

Ecuación 12 - Cálculo de P-value desde E-value

$$E = -\ln(1 - P)$$

Ecuación 13 - Cálculo de E-value desde P-value

Por último y gracias a la teoría estadística descrita por Karlin-Altschul, fueron definidas las bases matemáticas de las actuales aplicaciones de secuenciación, como por ejemplo BLAST. Esta aplicación, utilizada ampliamente en el ámbito de la bioinformática, ha sido utilizada en la presente tesis doctoral como referencia para la implementación del sistema de pre-filtrado propuesto y su correspondiente evaluación de prestaciones. A continuación, se detallarán las diferentes etapas que conforman dicha aplicación de referencia.

3.5. NCBI BLAST

Durante la década de los años 80, todos aquellos algoritmos destinados al alineamiento entre secuencias estaban basados en técnicas de programación dinámica donde el consumo de memoria era muy elevado y el tiempo de procesamiento eran muy costosos. De esta manera, la aparición de algoritmos basados en técnicas heurísticas fue cobrando especial relevancia dado su mejora a nivel computacional.

Sin embargo, antes de entrar a describir los principales algoritmos basados en este tipo de técnicas heurísticas, explicaremos los conceptos de similitud y homología. Ambos conceptos son ampliamente utilizados por aquellos algoritmos basados en técnicas heurísticas. La similitud es el grado de relación entre dos secuencias de aminoácidos cuyo eje central está basado en la combinación entre identidad y conservación. El concepto de identidad se define por el grado en el cual dos secuencias de aminoácidos son invariantes. De la misma manera, el concepto de conservación se define como aquellos cambios sobre posiciones concretas de una secuencia de aminoácidos sin modificar su comportamiento químico original. Por ejemplo, la propia evolución genética es

en sí mismo un comportamiento conservativo ya que modifica un conjunto de residuos sobre una secuencia sin alterar sus propiedades bioquímicas y procesos fisiológicos. Este concepto surgió a finales de los años 60 y principios de los años 70 siendo su principal precursora, Margaret Dayhoff. Sus estudios, basados en técnicas de medición cuantitativas, supusieron la creación de las actuales matrices de substitución BLOSUM y PAM [32] y que representan las tendencias evolutivas entre substituciones de aminoácidos. Un ejemplo de estas matrices se encuentra representado en la Imagen 9.

El proceso de construcción de estas matrices comenzó mediante el alineamiento de múltiples secuencias relacionadas entre sí y su comparación a través de frecuencias sobre substituciones de valores concretos. De esta manera y tal como se observa en la Imagen 8, ciertos patrones de similitud fueron observados a través de sus propiedades químicas.

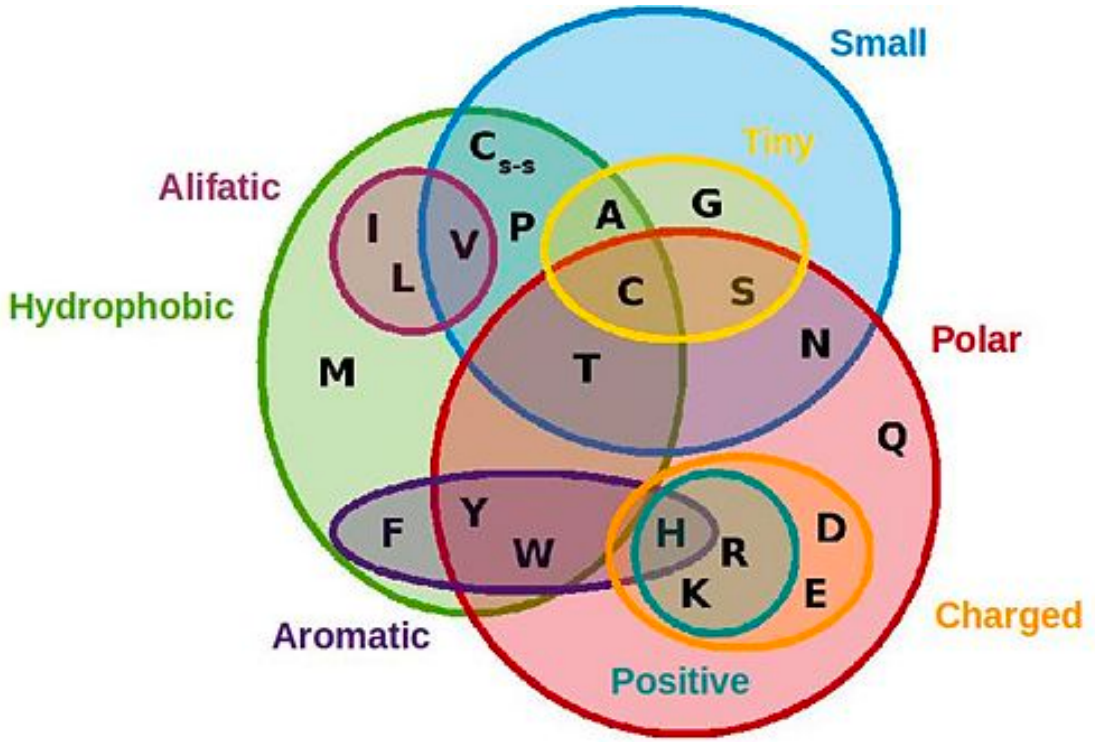

Imagen 8 - Relaciones químicas entre aminoácidos

De esta manera, Dayhoff representó la verosimilitud entre dos secuencias de aminoácidos o *lod score* a través de la siguiente relación matemática:

$$S_{ij} = \log(^{q_{ij}}/_{p_i p_j})$$

Ecuación 14 - Cálculo de versosimilitud entre aminoácidos

Como se puede observar en Ecuación 14, p_i y p_j se corresponden con las probabilidades individuales de los aminoácidos *i* y *j* y q_{ij} como su frecuencia de emparejamiento. Por último, en caso que se conociera la dirección del árbol evolutivo, su puntuación de alineamiento podría ser diferente. Sin embargo, dicho modelo fue simplificado de tal manera que los valores obtenidos entre aminoácidos se comportarían de manera simétrica.

	Ala	Arg	Asn	Asp	Cys	Gln	Glu	Gly	His	Ile	Leu	Lys	Met	Phe	Pro	Ser	Thr	Trp	Tyr	Val
Ala	4																			
Arg	-1	5																		
Asn	-2	0	6																	
Asp	-2	-2	1	6																
Cys	0	-3	-3	-3	9															
Gln	-1	1	0	0	-3	5														
Glu	-1	0	0	2	-4	2	5													
Gly	0	-2	0	-1	-3	-2	-2	6												
His	-2	0	1	-1	-3	0	0	-2	8											
Ile	-1	-3	-3	-3	-1	-3	-3	-4	-3	4										
Leu	-1	-2	-3	-4	-1	-2	-3	-4	-3	2	4									
Lys	-1	2	0	-1	-3	1	1	-2	-1	-3	-2	5								
Met	-1	-1	-2	-3	-1	0	-2	-3	-2	1	2	-1	5							
Phe	-2	-3	-3	-3	-2	-3	-3	-3	-1	0	0	-3	0	6						
Pro	-1	-2	-2	-1	-3	-1	-1	-2	-2	-3	-3	-1	-2	-4	7					
Ser	1	-1	1	0	-1	0	0	0	-1	-2	-2	0	-1	-2	-1	4				
Thr	0	-1	0	-1	-1	-1	-1	-2	-2	-1	-1	-1	-1	-2	-1	1	5			
Trp	-3	-3	-4	-4	-2	-2	-3	-2	-2	-3	-2	-3	-1	1	-4	-3	-2	11		
Tyr	-2	-2	-2	-3	-2	-1	-2	-3	2	-1	-1	-2	-1	3	-3	-2	-2	2	7	
Val	0	-3	-3	-3	-1	-2	-2	-3	-3	3	1	-2	1	-1	-2	-2	0	-3	-1	4

Imagen 9 - Matriz de substitución BLOSUM62

La definición de homología entre dos secuencias de aminoácidos se define como aquella relación bidireccional o alineamiento entre dos elementos de ambas secuencias sin reordenación, pero con la posibilidad de inserciones o borrados. Esta relación resulta en una puntuación del alineamiento que es determinado *a priori* por su relevancia biológica. Por consiguiente, todos aquellos algoritmos

heurísticos tienen por objetivo recorrer el espacio de búsqueda mediante técnicas rápidas, aunque ligeramente menos precisas, para localizar la región y relación entre secuencias con mayor verosimilitud.

Llegados a este punto, aquellas aplicaciones de detección de homologías basadas en técnicas heurísticas son consideradas la piedra angular de las líneas de investigación en genómica de altas prestaciones. El principal objetivo de estas investigaciones es reducir los tiempos de ejecución de los algoritmos de secuenciación de referencia y aumentar la precisión de los mismos.

La principal aplicación de esta categoría es BLAST (*Basic Local Alignment Search Tool*), siendo la referencia de secuenciación más utilizada en entornos bioinformáticos. Este algoritmo, creado por Stephen Altschul, Warren Gish, Webb Miller, Eugene Myers y David J. Lipman en 1990 dentro del NIH (National Institutes of Health), es altamente paralelizable [37] y está basado principalmente en métodos estadísticos con heurísticas *hit-and-extend* para proteínas y nucleótidos. A continuación, se detallan las principales fases que componen este algoritmo:

1. *Seeding*

   Como se aprecia en la Imagen 10, BLAST está basado en que, para obtener un alineamiento relevante, tienen que existir elementos en común procedentes de ambas secuencias.

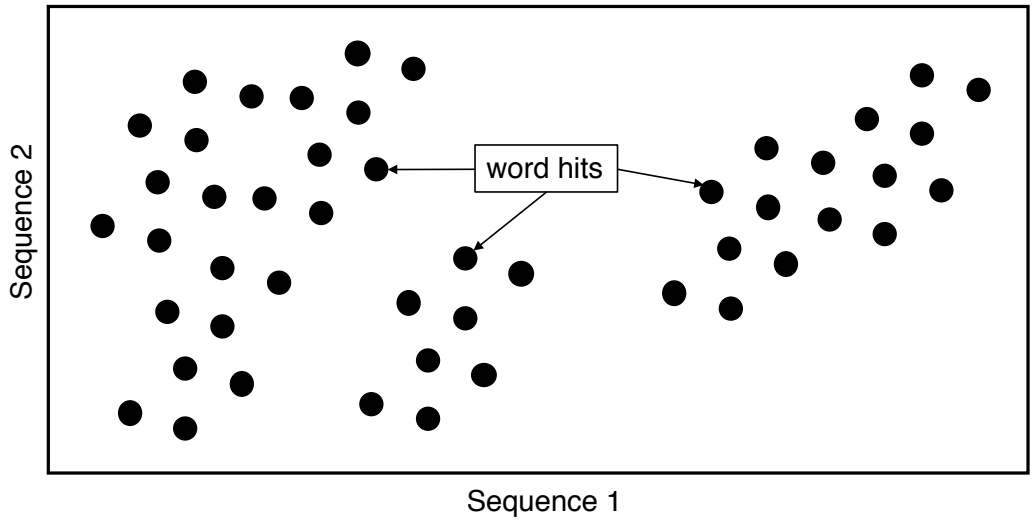

Imagen 10 - Fase seeding NCBI BLAST

Una vez identificados los elementos comunes, sólo aquellas regiones que superen un determinado número de coincidencias, introducido previamente por el usuario, serán tenidas en cuenta y, por tanto, permitirá reducir el espacio de búsqueda existente inicialmente. Sin embargo, como se puede observar en la Imagen 11, el efecto de reducción del espacio de búsqueda producido por dicho valor o sesgo de coincidencias dependerá de las matrices de substitución. Esto tendrá una repercusión directa en el equilibrio entre velocidad y precisión de los alineamientos realizados. Finalmente, y pese a que BLAST asume que, tanto aminoácidos como nucleótidos son independientes unos de otros y por tanto con la misma probabilidad de ocurrir, aquellas regiones consideradas de baja complejidad (LCR) [38] serán eliminadas de esta fase mediante procesos de enmascaramiento a nivel software. Algoritmos como SEG [39] para el caso de proteínas y DUST para el caso de nucleótidos son algunos ejemplos empleados a día de hoy para resolver dicha problemática.

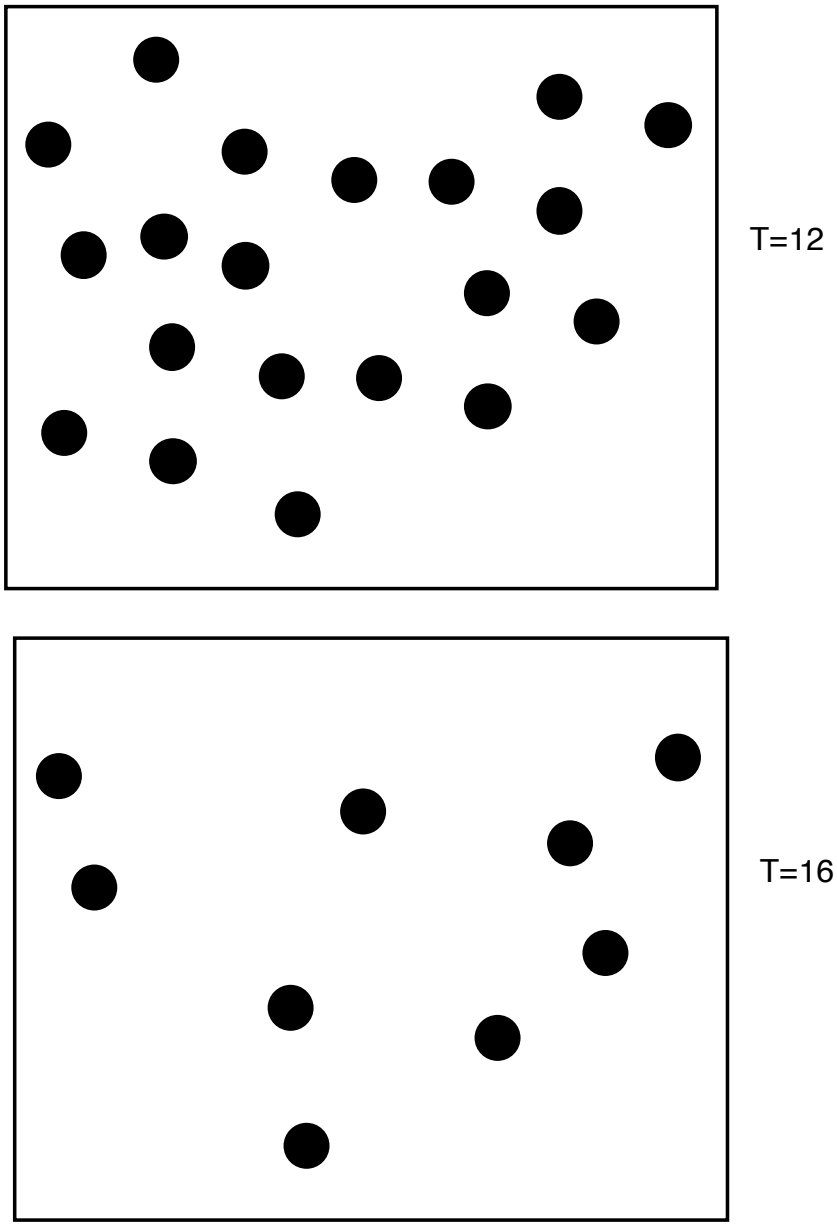

Imagen 11 - Efectos del sesgo durante la fase de seeding NCBI BLAST

2. *Extension*

   Una vez que el espacio de búsqueda ha sido reducido a través de la fase de *seeding,* se procederá a la generación del alineamiento final mediante el proceso de extensión. Como se aprecia en la Imagen 12, dicho proceso extenderá, como su propio nombre indica, el alineamiento inicial a izquierdas y/o derechas hasta alcanzar la condición de parada o *drop off score* y teniendo en cuenta los valores procedentes de las matrices de substitución, tanto positivos como negativos, durante las extensiones.

Este valor de parada se corresponde con la puntuación a partir del cual, una vez la puntuación del alineamiento sea inferior a dicho valor, se detendrá el proceso.

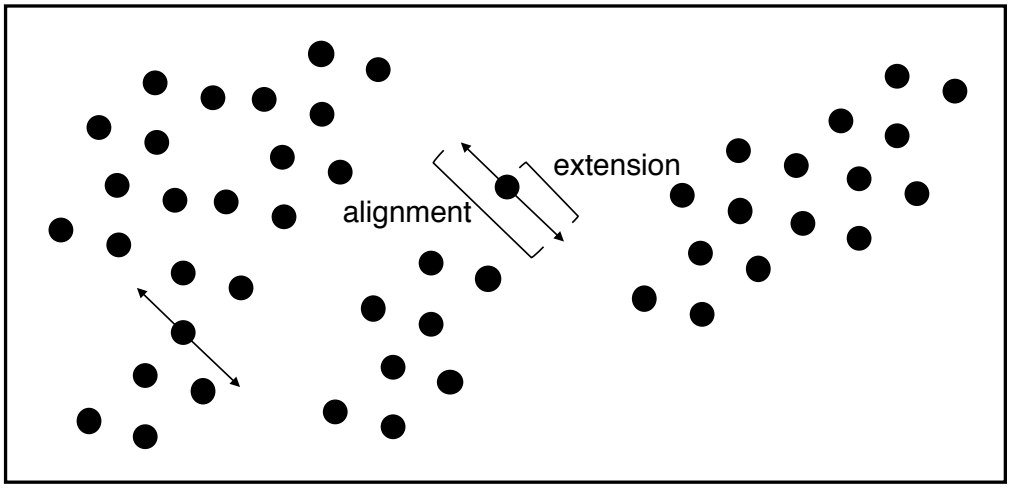

Imagen 12 - Fase extensión NCBI BLAST

Tras la detención del proceso de extensión, se llevará a cabo un mecanismo de marcha atrás o *trace-back* con el objetivo de alcanzar el alineamiento máximo obtenido. Asimismo, y con carácter opcional, se podrán considerar alineamientos con huecos, pero con la inclusión de las correspondientes penalizaciones por apertura y extensión del hueco.

3. *Evaluation*

Una vez finalizados los dos primeros procesos y creado el alineamiento resultante, se procederá a evaluar si dicho alineamiento es realmente significativo en términos estadísticos y mediante la ecuación de Karlin-Altschul. Además, se podrán unificar HSPs cercanos como un único alineamiento, considerando tanto solapamientos como huecos, y volviendo a evaluarse respecto al nuevo espacio de búsqueda resultante.

A continuación, se describe de manera más detallada las diferentes fases y tareas asociadas del algoritmo NCBI BLAST:

1. *Seeding*
    a. Como se puede observar Para cada *lmer* procedente la secuencia de entrada:
        i. Se aplica la matriz de sustitución de referencia correspondiente, como por ejemplo BLOSUM62 [40], [41] o PAM250 [42].
        ii. Se listan todas las coincidencias de *lmer* con valor superior al proporcionado por el usuario.
        iii. Se organizan aquellos *lmers* más significativos en un árbol de búsqueda.

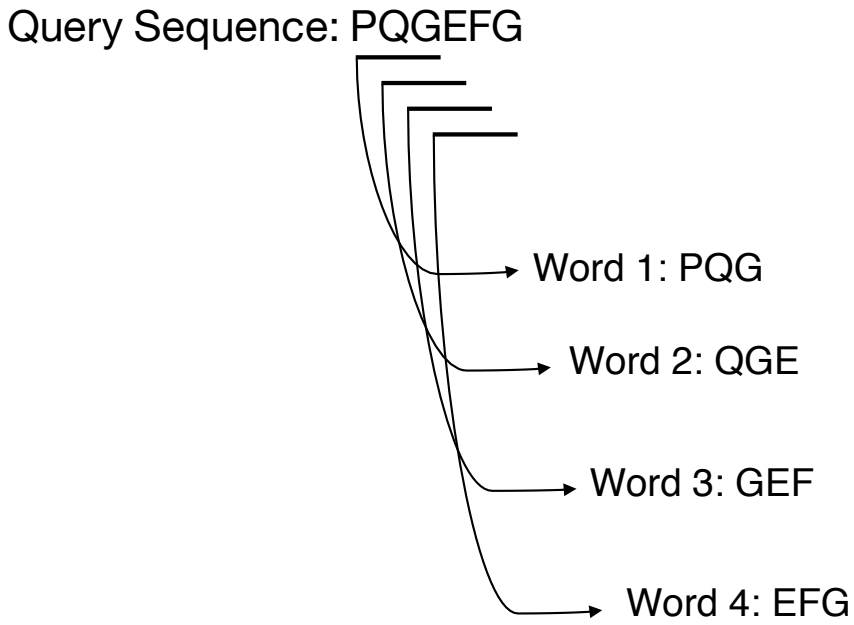

Imagen 13 - Esquema de división de secuencias por lmers

2. *Hitting*
    a. Se analiza la base de datos de secuencias en búsquedas de coincidencias exactas junto con su correspondiente valor de alineamiento máximo.
3. *Extension*

a. Como se observa en la Imagen 14, se extienden las coincidencias exactas hasta alcanzar su *High-scoring Pair* (HSP).
b. Se alarga el alineamiento entre la secuencia de entrada y la secuencia de la base de datos tanto a derechas como izquierdas y a partir de la posición donde se localizó la coincidencia exacta.
c. Se listan todos los HSPs cuyo valor de alineamiento es superior a un valor de referencia introducido por el usuario (*cutoff* score).

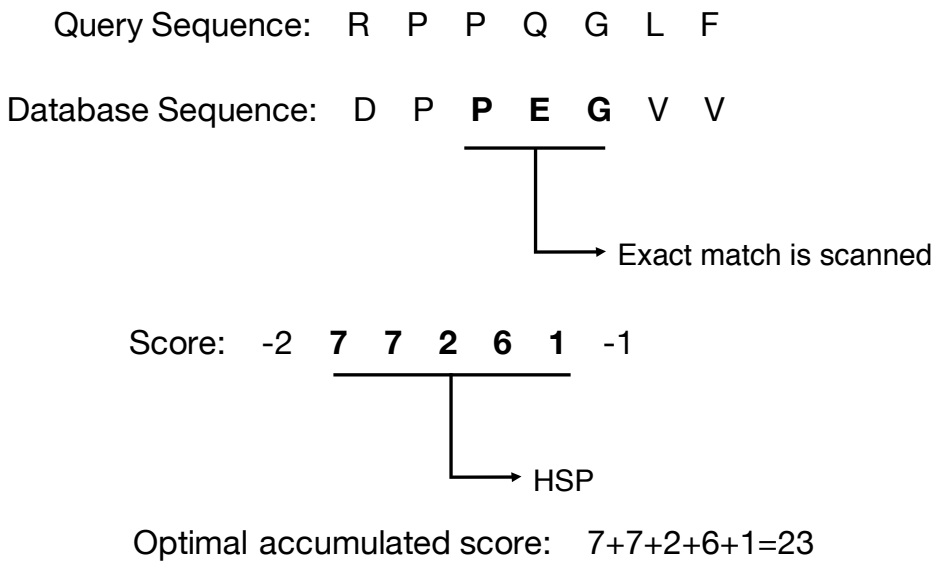

Imagen 14 - Extensión de lmers y búsqueda del HSP

4. Evaluation

a. Se evalua la relevancia del valor de alineamiento del correspondiente HSP.
    i. Como se puede observar en la Imagen 15, este proceso se basa en evaluar la relevancia estadística de cada HSP mediante la aplicación de la distribución del valor extremo de Gumbel (EVD) y la ecuación de Karlin-Altschul.

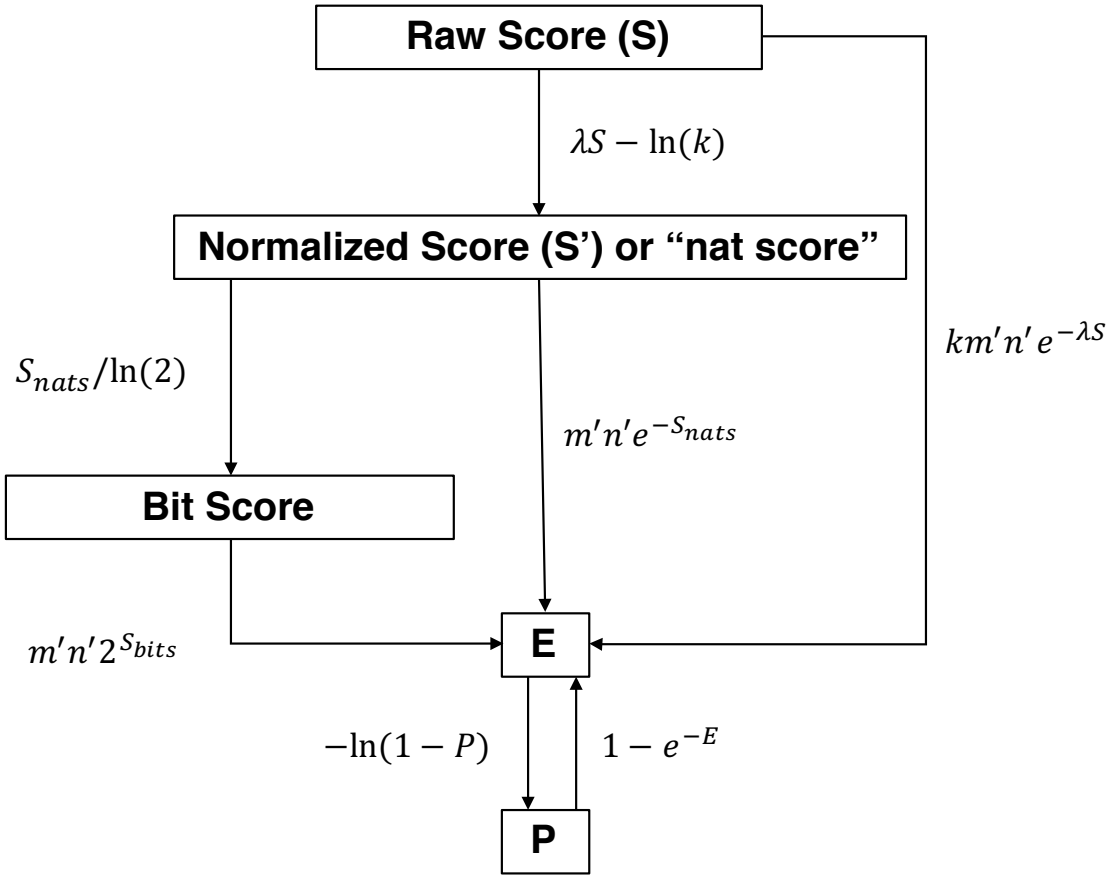

Imagen 15 - Diagrama de flujo de evaluación NCBI BLAST

5. Se unen dos o más regiones de HSPs en un solo alineamiento.
    a. Como se puede observar en la Imagen 16, este proceso de reunificación de regiones consiste en la aplicación de métodos matemáticos como *Poisson* (BLAST [5]) o *Sum-of-Scores* (BLAST2 [43]).

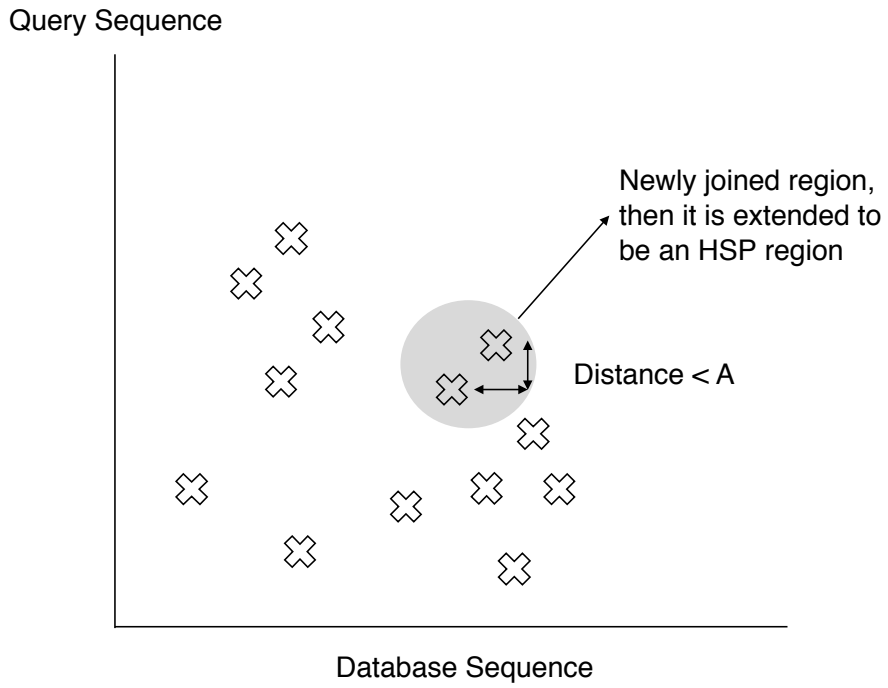

Imagen 16 - Esquema de fusión de HSPs

6. Se reporta cada alineamiento obtenido de los pasos anteriores y cuyo valor de relevancia de alineamiento (*Expect-value* o *E-value*) es menor que un valor introducido por el usuario.

3.6. FASTA

Este algoritmo basado en el alineamiento local de secuencias y descrito como FASTP por su inicial concepción para proteínas, fue creado en 1985 por David J. Lipman y William R. Pearson [44]. Su criterio de búsqueda tiene como principal objetivo tomar un determinado nucleótido o aminoácido y buscar una base de datos para encontrar similitudes entre secuencias mediante técnicas heurísticas. Dichas técnicas requieren necesariamente el parámetro *lmer*, que se define a su vez como el conjunto de aminoácidos o nucleótidos a buscar dentro de la base de datos. Por consiguiente, su longitud determinará tanto el rendimiento como la sensibilidad del algoritmo y su correcto ajuste es de especial relevancia.

Pese a que existen algunas ligeras diferencias entre proteínas (*fastp*) y nucleótidos (*fastn*), el algoritmo se divide en las siguientes fases principales:

1. Identificar regiones de alta densidad de *hits (véase Fase a sobre* Imagen 17*)*
    a. Este proceso comenzará con la búsqueda de aquellos *lmer* comunes a ambas secuencias y su posterior puntuación en diagonales para finalmente obtener las diez diagonales con mayor valor de puntuación.
2. Reprocesar las regiones obtenidas mediante matrices de substitución *(véase Fase b sobre* Imagen 17*)*
    a. Partiendo como base que el valor de puntuación de la región supere un determinado sesgo (*cutoff score*), este proceso se podrá sumar aquellos alineamientos con huecos entre medias, pero con la penalización correspondiente por hueco y el valor máximo obtenido será denominado *init1.*
3. Usar la particularización *banded* del algoritmo *Smith-Waterman* para obtener el valor óptimo del alineamiento *(véase Fase c sobre* Imagen 17*)*
    a. En este caso se toma una banda de 32 residuos centrados en la región *init1* del paso 2 para calcular el alineamiento óptimo. En el caso de proteínas, el alineamiento final es obtenido mediante la ejecución completa del algoritmo mientras que en el caso de nucleótidos se obtiene a partir del alineamiento *banded* provisto [45].

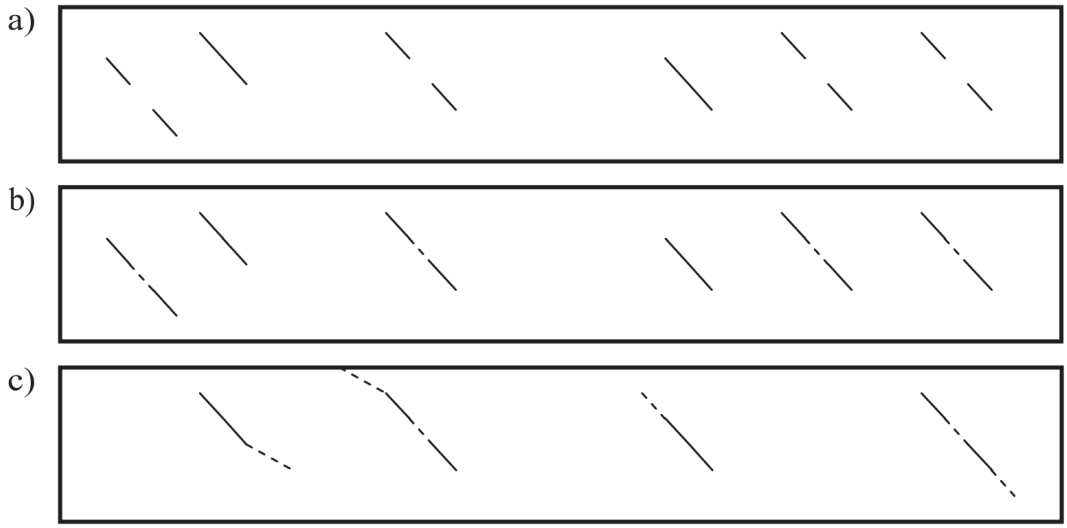

Imagen 17 - Fases del algoritmo FASTA

A diferencia del algoritmo BLAST [5], FASTA no es capaz de eliminar regiones de baja complejidad con anterioridad al alineamiento entre secuencias. Esta consideración podría ser especialmente problemática en el caso que la secuencia de consulta tuviera características especiales como mini o micro-satélites, es decir elementos de tamaño reducido con un elevado porcentaje de similitud, ya que incrementaría el valor de relevancia sobre secuencias sin similitud en la base de datos y afectando por tanto a los resultados del alineamiento. A consecuencia de ello, el programa PRSS fue añadido a la distribución de paquetes FASTA para contemplar dicha consideración [5].

Sin embargo, aparte de la propia aportación del algoritmo FASTA y sus diferentes variantes, sus autores crearon, lo que es hoy, el formato de representación de secuencias de aminoácidos y nucleótidos de referencia y denominado con el mismo nombre. Dicho formato está compuesto por los siguientes elementos:

1. Línea de definición de secuencia
2. Secuencia de aminoácidos o nucleótidos

La línea de definición de secuencia, mostrada en la primera línea de la Imagen 18, es una única línea que comienza con el símbolo obligatorio “>” seguido a su vez por un identificador único y su descripción. En particular, dicho identificador no puede contener espacios en blanco ya que se corresponde con el delimitador entre ambos campos. Asimismo, la elección de un correcto identificador es una tarea complicada dado el elevado número de bases de datos y éstas a su vez con un elevado número secuencias de aminoácidos y nucleótidos. Por consiguiente, con el objetivo de minimizar conflictos entre secuencias procedentes de diferentes bases de datos, la mejor solución es subir la secuencia a bases de datos públicas, como por ejemplo GenBank [29], DDBJ o EMBL, y utilizar sus propios valores de identificación. Sin embargo, en muchas ocasiones no existe un control detallado durante el proceso de identificación de secuencias que garantice que el valor utilizado sea único. Por este motivo y tal como se muestra en la Imagen 19, el NCBI propuso un formato de identificación propio para solucionar todos estos problemas.

```
>GXP_210035 loc=GXL_175098|sym=FAM149A|taxid=9606|spec=Homo
sapiens|chr=4|ctg=NC_000004|str=(+)|start=187065495|end=187066181|len=687|comm=Promoter
Region
GGACGGGCGTGGAAGGGTCCACGTCTTTAGTATGCATGCTTAGATCTAGCGTTCCTGTTGATGGAGTAATGGTTCTCGCA
TTGACCAGATCCGGGGCTTCATTTTTTAAACCTCATTCGTCCACTCCCCACCCCAGCCTGGTGTGCGCACCCTTTGATGG
GGCGGGGATAGGCGAGATGGTCCTGTGGTTCTCTGCCTTCTTCTGGTGAATTAAAATCCGATTTGGAAGAGAGAAGGGCA
GCCAGCACCAGTATGCACAGCCCCCGGCCCCAGAGACCCGGGAAGGAGTAGGGAGGCCGGGCCGTGCGCGGAGGAGTGGC
CGCTGGGTTGGAAACCCGGCCCGGCAGGGAGCGGGGAAGGCGCGCTTTCCCGGAGGTCGGCGCGGGGCCGGGGCCGGGGC
CGGGGCCCGGAGCGGGGATGGGCGGGCGCAGCCGGGATTAGCTGGCGGGCGAGGGCGCAGCGCAGGGAGGAGGAGGGGAG
GCGGCGCCGGCGCGGGCGGGGCGGAGGATCTGGAGAGGGAAGGGGCGTGCGAGCCCCGCGGACCCCGGGCGCGCCCGGGC
CGCCTGAGCTGGGCCAGCCGCGCGGCGGGCGCGGGCGCGGGCGCGGGCGCGGGCGGGTGGGGAGCCCCAGCCCC
GGGGCCGCGGGGGCGCGTGACCGGCTGTCTGCGTGGGGCCCGCGCGC
```

Imagen 18 - Formato FASTA

En segundo lugar y tal como se observa en las líneas restantes de la Imagen 18, se encuentra la línea o líneas correspondientes con la secuencia de aminoácidos o nucleótidos. En la mayoría de los casos, existe una convención de representación de la secuencia en líneas de longitud comprendida entre 50 y 80 caracteres por línea. El formato esperado para la representación de secuencias es a través de los códigos estándar IUB/IUPAC para aminoácidos y ácidos nucleicos, con las siguientes excepciones:

- Se aceptan letras minúsculas y se mapean a mayúsculas

- Un único guion o raya puede usarse para representar un hueco
- No se admiten dígitos numéricos, pero se utilizan en algunas bases de datos para indicar la posición en la secuencia.

```
>gi|5524211|gb|AAD44166.1| cytochrome b [Elephas maximus maximus]
LCLYTHIGRNIYYGSYLYSETWNTGIMLLLITMATAFMGYVLPWGQMSFWGATVITNLFSAIPYIGTNLV
EWIWGGFSVDKATLNRFFAFHFILPFTMVALAGVHLTFLHETGSNNPLGLTSDSDKIPFHPYYTIKDFLG
LLILILLLLLLALLSPDMLGDPDNHMPADPLNTPLHIKPEWYFLFAYAILRSVPNKLGGVLALFLSIVIL
GLMPFLHTSKHRSMMLRPLSQALFWTLTMDLLTLTWIGSQPVEYPYTIIGQMASILYFSIILAFLPIAGX
IENY
```

Imagen 19 - Formato de identificación NCBI

Partiendo de estas consideraciones, el formato de representación FASTA tenía una serie de problemas, como por ejemplo la variabilidad de las líneas de descripción de secuencias, compresión de datos o la disponibilidad de la taxonomía. A consecuencia de ello, aparecieron formatos alternativos como HUPO-PSI [46], FASTQ [47], SAM o BAM [48] para solucionar dichas problemáticas.

3.7. Modelos predictivos

En la actualidad, existe una nueva línea de investigación que está cobrando especial relevancia dentro del ámbito de la bioinformática y los modelos de secuenciación genómica de nueva generación, la aplicación de modelos predictivos a la detección remota de homologías. Este modelo de detección es especialmente relevante en aquellos casos donde se desea estudiar las estructuras y funciones de aquellas secuencias con muy bajo porcentaje de verosimilitud, es decir, aproximadamente menor a un 30% basado en estudios experimentales [49].

Este modelo de detección de homologías ha sido estudiado en numerosas ocasiones y desde diferentes puntos de vista. De esta manera, se ha podido realizar una categorización en función de tres grupos principales:

- Método tradicional o basado en el alineamiento de secuencias.
- Método discriminativo.
- Método basado en técnicas de *ranking*.

El primer modelo de detección de homologías es el método tradicional en el cual, dado su esquema de detección original, presentaría bastantes limitaciones ya que implicaba bajar sus correspondientes umbrales de sensibilidad y, de esta manera, degradar su rendimiento de manera significativa. Sin embargo, con el objetivo de mitigar esta limitación, se empezó a considerar que la información procediera de un conjunto de secuencias o *multiple sequence alignments* (MSA) [50] en lugar de una secuencia individual. De esta manera, el alineamiento sobre modelados de secuencias fue cobrando especial relevancia por su buen rendimiento y precisión en este tipo de escenarios. A continuación, se presentan las tres líneas de investigación principales, ordenadas de menor a mayor rendimiento, que han sido llevadas a cabo hasta la fecha:

- Métodos basados en alineamientos por secuencias (*two-sequence aligment*)
    - PSI-BLAST [51] o IMPALA [52]
- Métodos basados en alineamientos por modelados de secuencias (*profile-profile alignment*)
    - COMPASS [53], FFAS [54], SPARK-X [55] o COMA[56]
- Métodos basados en cadenas de Markov ocultas (*profile*-HMM)
    - Hmmer [57] o HHblits [58]

En segundo lugar, se encuentran los métodos discriminativos donde, mediante modelos de clasificación basados en técnicas de aprendizaje automático tales como *Support Vector Machines* (SVM), *Neural Networks* (NN) o *Random Forests* (RF), se puede llegar a clasificar una nueva proteína dentro de una super-familia ya existente. Algunas propuestas tales como SVM-PDT [59] o SVM-DR [60], son un claro ejemplo de este tipo de métodos donde se tiene por objetivo encontrar

vectores de patrones comunes mediante los alineamientos de pares de secuencias. Sin embargo, este tipo de aproximación basado en reconocimiento de patrones requiere de un entrenamiento previo con un conjunto de muestras conocidas por lo que todas aquellas proteínas desconocidas no podrán ser clasificadas.

Por último, se encuentran los métodos basados en técnicas de *ranking*. Este tipo de técnicas, a semejanza del método tradicional basado en el alineamiento de secuencias, está basado en un valor de puntuación estimado por encima de un umbral predeterminado. Este valor se define como la distancia de homología entre secuencias dentro del espacio de patrones comunes obtenido previamente mediante una fase de aprendizaje. Dentro de este ámbito, las principales soluciones propuestas hasta la fecha se han basado, o bien en el algoritmo PageRank [61] desarrollado por Google [62], o bien mediante la combinación de diversas técnicas de *ranking* mediante *Learning-to-rank* (LTR) [22]. En este caso y de manera similar al caso anterior, la necesidad de un aprendizaje previo implica que todas aquellas secuencias desconocidas no formarán parte del algoritmo de *ranking* y, por tanto, no serán alineadas correctamente.

Tal como se ha podido observar a través de las diferentes categorías y soluciones existentes en la actualidad, el modelado de proteínas mediante la información evolutiva de su perfilado, hace posible llevar a cabo alineamientos de secuencias eficientes, precisos y sin implicar *a priori* tecnología de procesamiento de altas prestaciones.

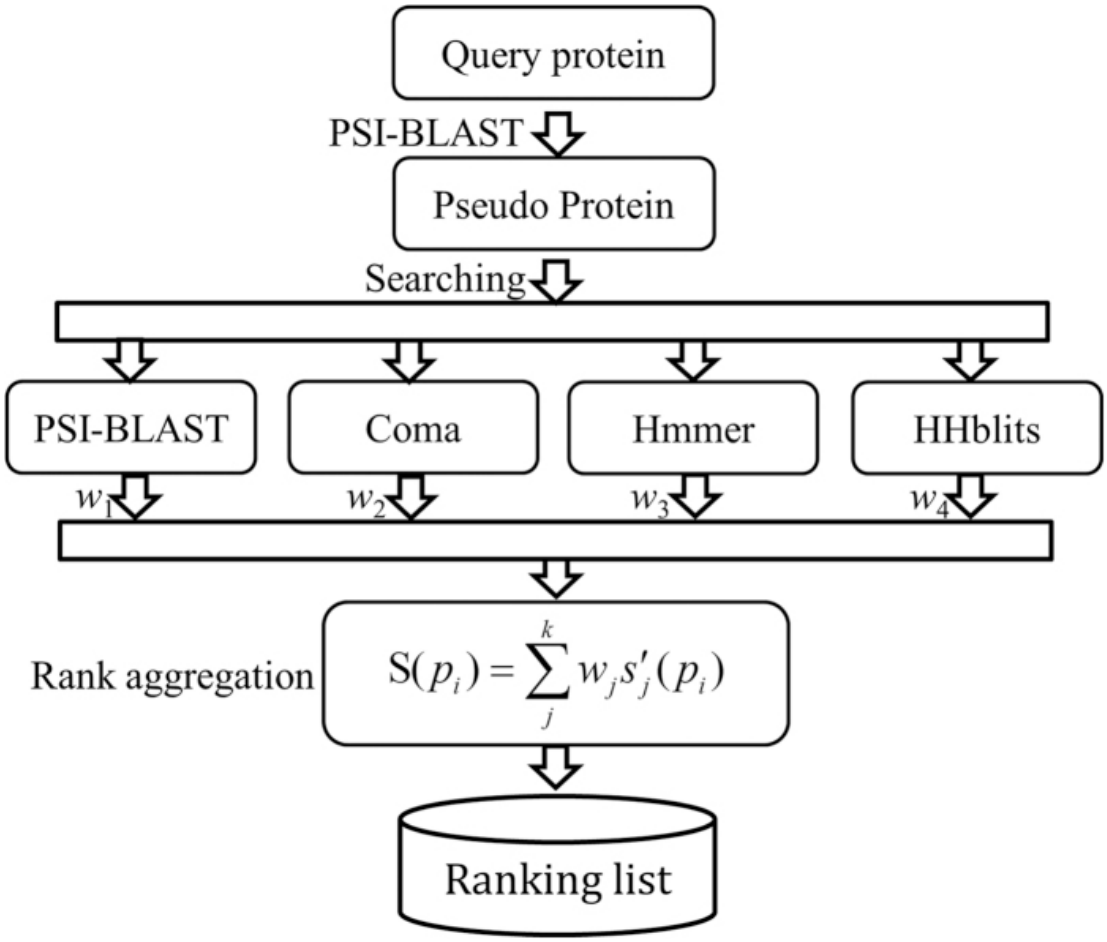

Imagen 20 - Arquitectura dRHP-PseRA

La Imagen 20 se corresponde con la arquitectura de la solución dRHP-PseRA donde confluyen técnicas basadas en algoritmos de *ranking* y técnicas tradicionales donde, una vez llevado a cabo el proceso de modelado de proteínas, se podrá utilizar un predictor común obtenido a partir del proceso *rank aggregation* sobre los algoritmos PSI-BLAST [51], Coma [56], Hmmer [57] y HHblits [58]. Finalmente, se obtendrá un listado a modo de ranking ponderado mediante el valor del predictor obtenido como elemento clave de relevancia entre secuencias.

Tras la descripción cronológica del presente estado del arte, se ha venido observando la vertiginosa evolución que han experimentado los algoritmos y aplicaciones de secuenciación basados en detección de homologías durante los últimos 30 años. Esta aceleración no ha venido determinada por la propia naturaleza evolutiva de los diferentes genomas sino por el descubrimiento de

nuevas familias, o variantes sobre otras ya existentes, que han aumentado de manera considerable el volumen de las actuales bases de datos de secuencias.

En lo que respecta al diseño de las aplicaciones de secuenciación basadas en detección de homologías, la tendencia seguida durante todos estos años se ha basado en la reducción de tiempos de ejecución afectando mínimamente en la precisión de los resultados. De esta manera, las técnicas de programación dinámica basadas en alineamientos globales han evolucionado hacia técnicas heurísticas o modelos predictivos basados en alineamientos locales.

La actual tesis doctoral sigue la misma línea evolutiva seguida por las aplicaciones de secuenciación basadas en detección de homologías, pero con la inclusión de los sistemas de computación de altas prestaciones, que se detallarán en el siguiente capítulo, para garantizar la escalabilidad, precisión y rendimiento frente al incremento de secuencias durante los próximos años. Además, esta propuesta, de carácter innovador, ha puesto el foco en aislar su desarrollo respecto al algoritmo o aplicación de detección de homologías utilizado.

Capítulo 4 Sistemas altas prestaciones

Este capítulo presenta la segunda parte el estado del arte de la presente tesis doctoral, revisando los principales sistemas de computación de altas prestaciones existentes en la actualidad. Asimismo, se evaluarán sus principales ventajas e inconvenientes respecto al sistema propuesto.

4.1. Sistemas multicore

Un procesador *multicore* se define como el sistema de procesamiento o circuito integrado compuesto por dos o más *cores* independientes. Cada uno de estos *cores* es integrado dentro de un circuito integrado propio, también conocido como multiprocesador (CMP) [63], y siendo en la actualidad, pieza clave dentro de los sistemas de computación utilizados en la industria gracias a su capacidad de llevar a cabo una gran cantidad de operaciones, tanto a nivel gráfico como matemático, y todo ello a un coste muy reducido. Sin embargo, al mismo nivel que sus homólogos tecnológicos y que describiremos a continuación, los procesadores *multicore* han sufrido una vertiginosa evolución desde la mejora de frecuencias del procesador para ejecutar instrucciones más rápidamente hasta implementar nuevas etapas de *pipelining* que permitan aumentar el paralelismo de tareas. Además, la diferencia de velocidades y evoluciones entre procesadores y memorias han llevado a los procesadores a pasar la mayor parte de su tiempo a esperar a la memoria para proporcionarle datos y haciendo todas estas evoluciones poco efectivas.

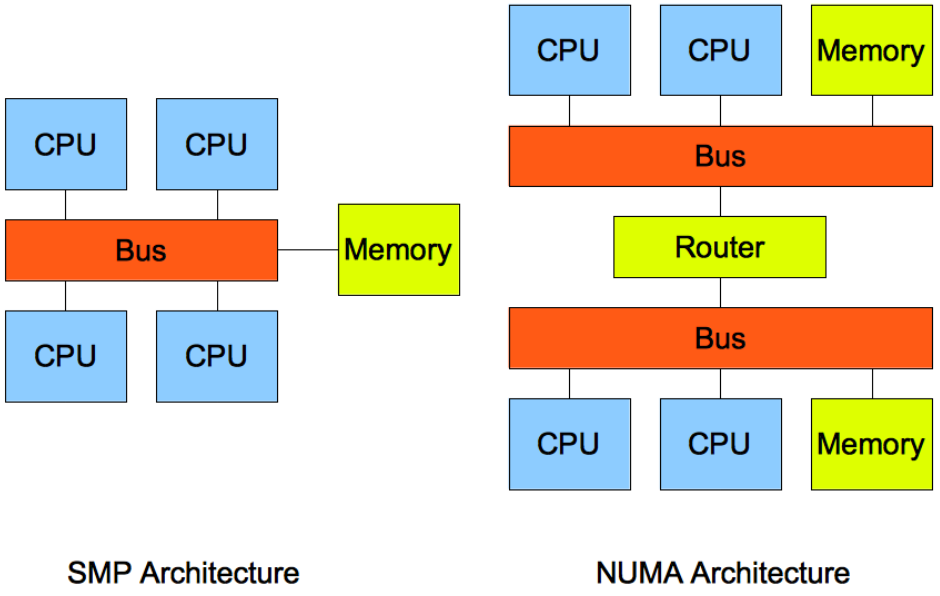

Imagen 21 - Arquitecturas SMP vs. NUMA

Como se puede observar en la Imagen 21 y antes de los sistemas *multicore*, existían dos grandes arquitecturas, *Symmetric Multiprocessor* (SMP) [64] y *Non Uniform Memory Access* (NUMA) [65], donde la principal diferencia entre ambas arquitecturas se centraba en la gestión de memoria, centralizada o distribuida por procesador. Por consiguiente, estas diferencias tienen una repercusión directa sobre el rendimiento y la escalabilidad de los sistemas *multicore*.

Finalmente, esta constante evolución y partiendo de teorías como la ley de Moore o Dennard, ha supuesto el fin de las arquitecturas RISC y el inicio de los sistemas de altas prestaciones.

4.2. GP-GPU

Graphics Processing Unit (GPU) son las siglas correspondientes al elemento hardware encargado del procesamiento gráfico y que puede ser encontrado actualmente en cualquier ordenador personal (PC), teléfono móvil o tableta. Gracias a su arquitectura, una GPU es capaz de ejecutar miles de millones de

instrucciones por segundo, tanto en entornos gráficos como no gráficos. Aquellas GPU utilizadas para entornos no gráficos son denominadas *General Purpose GPU* (GP-GPU), es decir, GPU de propósito general. El rendimiento obtenido por este tipo de GPU ha convertido este hardware como parte esencial de centros de procesamiento de altas prestaciones o *High Performance Computing Centers* (HPC). Concretamente, algunos fabricantes de supercomputadores han incluido GP-GPU en su interior como una vía alternativa al procesamiento dentro del procesador (CPU) de la estación de trabajo. Un ejemplo de ello es el supercomputador Cray XK7 con NVidia GPU en su interior.

Actualmente, existe una gran variedad de arquitecturas y modelos de GPU. NVidia y AMD son los fabricantes con mayor prestigio y cuota de mercado. Muchas investigaciones se han llevado a cabo para evaluar, dentro del amplio abanico de tecnologías existentes, aquella que proporcione un mayor rendimiento. Sin embargo, dados los numerosos avances que aparecen diariamente resulta especialmente complicado decantarse por una tecnología o por otra. A continuación, se procederá a explicar en detalle la arquitectura de una GP-GPU y sus principales elementos que hacen de ella, una tecnología de vanguardia y altas prestaciones en procesamientos costosos y donde la paralelización de tareas cobra una especial relevancia.

En la mayoría de los casos, una GPU es un elemento externo al microprocesador. Por consiguiente y tal como muestra la Imagen 22, tanto GPU como CPU están interconectados a través del *Peripheral Component Interconnect Express* (PCIe). Este tipo de interconexión implica copiar datos desde la CPU a la GPU antes de llegar a usarlos, pudiendo convertir una GPU en un elemento ineficiente si esta copia de información llevara una gran cantidad de tiempo respecto al tiempo de procesamiento. La arquitectura de GPU de NVidia está basado en un gran número de *Streaming Processors* (SP) y agrupado en *Streaming Multiprocessors* (SM). Los SP son pequeños procesadores de datos capaces de llevar a cabo operaciones entre enteros y de precisión simple. Asimismo, los SM contiene también las unidades de punto

flotante, registros, cache L1 y memoria compartida. Cada SM comparte estos recursos entre otros SP. De manera similar, cada SM comparte una caché L2 y una memoria global entre el resto de SM.

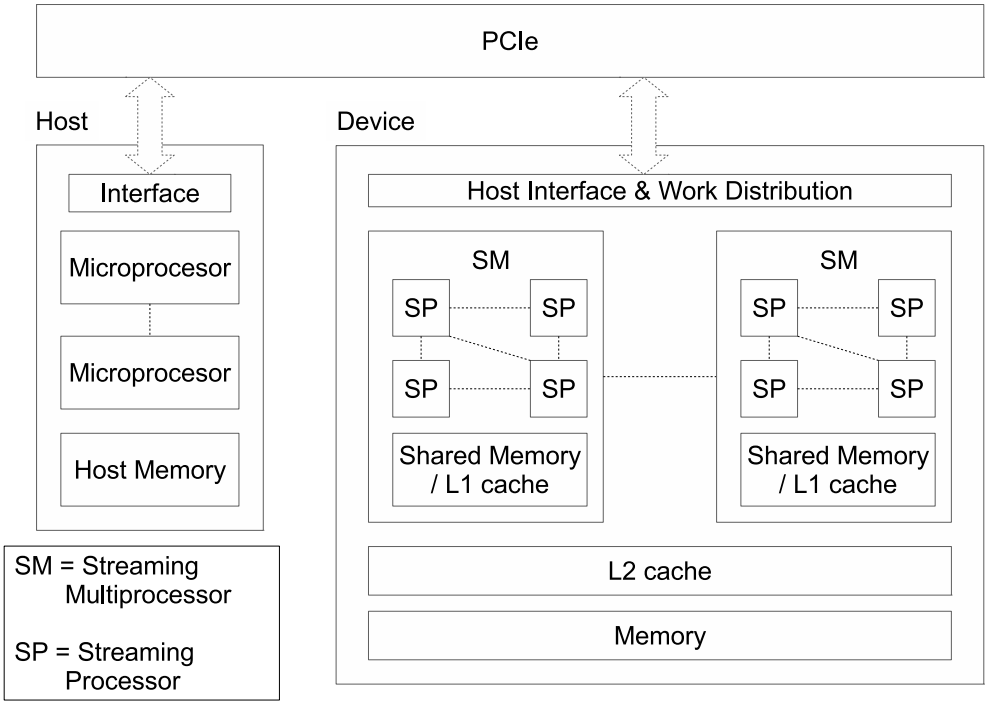

Imagen 22 - Arquitectura GP-GPU

Esta tecnología se encuentra en continua evolución. Un ejemplo de esta evolución es la transición entre la arquitectura Fermi a la arquitectura Kepler de NVidia donde se produjo una evolución de 16 SM, cada uno con 32 SP, hasta 15 SM, cada uno con 192 SP y 64 DPU. Todo ello dentro de la misma área de trabajo que es la GPU.

El modelo de programación de NVidia CUDA, representado a través de la Imagen 23, proporciona la posibilidad de programar funciones paralelizables que serán ejecutadas posteriormente por la GPU. Cada una de estas funciones es denominada CUDA *kernel*. Cada *kernel* es una parte dentro un código fuente más completo, generalmente escrito en ANSI C, y el cual puede ser ejecutado en paralelo junto con otros *kernels*. El número de posibles ejecuciones en

paralelo dependerá de los recursos requeridos por la aplicación y cuántos de ellos estén disponibles en el momento de la ejecución. Cada *kernel* es ejecutado dentro de un *grid*. Un *grid* está compuesto por un conjunto de bloques los cuales están definidos como estructuras de datos en 1, 2 o 3 dimensiones. De hecho, cada bloque está compuesto por un conjunto de hilos o *threads* que pueden a su vez estar definidos en estructuras de datos de 1, 2 o 3 dimensiones. Cada hilo será ejecutado dentro de un SP y cada bloque en un SM. Por lo que respecta a la arquitectura previamente descrita, diferentes hilos en el mismo bloque tienen la capacidad de compartir bloques de memoria de manera muy eficiente y sin tener que acceder a la memoria global de la GPU.

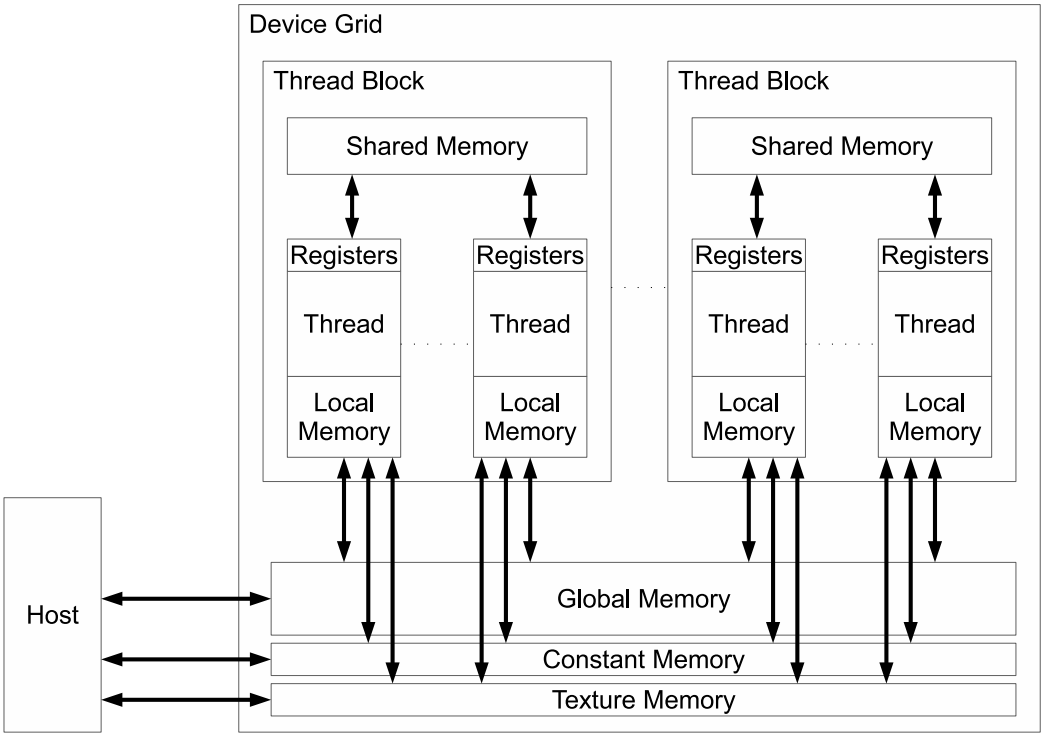

Imagen 23 - Arquitectura detallada GP-GPU

Con el objetivo de obtener un óptimo rendimiento, todo programador debe asegurarse que cada hilo de ejecución no debe generar una divergencia excesiva, ya que en caso contrario provocará una serialización de las ejecuciones entre hilos del mismo bloque. Asimismo, todo programador deberá tener en cuenta el número total de hilos disponibles y su distribución entre

bloques al igual que la cantidad de memoria compartida empleada por cada hilo de ejecución.

De manera alternativa al modelo de programación de NVidia CUDA, se encuentra OpenCL. Este modelo de programación desarrollado por Khronos se asemeja a su homólogo en que es una extensión de bajo nivel escrita en C/C++ para computación heterogénea y sobre GPU. De la misma manera que con NVidia CUDA, existe una parte de código secuencial en la aplicación (*kernel*) que es ejecutada de manera individual por todos los hilos disponibles en la GPU. Esta parte de código está desarrollada empleando un pequeño subconjunto de ANSI C para GPU.

Por último, las GPU han sido especialmente diseñadas para resolver problemáticas de procesamiento de datos en paralelo. Por consiguiente, con el objetivo de obtener una paralelización óptima, se deberán adoptar cuidadosamente los esquemas de optimización de memoria de tal manera que se empleen de manera eficiente las tres capas existentes: registros, memoria compartida y memoria global.

4.3. FPGA

Field Programmable Gate Arrays (FPGAs) son una particularización a nivel de diseño de circuitos integrados con la capacidad de configurarse mediante un conjunto de memorias tras la fabricación del chip [66]. Esta arquitectura es representada en la Imagen 24. Una vez configurada, su programación puede ser modificada en cualquier momento desde lenguajes de programación de alto nivel, *Hardware Description Language* (HDL) y un proceso de compilación compuesto de las siguientes fases:

1. Síntesis
2. Mapeo de los diferentes componentes
3. Enrutamiento de los diferentes procesos

De esta manera, FPGAs permiten a los desarrolladores crear un gran abanico de componentes digitales, tanto a nivel individual como combinado, y optimizarlos a lo largo del tiempo acorde con la evolución natural de la tecnología. Además, se postulan como el nexo de unión entre tecnologías más estrictas, como por ejemplo *Application-Specific Integrated Circuits* (ASICs), y las más extendidas como los microprocesadores.

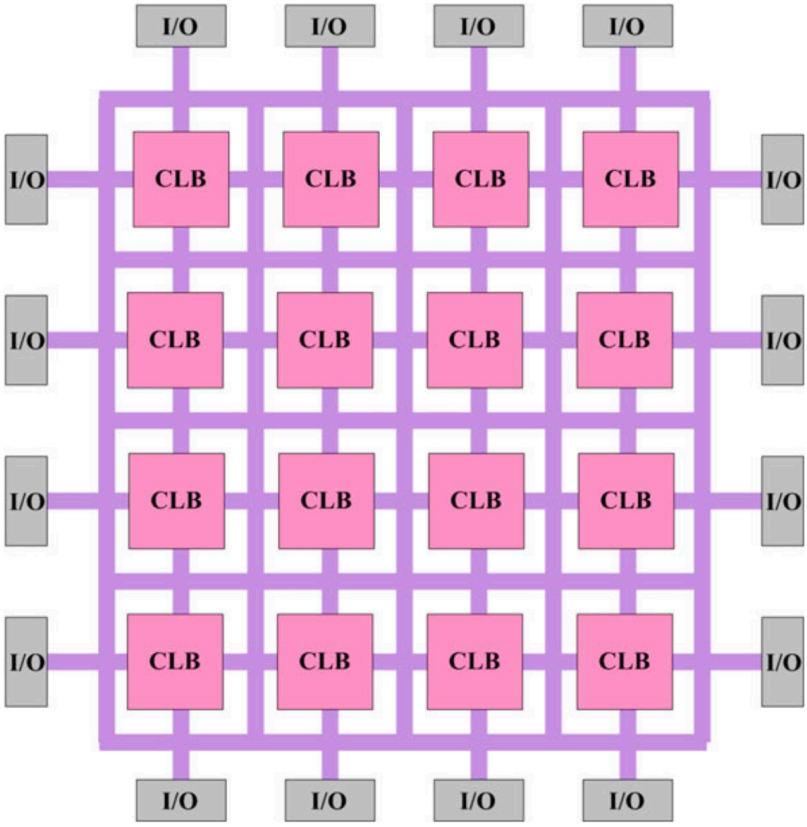

Imagen 24 - Arquitectura FPGA

En términos de rendimiento, un diseño de FPGA es equivalente a un diseño ASIC desde la perspectiva temporal de la frecuencia de reloj, por lo que su comportamiento es extremadamente eficiente. Además, las posibilidades de interconexión con otras tecnologías hardware son bastante extensas, pudiendo funcionar de manera aislada como coprocesador conectado a la CPU [67] o conectado a través del puerto PCIe.

De la misma manera que ocurre con el diseño en ASICs, cuanto más complejo es el sistema que se conecta a la FPGA, más complicado puede resultar tanto el diseño como el control de los diferentes periféricos existentes. Esta consecuencia tiene efectos directos a nivel de programación sobre los lenguajes HDL, ya que aumentará su ciclo de vida convirtiéndolo largo y costoso. Sin embargo, a lo largo de esta última década, todos estos lenguajes de programación de alto nivel (HLL) han sufrido una vertiginosa evolución con el fin de reducir este ciclo de vida, aunque por el momento con algunas dependencias con el propio fabricante de la FPGA [68], [69].

4.4. Trabajos previos

Como se introdujo en el capítulo anterior, muchas investigaciones han sido llevadas a cabo para mejorar los algoritmos de detección de homologías y, en especial, aquellos más utilizados dentro del ámbito de la bioinformática, como por ejemplo la aplicación NCBI BLAST [70] o Smith-Waterman [3], mediante técnicas teóricas o aproximaciones de altas prestaciones. En esta sección pondremos el foco de atención en las diferentes tecnologías de altas prestaciones utilizadas y sus aportaciones dentro de este ámbito.

El contexto de los multiprocesadores es uno de ámbitos que más ha evolucionado en las últimas décadas mejorando en niveles tan diversos como la frecuencia de reloj para llevar a cabo instrucciones en el menor tiempo posible o mayor número de *cores* con el fin de poder paralelizar tareas según sean necesarias. A este nivel, el *National Center for Biotechnology Information* (NCBI) implementó una versión propia de BLAST, con las mismas fases y esquema del algoritmo original, pero utilizando POSIX *threads* [70], también conocidos como *pthreads*. Esta variante tiene la gran ventaja que puede ser usado sobre hardware de propósito general, pero una vez alcanzado un cierto número de procesadores y de acuerdo con la ley de Amdahl, puede llegar a convertirse en una solución costosa. Soluciones como BLAST+ [71] o BLAST++ [72] son

algunos de los ejemplos más conocidos y significativos de mejoras a nivel de implementación de las diferentes fases del algoritmo original y que permiten explotar, de manera más eficiente, la potencia computacional de los multiprocesadores actuales.

Tal como se muestra en la Imagen 25, las diferentes fases que componen esta variante son idénticas al algoritmo original. Sin embargo, tanto la fase de *scanning* como la fase de *trace-back* han sido re-implementadas para un uso más eficiente y con mayores prestaciones. En particular, durante la fase de extensión de BLAST+ no se almacenan ni la inserciones o borrados de huecos ni las coincidencias de nucleótidos o aminoácidos, reduciendo así los requerimientos de memoria y su consumo de CPU.

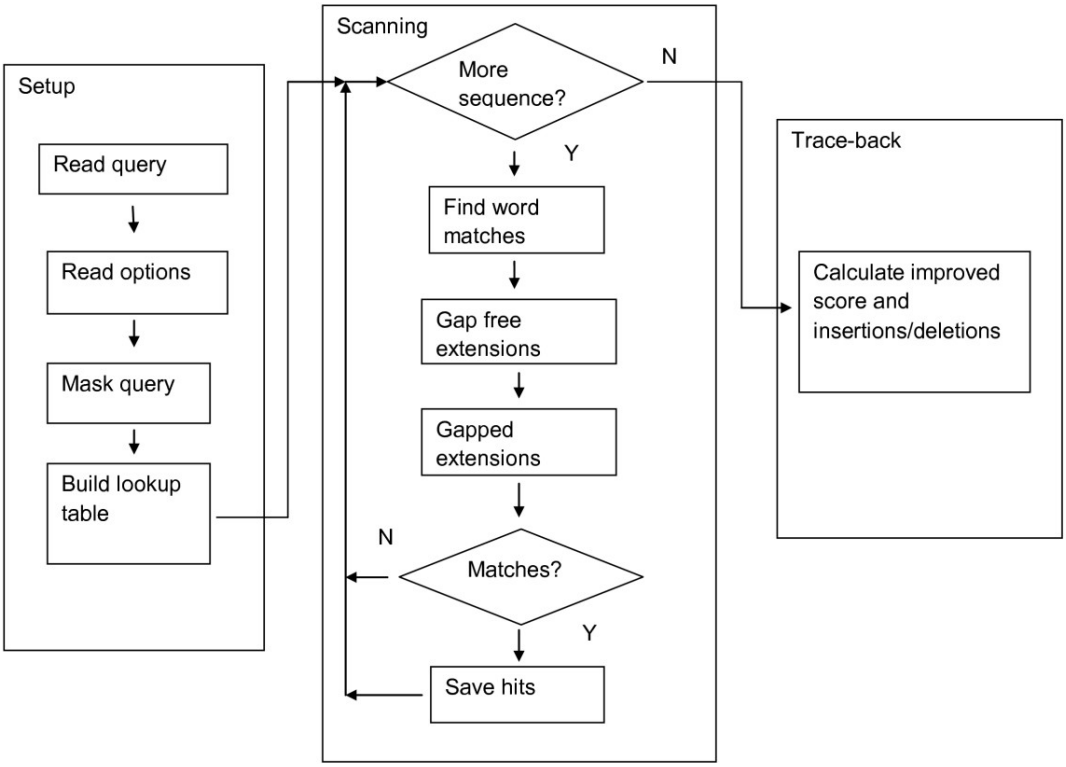

Imagen 25 - Arquitectura BLAST+

En segundo lugar y por lo que respecta a la segunda variante BLAST++, pone su foco de mejora en el procesamiento de secuencias por lotes aprovechando

las estructuras comunes existentes en memoria y un procesamiento paralelo más optimizado.

Sin embargo, el punto de vista de multiprocesadores no es el único existente desde la aproximación de procesadores convencionales, también existe la perspectiva de *cluster.* Un *cluster* se define como un conjunto de ordenadores que son capaces de llevar tareas intensivas de cálculo de manera unificada. Asimismo, su arquitectura está diseñada para soportar tolerancia a fallos y un elevado nivel de escalabilidad. A este nivel, existen en la actualidad dos tipos diferentes de *cluster*, homogéneos o conjunto de estaciones de trabajo con la misma arquitectura o heterogéneos con estaciones de trabajo procedentes de diferentes arquitecturas, a este segundo modelo también se le denomina *beowulf*.

Desde el ámbito de la bioinformática, existen varias implementaciones que explotan el paralelismo del algoritmo como MPIBLAST [73] basado en el lenguaje de programación de paso por mensajes *Message Passing* Interface (MPI).

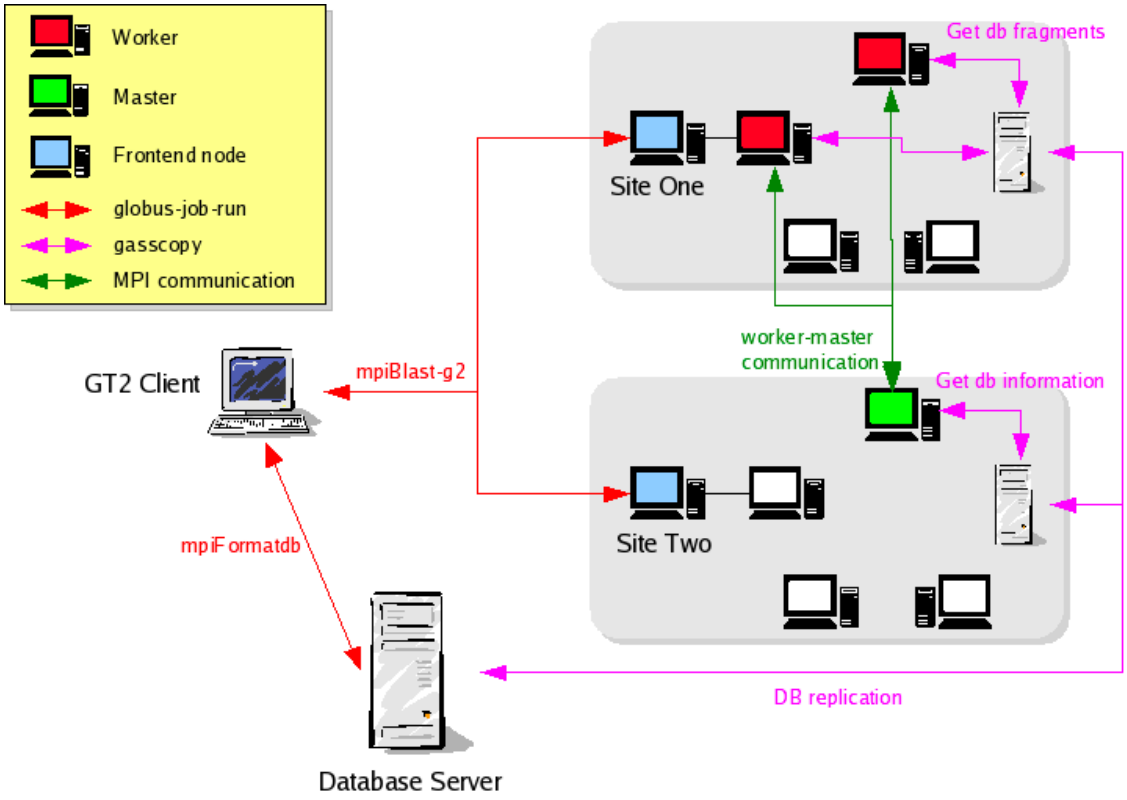

Imagen 26 - Arquitectura MPIBLAST

Como se puede observar en Imagen 26, estas aproximaciones están centradas en *cluster* heterogéneos, es decir, de fabricantes y/o arquitecturas. A pesar del hecho que, el concepto de aceleración es linealmente creciente con el número de procesadores dentro del *cluster* y que presenta un mejor comportamiento y una relación rendimiento-coste más efectiva que los multiprocesadores, los potenciales efectos de congestión de E/S entre los diferentes nodos que componen el *cluster* (comunicación entre los diferentes nodos MPI por paso de mensajes) y la escalabilidad horizontal de la tecnología en términos de número de procesadores lo convierte en una solución con una relación rendimiento-coste poco efectivo respecto a otras alternativas existentes que serán analizadas a continuación.

Field Programmable Gate Arrays (FPGA) es otra de las tecnologías que más variantes del algoritmo NCBI BLAST ha aportado a lo largo del tiempo, tanto a nivel de mecanismo de pre-filtrado [74], como de re-implementación de las fases

originales del algoritmo [8] o mediante la inclusión de heurísticas adicionales que permitan optimizar los procesos originales del algoritmo [10], [75].

En el primer caso y tal como se observa en la Imagen 27, el proceso de pre-filtrado aplicado se realizará a semejanza de un alineamiento convencional sin huecos y por tanto reduciendo la complejidad algorítmica original y mejorando su rendimiento. De esta manera, se alcanzará el objetivo deseado de reducir la base de datos para que el algoritmo original alinee únicamente aquellas secuencias que son relevantes. Sin embargo, este tipo de soluciones requieren de una elevada carga computacional que conlleva a su vez a una gama de FPGAs de alto coste y, por tanto, convirtiéndola en una solución poco accesible.

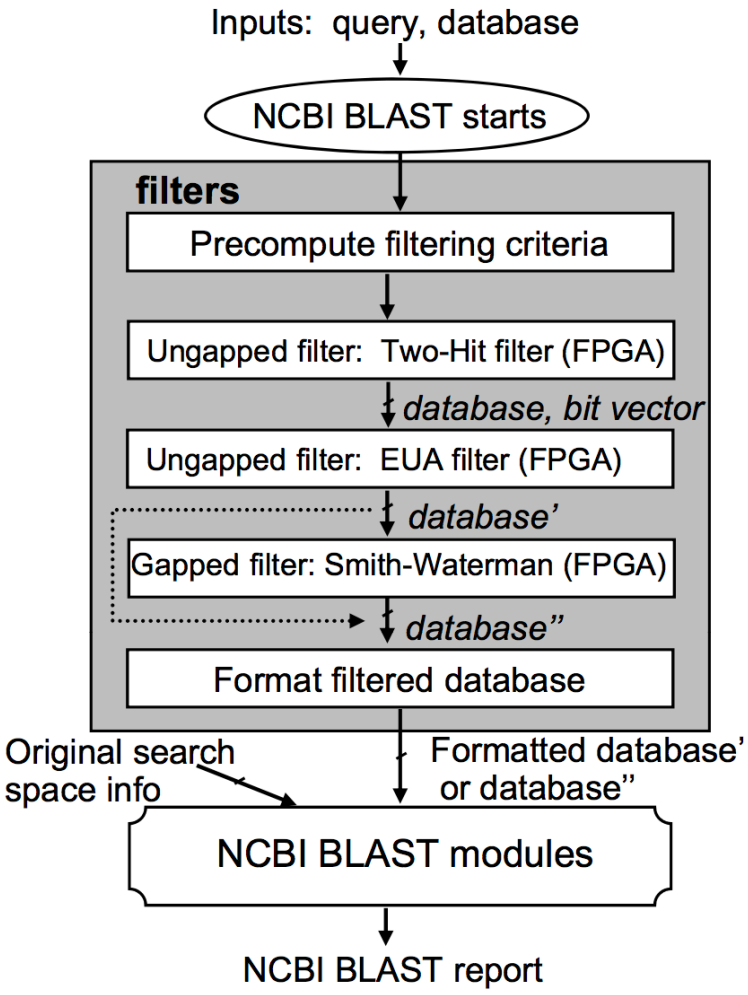

Imagen 27 - Arquitectura prefiltrado BLAST basado en FPGA

El segundo caso presentado tiene por objetivo llevar a cabo un perfilado del código fuente del algoritmo con el fin de optimizarlo y acelerarlo en aquellos módulos con mayor carga de procesamiento. Pese a que esta opción pueda proporcionar los mismos resultados, no resulta una solución con buena relación rendimiento-coste ya que su factor de aceleración estará vinculado en todo momento con la traducción del código fuente dentro del área del *chip* de procesamiento dentro la FPGA. En la Imagen 28 se puede observar la arquitectura modelo de este tipo de alternativas.

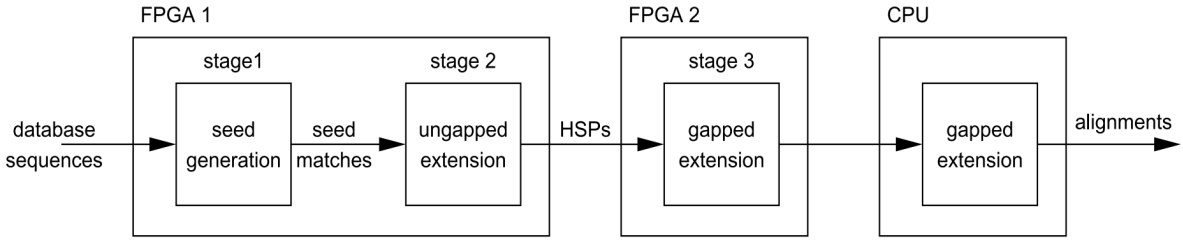

Imagen 28 - Arquitectura FPGA de re-implementación del algoritmo BLAST

El tercer y último caso está basado en la re-estructuración del código fuente, ya sea mediante la modificación de algunos criterios heurísticos del algoritmo original como de su propia eliminación. En este caso particular, se pueden observar rendimientos muy significativos, pero es extremadamente complicado llegar a una solución de compromiso entre precisión y rendimiento. Este caso de particularización y su correspondiente arquitectura se puede observar en la Imagen 29.

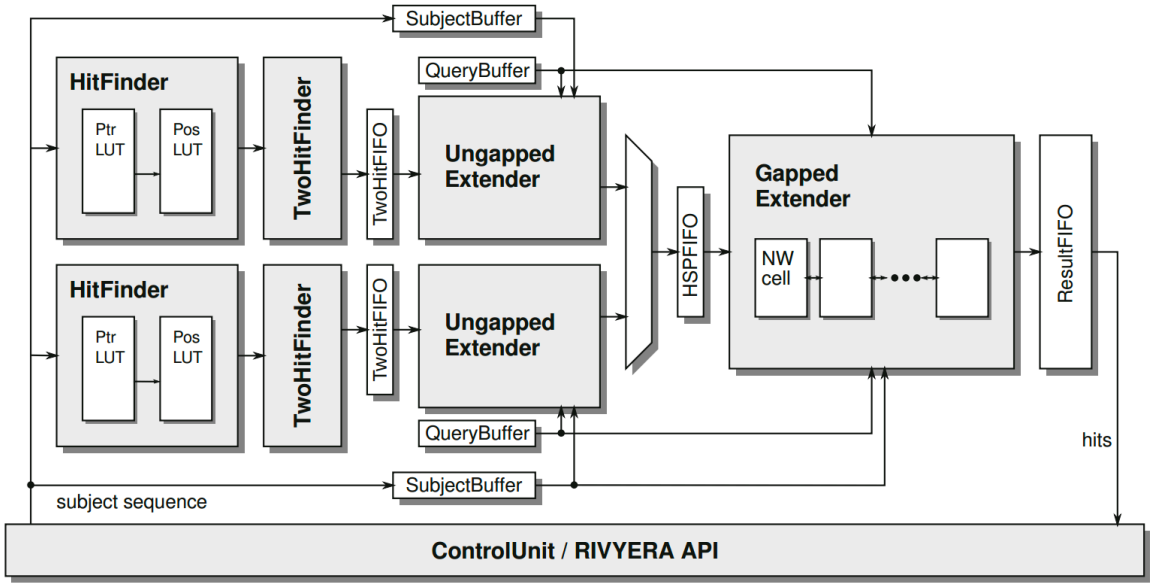

Imagen 29 - Arquitectura Rivyera sobre algoritmo BLAST

Las soluciones aportadas desde las FPGAs son muy variadas, con múltiples enfoques y con rendimientos bastantes significativos llegando a conseguir aceleraciones 376x. Sin embargo, los factores de aceleración de las soluciones propuestas poseen unas limitaciones físicas claras ya que dependerá directamente del área del propio *chip*. Esta limitación, junto con su precio actual, las convierten en una tecnología difícil de abordar.

Graphical Processor Units (GPU) representan, a diferencia del caso anterior con las FPGAs, una solución equilibrada entre coste y rendimiento. De esta manera, la presente tesis doctoral ha puesto su foco de atención en este tipo de tecnología y en la implementación de una variante del algoritmo BLAST que permita explotar todas las funciones de paralelismo mediante el uso de GPU. Llegados a este punto y a semejanza de las investigaciones llevadas a cabo en el ámbito de las FPGAs, existen varias líneas trabajo que, a pesar de emplear la misma tecnología de procesamiento (GPU), aportan enfoques completamente diferentes:

- Técnicas basadas en la re-implementación de las fases originales del algoritmo.
- Técnicas basadas en el pre-filtrado del algoritmo original.

Desde la perspectiva del primer tipo de técnicas, tanto los trabajos de Liu et al. con CUDA-BLASTP [15] o Vouzis et al. con GPU-BLAST [16] o Xiao et al. están basados en la separación de cada una de las fases de cálculo de algoritmo [15], su perfilado, re-implementación y aceleración de manera independiente.

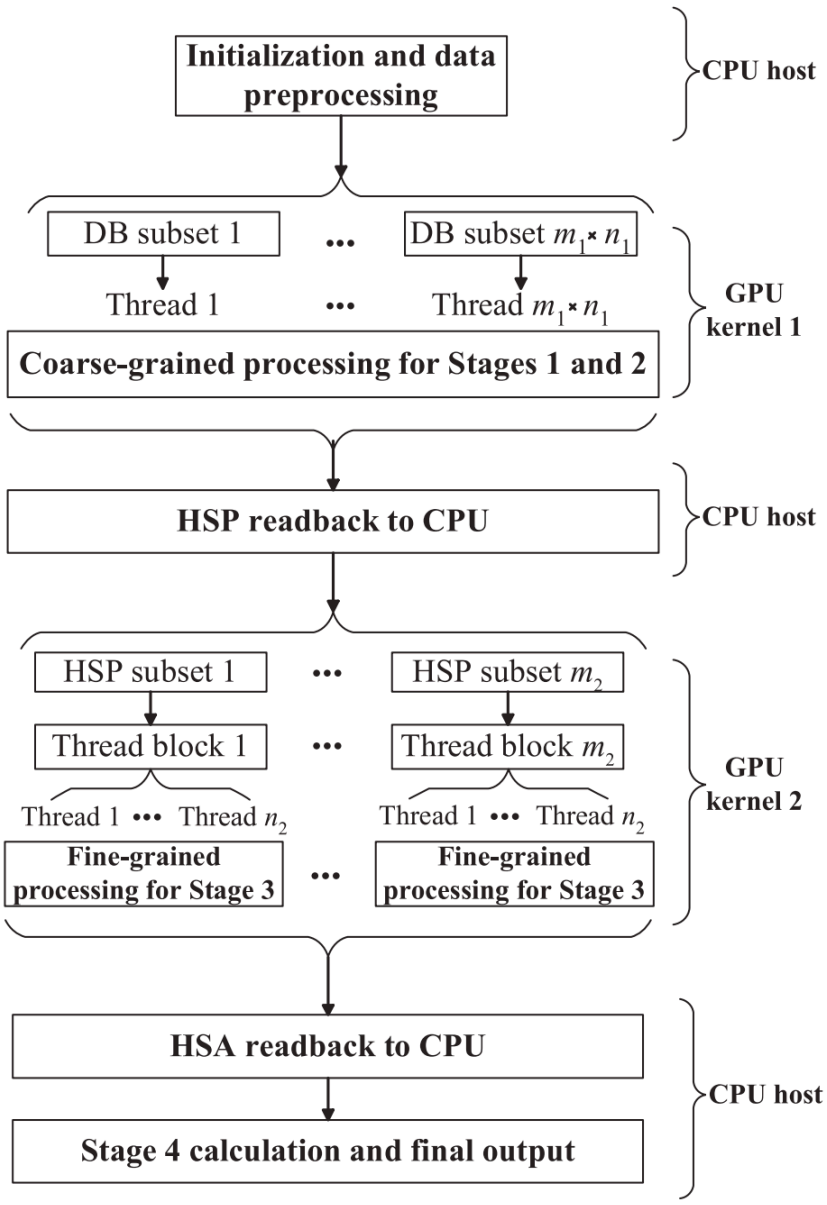

Imagen 30 - Arquitectura CUDA-BLASTP

Poniendo todos estos razonamiento e implementaciones sobre la práctica y tal como se puede apreciar en la Imagen 30, podemos observar que, a nivel de diseño, existe un elevado trasiego de información entre la GPU y la CPU por lo que, dependiendo del volumen de información que tratemos, podrá llegar a ser un factor limitante o cuello de botella en sí mismo. Desde el lado de implementación, la re-escritura de las diferentes fases que componen el algoritmo original y su complejidad a nivel heurístico pueden llegar a convertir la propuesta en un foco de errores potenciales difíciles de identificar. Finalmente, a nivel de evaluación podemos ver la parte más significativa donde vemos que Liu et al. con CUDA-BLASTP [15] únicamente evalúan el rendimiento de su modelo propuesta con 5 secuencias, Vouzis et al. con GPU-BLAST [16] evalúan su modelo con 51 secuencias y Xiao et al. emplean 1.000 secuencias sin ningún tipo de información sobre su filogenia. Por consiguiente, en todos estos casos el número de secuencias evaluadas está lejos de ser suficiente para tener una perspectiva real de la aceleración.

Por el contrario, nuestra investigación y la presente tesis ha seguido una línea de trabajo diferente respecto a los trabajos anteriormente descritos, y que se corresponde con el segundo tipo de técnicas basadas en el pre-filtrado de secuencias. Este tipo de técnicas no tienen por objetivo separar las fases de cálculo del algoritmo original e implementar una nueva versión de BLAST, sino que se basan en desarrollo de un sistema completamente independiente del algoritmo original de tal manera que permita cambios, tanto a nivel interno del propio algoritmo, es decir, modificaciones de su código fuente original como cambios del tipo algoritmo a lo largo del tiempo. Además, en nuestro caso particular la implementación ha sido evaluada por el Centro Nacional de Biotecnología (CNB) con 20.000 secuencias procedentes de diferentes familias de proteínas (*skewed* o no) en términos de precisión y rendimiento. Por consiguiente y a diferencia de los casos anteriores, este proceso ha sido evaluado mediante un organismo oficial especialista en el ámbito del algoritmo y frente a un mayor volumen de secuencias y de mayor variedad a nivel de filogenia por lo que aporta una visión más realista de la aceleración obtenida.

A modo de conclusión, la Tabla 1 muestra un resumen con las principales alternativas existentes junto con información relativa asociada, como por ejemplo el tipo de tecnología utilizado, el factor de aceleración en el caso mejor y caso peor respecto al algoritmo BLAST, y el conjunto de secuencias utilizadas para su evaluación.

Nombre	**Tecnología**	**Diseño**	**Mejor aceleración**	**Peor aceleración**	**# Secuencias evaluadas**
CUDA-BLASTp	GPU	Rediseño	3.2	1.5	5 secuencias
GPU-BLAST	GPU	Rediseño	6.1	4.8	51 secuencias
Xiao et al. solution	GPU	Rediseño	5.7	4.3	1.000 secuencias
Mercury BLASTp	FPGA	Rediseño	15	11	4.241 secuencias
RIVYERA BLAST	FPGA	Rediseño	376	2	7.476 secuencias
CAAD BLASTp	FPGA	Pre-filtrado	11	6	No especificado
MPIBLAST	CPU	Rediseño	Lineal con el número de procesadores		
BLAST++	CPU	Rediseño	800	600	300 secuencias
Nuestra solución	GPU	Pre-filtrado	4.1	2.3	20.000 secuencias

Tabla 1 - Comparativa entre soluciones de altas prestaciones

Tal y como se ha venido describiendo en las secciones previas, existen múltiples alternativas dentro de los sistemas de altas prestaciones que permiten acelerar las aplicaciones de secuenciación basadas en detección de homologías

existentes en la actualidad. Sin embargo, todas ellas constan de una serie de ventajas e inconvenientes que deben ser tenidas en cuenta a la hora de elegir aquella que se adapte mejor a cada caso particular.

Tras analizar los sistemas *multicore*, se ha observado que son sistemas de altas prestaciones capaces de ofrecer un gran rendimiento a un coste muy competitivo. Sin embargo, y en nuestro caso particular, requeriría de un uso prácticamente exclusivo para tareas de secuenciación, al tratarse de procesos muy exigentes en términos de computación y no disponer de una unidad de procesamiento independiente al procesador donde se ejecuta el sistema operativo. Esta consideración anula por completo una de las principales ventajas de nuestra propuesta, basada en la ejecución de aplicaciones de secuenciación sobre plataformas *commodity hardware*. Además, este tipo de sistemas presenta una escalabilidad horizontal con mayores costes que otros sistemas, como por ejemplo las GPUs, por lo que fue finalmente descartado como sistema de altas prestaciones de referencia en la presente tesis doctoral.

En lo que respecta a las FPGAs, se ha observado que son sistemas de altas prestaciones capaces de ofrecer un excelente rendimiento, aunque por el momento, con un coste elevado. Al igual que en el caso anterior, esta barrera de entrada incumple la premisa inicial de la solución propuesta, basada en la ejecución de aplicaciones de secuenciación sobre plataformas *commodity hardware*. Además, las limitaciones físicas impuestas por la FPGA a la hora de pre-cargar la aplicación condiciona y limita el número de aplicaciones compatibles, ya que dependerá de su tamaño y complejidad. Por este motivo, las FPGAs fueron descartadas como sistema de altas prestaciones de referencia en la presente tesis doctoral, y a favor de las GPUs, por ofrecer una relación equilibrada entre rendimiento y coste y sin presentar limitaciones físicas significativas en entornos *commodity hardware*.

Una vez seleccionadas las GPUs como sistema de altas prestaciones, se procedió a la toma de decisión relativa al lenguaje de programación a emplear,

OpenCL o CUDA. Ambos lenguajes son dependientes del fabricante de la GPU y, de la misma manera que con los sistemas de altas prestaciones, poseen sus ventajas e inconvenientes. Tras analizar múltiples trabajos de investigación [76][77], se ha optado por CUDA como lenguaje de programación de referencia y NVidia como fabricante seleccionado por ofrecer un mejor rendimiento en las transferencias de datos entre GPU – CPU y ejecuciones del *kernel*.

Finalmente, y en lo que respecta a la solución propuesta en la presente tesis doctoral, las primeras aproximaciones llevadas a cabo se basaron en la re-implementación del código fuente de la aplicación BLAST. Este proceso de diseño e implementación siguió la misma línea que los trabajos de investigación previos sobre la aceleración de la aplicación BLAST en GPUs [15][16][17]. Este primer enfoque implicó la asignación de una gran cantidad de recursos y tiempo de desarrollo con un resultado muy limitado ya que, debido a la complejidad de la aplicación BLAST, únicamente su variante para proteínas BLASTP fue implementada parcialmente y sin poder llegar a evaluarse respecto a la aplicación de referencia. A consecuencia de ello, se decidió optar por una solución de pre-filtrado independiente a la aplicación de secuenciación y que no necesitase conocer en detalle su código fuente para acelerarlo. De esta manera, surgió el sistema de pre-filtrado propuesto que se detalla en el siguiente capítulo.

Capítulo 5 Metodología

Mediante el uso de técnicas de computación de altas prestaciones (HPC) descritas en el capítulo anterior, se ha procedido al diseño e implementación de la solución de pre-filtrado propuesta. Dicha propuesta parte de la consideración inicial que el número de *hits* entre dos secuencias es directamente proporcional a la relevancia del alineamiento. Con el objetivo de evitar que secuencias de mayor longitud sean erróneamente más relevantes por defecto, este valor de *hits* se encuentra normalizado respecto al espacio de búsqueda del alineamiento, es decir, respecto a la longitud de ambas secuencias que lo conforman. Al contrario que otras soluciones alternativas [24], el sistema propuesto es completamente indiferente al algoritmo de detección de homologías utilizado y está basado en un modelo teórico optimizado para GPU, lo cual proporciona un mejor comportamiento tanto en proteínas, nucleótidos y metagenomas.

Por consiguiente, este capítulo constará de los dos elementos principales para la formalización del modelo teórico propuesto, comenzando con la definición de *hit* y seguido a continuación de múltiples escenarios donde este concepto toma especial relevancia. En segundo lugar, se extenderá dicha definición a través del concepto de la proximidad entre *hits* y cómo la aparición de un mayor número de *hits* en una región próxima puede conllevar a una mayor relevancia del alineamiento entre secuencias.

5.1. Definición del concepto de Hit

Tal como se ha visto en el capítulo anterior, la mayor parte de los algoritmos basados en detección de homologías, ya estén destinados al alineamiento de proteínas, nucleótidos o metagenomas, tienen como piedra angular al alineamiento de secuencias, la búsqueda de coincidencias o *hits.* Uno de los ejemplos más claros en este sentido es la aplicación de referencia BLAST y su

fase de *seeding* [5]. Sin embargo, dependiendo de cada caso particular, la definición de *hit* puede ser ligeramente diferente.

Desde la perspectiva de los alineamientos entre proteínas y metagenomas, el concepto de *hit* se define como la ocurrencia entre dos aminoácidos alineados previamente y con una puntuación positiva dada una matriz de substitución concreta [32], [42]. En el caso de los metagenomas, esta consideración tiene una puntualización adicional ya que se requiere el proceso de traducción correspondiente de nucleótidos a las diferentes versiones o *strands* de proteínas de la secuencia. Asimismo y como analizamos en capítulos anteriores, la obligatoriedad que la puntuación del alineamiento sea positiva se debe al modelo estadístico de Karlin-Altschul y la definición de una matriz de substitución procede de los estudios empíricos llevados por Margaret Dayhoff con las matrices BLOSUM62 [40] o PAM250 [42]. Estas matrices son definidas como tablas bidimensionales que describen la proporción por el cual un aminoácido dentro de una secuencia cambia a otro aminoácido a lo largo del tiempo y por consiguiente, proporciona un factor de similitud. De esta manera, la definición de una matriz de substitución y sus términos propios quedan definidos como elementos basados en términos empíricos y fundamentados en la propia experimentación con diferentes secuencias de aminoácidos [78]. En la actualidad, esta situación representa un problema en el ámbito de la bioinformática ya que muchas de estas matrices y sus valores están quedando obsoletas respecto al ritmo con el que evolucionan las secuencias de aminoácidos. Sin embargo, con el objetivo de solventar dicha problemática, existen otras líneas de investigación alternativas basadas en las propiedades físico-químicas de los propios aminoácidos que permiten tener una representación de los datos que varíe con el tiempo y se adapte a la propia evolución de los aminoácidos [79], [80].

$$S(i,j) = \log\frac{p_i M_{i,j}}{p_i p_j} = \log\frac{M_{i,j}}{p_j} = \log\frac{observed\ frequency}{expected\ frequency}$$

Ecuación 15 - Cálculo de hit

La Ecuación 15 se corresponde con la expresión de puntuación de un *hit* entre dos aminoácidos, donde el parámetro $M_{i,j}$ se define como la probabilidad que un aminoácido *i* se transforme en un aminoácido *j* y los parámetros $p_i p_j$ se corresponden con sus frecuencias individuales respectivamente. Por consiguiente, a partir de los cálculos previos se puede determinar que la probabilidad o correlación entre dos aminoácidos *S(i,j)*, sigue una distribución logarítmica donde una mayor verosimilitud entre los aminoácidos alineados implica una mayor la relevancia alcanzada y viceversa.

Por último, el proceso de secuenciación entre nucleótidos define un *hit* de dos maneras diferentes. En primer lugar, existen aplicaciones de detección de homologías de referencia, como por ejemplo BLAST, que definen un *hit* como la coincidencia exacta entre dos aminoácidos [32], [42]. En este caso particular, la no existencia de una matriz de substitución como en el caso anterior, simplifica el proceso de búsqueda de *hits* e incrementa la fiabilidad de estimación en términos de verosimilitud entre secuencias. Sin embargo, existen líneas de investigación alternativas que tienen en cuenta las substituciones y transiciones entre nucleótidos para crear matrices de substitución personalizadas a cada caso y adaptar mejor el modelo a posibles mutaciones o transformaciones de nucleótidos dado su carácter evolutivo [81].

5.2. Principio de densidad de hits

Tal y como se ha descrito en secciones anteriores, el proceso de búsqueda de *hits* es una fase preliminar para predecir la relevancia de un alineamiento entre dos secuencias. La naturaleza de este proceso es altamente paralelizable y con una complejidad algorítmica de $O(N^4)$ donde *N* se corresponde con el número de *lmers* encontrados por secuencia en cada base de datos. El siguiente pseudocódigo es una representación algorítmica de dicho proceso:

```
# Iterate over a list of query sequence
For query_sequence in query_sequence_list:
     For query_sequence_lmer in query_sequence:
           # Iterate over a list of database sequence
           for db_sequence in db_sequence_list:
                 for db_sequence_lmer in db_sequence:
                 # Check matches
                       if db_lmer == query_lmer:
                             match++
```

A modo de explicación, el siguiente pseudocódigo iterará por cada una de las secuencias de entrada y por cada uno de los *lmers* que componen dicha secuencia. Cabe destacar que, al igual que en los algoritmos tradicionales de detección de homologías, el tamaño del *lmer* determinará la sensibilidad del algoritmo y su rendimiento. A continuación, por cada *lmer* de la secuencia de entrada, se iterará sobre las secuencias procedentes de la base de datos y sus correspondientes *lmers* con el objetivo de encontrar una coincidencia. En tal caso, se incrementará un valor de puntuación que determine la relevancia sin normalizar de la secuencia dentro de la base de datos.

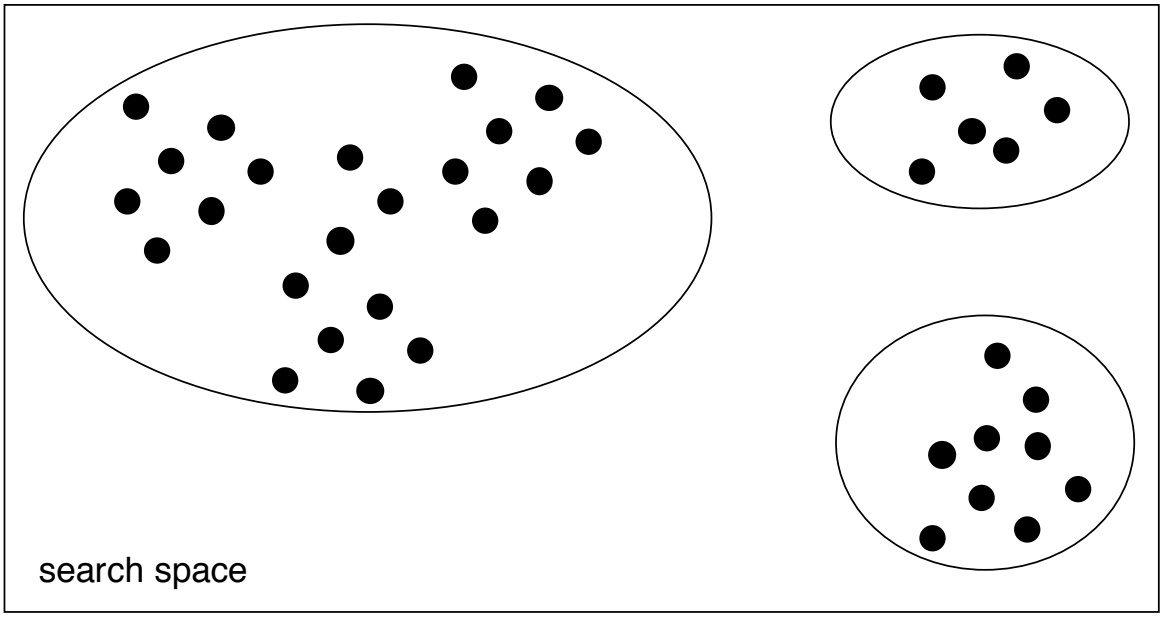

Imagen 31 - Relación entre proximidad de hits y espacio de búsqueda

Sin embargo y como se puede observar en la Imagen 31, para comprender correctamente el proceso de normalización sobre el valor de relevancia de un alineamiento, es necesario detallar el concepto de proximidad entre *hits*. Para ello, se procederá a analizar la frecuencia de un determinado *hit* sobre un *lmer y* respecto a un gran número de secuencias y de tamaño variable. Intuitivamente, aquellas secuencias con mayor longitud tendrán una mayor probabilidad de encontrar un valor más elevado de *hits,* que aquellas secuencias con una longitud inferior. Sin embargo y de acuerdo con la ecuación de Karlin-Altschul (Ecuación 4), se puede determinar que la longitud de las secuencias no tiene por qué ser determinante a la hora de calcular la relevancia de un alineamiento entre dos secuencias. Como se muestra en la siguiente ecuación, para la determinar la relevancia de un alineamiento entre dos secuencias, se debe establecer una relación en términos del espacio de búsqueda (*m'n'*), un valor de puntuación (*S*) y factores de escalado determinados por métodos experimentales (*K* y λ).

De manera detallada, se puede determinar que tanto los factores de escalado, como el valor de puntuación inicial de la secuencia son dependientes del esquema de puntuación utilizado y, por tanto, presentan una reducida varianza entre alineamientos. Por consiguiente, el espacio de búsqueda entre ambas secuencias es el punto clave para determinar la relevancia entre un alineamiento. De esta manera y una vez revisados todos los elementos de la ecuación, tenemos los datos suficientes para determinar que la proximidad entre *hits* reduce el espacio de búsqueda efectivo entre dos secuencias y por tanto incrementa la relevancia de su alineamiento.

Sin embargo, existe una fuerte contradicción entre el proceso de búsqueda de *hits* y la filogenética que debe ser tenida en cuenta. De acuerdo con algunos estudios especializados [82], todos aquellas aplicaciones de detección de homologías que confían únicamente en procesos estadísticos, como por ejemplo BLAST [5], deben ser interpretados con especial precaución ya que la proximidad entre *hits* no siempre representa una proximidad entre familias de secuencias o proximidad filogenética. Esta distancia entre familias de secuencias

debe ser entendida como aquella obtenida a partir de una matriz de valores morfológicos entre dos especies y acorde a su propio comportamiento evolutivo. En particular, el modelo propuesto de proximidad entre *hits* puede también contribuir a los recientes algoritmos de detección de homologías basados en el modelado de secuencias con nuevos niveles de verosimilitud filogenética dado a que es capaz de considerar grupos de *hits* en lugar de casos individuales.

Por último, otro punto de especial relevancia es que todo este proceso de pre-filtrado y su correspondiente salida puede ser redirigido a cualquier aplicación de detección de homologías, como por ejemplo BLAST [5], BWA [83], BLAT [84]. De esta manera, estas complejas aplicaciones, que dedican grandes cantidades de tiempo en evaluar la relevancia de cada secuencia, podrán mejorar su rendimiento al reducir la base de datos de secuencias a aquellas que sean relevantes.

Capítulo 6 Implementación

El modelo teórico descrito en el capítulo anterior, ha definido el concepto de *hit* sobre un alineamiento de secuencias y la importancia de su proximidad respecto a otros para determinar la relevancia del mismo. Fundamentada a través del modelo teórico de Karlin-Altschul, esta propuesta proporciona información suficiente para poder establecer una relación entre *hits* y la relevancia de un alineamiento. Además, como se puede observar en la Imagen 32, otra de las particularidades del modelo propuesto, es que ha sido diseñado, implementado y optimizado para una arquitectura específica, basada en multiprocesadores y GPUs [85].

El multiprocesador será el elemento encargado de formatear la base de datos secuencias en un formato propio de rápido acceso y proporcionársela, junto con las secuencias de entrada y el programa CUDA (*kernel*), a la GPU para su ejecución en paralelo. Como ya comentaremos en secciones posteriores, este intenso trasiego de datos llevará consigo un mecanismo propio de aprovisionamiento de datos que reduzca el impacto sobre la E/S. En segundo lugar, la GPU será la encargada, a través de la aplicación CUDA (*kernel*) proporcionada, de dividir los bloques de información acorde a su propia arquitectura para su posterior ejecución en paralelo. Para llevar a cabo todas estas operaciones en paralelo, la GPU podrá utilizar tanto su memoria RAM interna y de rápido acceso como el disco duro del sistema para aquellos datos que no puedan ser almacenados en su memoria interna. Una vez finalizados todos los trabajos en paralelo, se proporcionarán los resultados al multiprocesador para su presentación al usuario por pantalla.

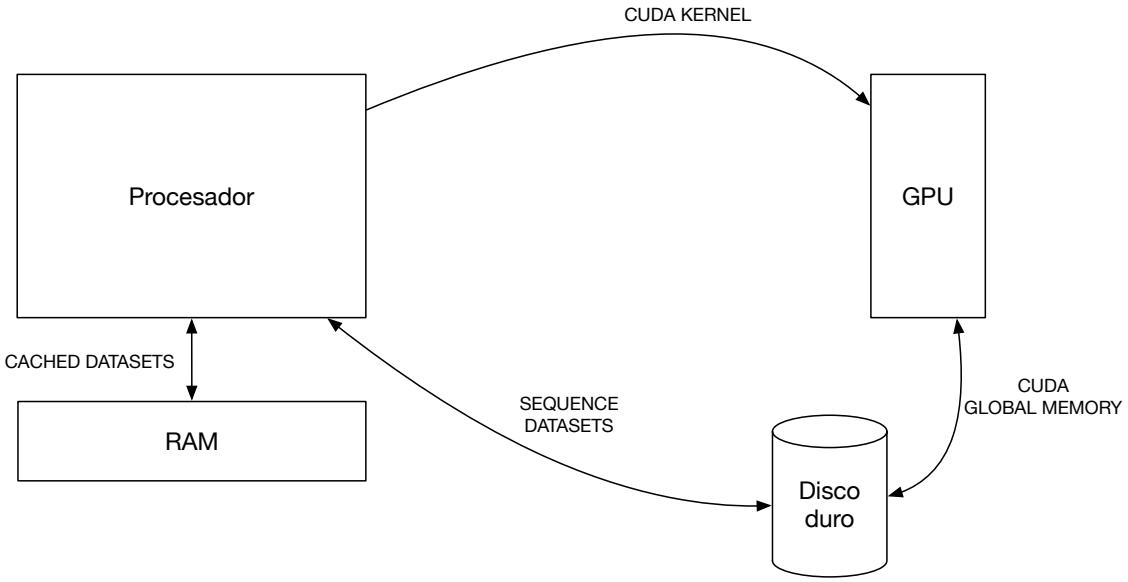

Imagen 32 - Arquitectura General

En un segundo nivel lógico, y como veremos en apartados posteriores, el diseño propuesta estará compuesto de los siguientes elementos principales:

- Una tabla de búsqueda o base de datos de secuencias (*lookup table*).
- Un modelo de filtrado basado en GPU.

6.1. Base de datos

El primer paso dentro de la fase de diseño e implementación fue proporcionar un formato adecuado a la base de datos de secuencias. Mediante el uso de tablas de búsqueda o tablas *hash* personalizadas y listas enlazadas, el procedimiento actual tiene por objetivo reducir el coste computacional del proceso de búsqueda de *hits* a una complejidad algorítmica de *O(1*), es decir, acceso directo. A diferencia de estudios previos que utilizan estructuras de datos basadas en listas estándares y con complejidad algorítmica lineal O(N), como por ejemplo la versión implementada por el NCBI *formatdb*, nuestra propuesta está basada en funciones *hash* y procesamiento optimizado a multiprocesadores lo que permite alcanzar mejoras significativas de rendimiento. Además, debido al hecho que la implementación estándar de tablas *hash* en los sistemas operativos Unix con la

librería GLIBC es muy compleja y difícil de integrar, se ha propuesto el siguiente modelo de implementación.

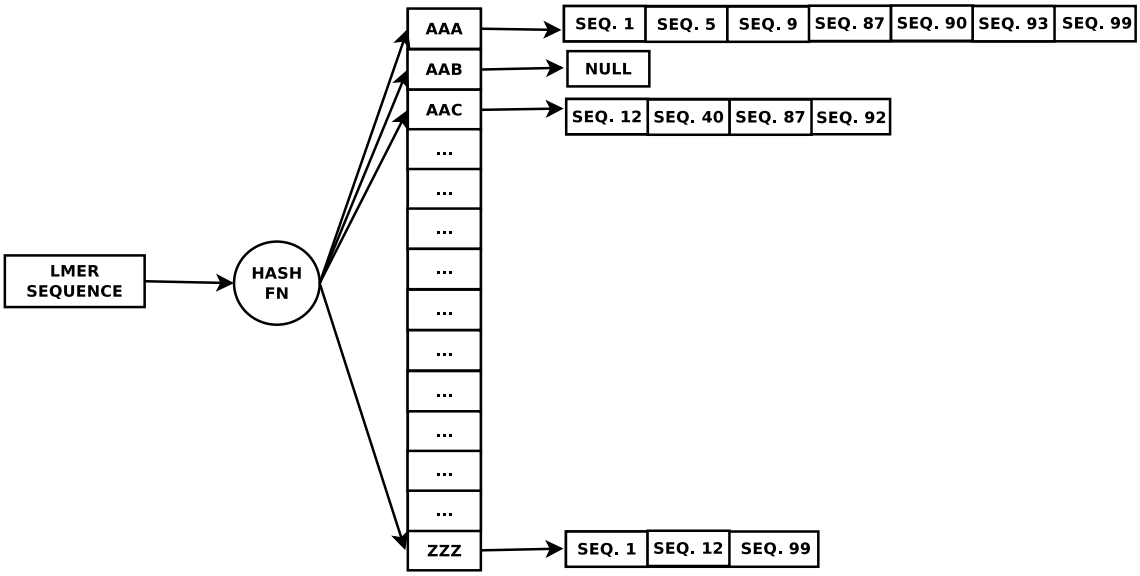

Imagen 33 - Arquitectura de Indexación de Secuencias

Como se puede observar en la Imagen 33, el sistema de procesamiento aprovecha todos los procesadores existentes y disponibles en la máquina, pero sin funciones adicionales *hyperthreading*, mediante la creación de un conjunto de hilos POSIX o *pthreads*. El motivo de esta decisión de arquitectura sobre la solución propuesta se ha basado en el impacto de los cambios de contexto entre los hilos existentes en un sistema operativo y sus efectos en términos de rendimiento. Por consiguiente, a mayor número de hilos existentes dentro del sistema operativo ejecutando diferentes tareas, mayor trabajo para el sistema operativo en procesos de cambios de contexto y, por tanto, mayor tiempo de ejecución perdido en estas tareas. Cada uno de estos hilos recorrerá cada secuencia de la base de datos, dividiéndola en la unidad mínima de procesamiento que llamaremos *lmer* y llevando a cabo la correspondiente función *hash*. La estructura correspondiente a la unidad de procesamiento estará directamente relacionada con el modelo de secuenciación, por lo que, en el caso de proteínas y metagenomas, es habitual utilizar *lmers* de tres aminoácidos de longitud mientras que en el caso de nucleótidos es habitual utilizar *lmers* de once aminoácidos de longitud. Por consiguiente, en términos de número de entradas

y volumen de los datos, el modelo de base de datos propuesto para proteínas y metagenomas y con un alfabeto de 24 aminoácidos tendrá 24^3 entradas, es decir, 55.296 bytes en memoria de CPU. En el caso de nucleótidos y su alfabeto de 4 aminoácidos, este valor será aún mayor con un tamaño de 4^{11} entradas y 4.194.304 bytes en memoria de CPU.

De esta manera, el rendimiento de este procedimiento dependerá directamente de ambos factores: longitud de la unidad de procesamiento y modelo de secuenciación. A continuación, se muestra la ecuación correspondiente de la función *hash* para proteínas que retornará el índice sobre la cual partirá la lista enlazada correspondiente con los identificadores únicos de cada secuencia.

$$protein\_hash = \sum hash\_value(lmer[i]) * 24^{sequence\_length-(i+1)}$$

Ecuación 16 - Cálculo hash para formateo de base de datos de proteínas

Como se observa en Ecuación 16, tras la obtención del índice *hash*, cada hilo comprobará la existencia de la lista enlazada con los identificadores únicos de cada secuencia. En caso afirmativo, se reasignará un nuevo bloque de memoria e insertará el número de secuencia dentro de la lista enlazada. El motivo por el cual se ha elegido usar este tipo de estructuras de datos para cada tabla *hash* ha estado basada en principios de claridad, simplicidad y rápido prototipado. Todas las reservas de memoria son consideradas operaciones atómicas con el fin de evitar problemas de concurrencia.

Finalmente, y con el fin de garantizar una compatibilidad con los diferentes modelos de secuenciación y optimizar el espacio de memoria, se ha determinado realizar un modelo de implementación con resolución a nivel de *bit*. Una de las principales ventajas de este modelo propio es la prevención de colisiones entre entradas de la tabla *hash*. Esta consideración de diseño mejora el rendimiento respecto a otras soluciones, como por ejemplo el modelo de información ordenada a nivel de secuencia, el cual ha sido probado y evaluado con el objetivo

de tener un criterio más amplio y contrastado en términos de coste computacional y rendimiento.

6.2. Modelo de filtrado

Como se muestra en la Imagen 34, el modelo de filtrado propuesto está basado en cuatro fases de procesamiento:

- Modelo de pre-procesado de secuencias.
- Modelo de particionamiento de datos.
- Modelo de *hitting*.
- Modelo de reducción en dos fases.

El modelo de pre-procesamiento dentro de la base de datos está basado en el procedimiento que se describe en la sección anterior, en el cual se comienza con un pre-procesamiento de la base de datos y conjunto de secuencias de entrada que serán enviados, dado su elevado volumen, a la memoria global de la GPU. Este tipo de memoria, aunque será analizada en mayor detalle en la sección de consideraciones adicionales, se trata de una zona de disco duro mapeada desde la GPU para acceso a grandes cantidades de datos que no pueden ser alojadas en las memorias internas de la GPU, como por ejemplo la memoria compartida o de texturas. Como resultado de la política de procesamiento, una base de datos pre-procesada será retornada en forma de tabla *hash* con los índices de las diferentes secuencias que componen la base de datos en una estructura de datos de lista enlazada.

Una vez procesada la base de datos, será el turno del conjunto de secuencias de entrada donde el modelo de pre-procesado procederá a crear un flujo de datos, mediante la concatenación de secuencias y con tamaño variable. Este modelo de representación, especialmente en el caso de las secuencias entrada, ha sido diseñado con el fin de minimizar las transferencias de memoria entre CPU y GPU, y optimizar el rendimiento de la solución propuesta.

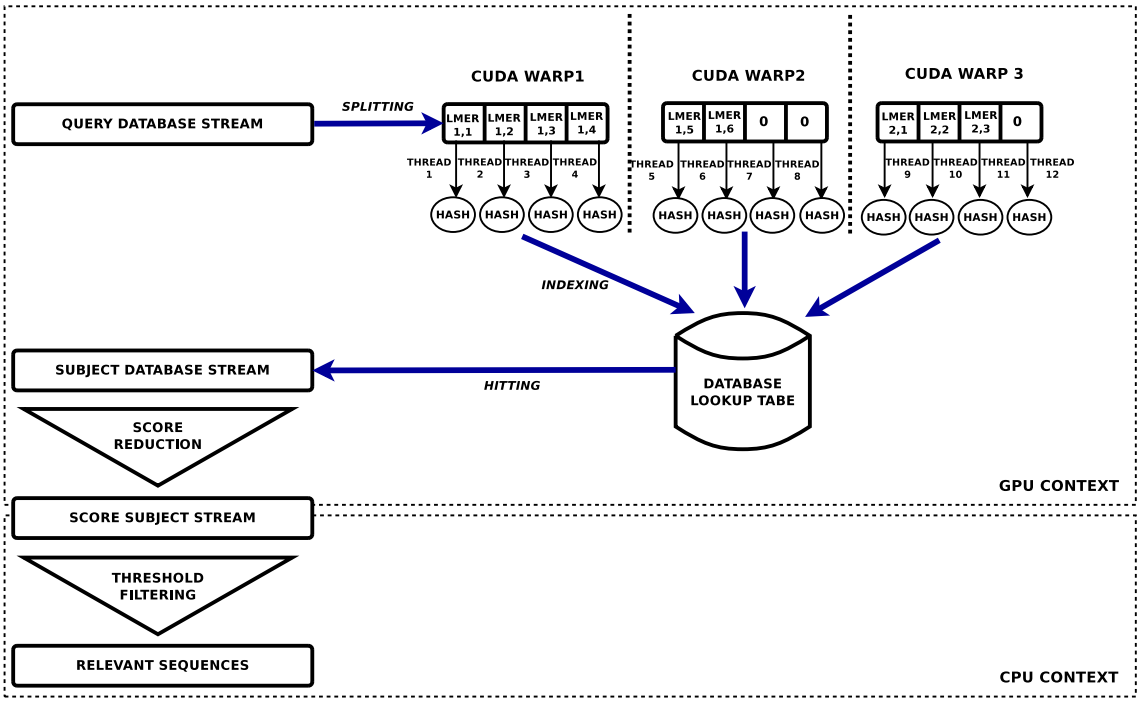

Imagen 34 - Arquitectura Detallada

En segundo lugar, se procederá al modelo de particionado de los datos donde cada secuencia de entrada, como parte del flujo de datos proporcionado a la GPU, será dividido por cada uno de los hilos de ejecución disponibles en el sistema en *lmers* y agrupado por bloques de *threads*, también denominados CUDA *warps*. Cada CUDA *warp* estará asociado a un único procesador CUDA y será el encargado de dividir el bloque correspondiente de datos en *lmers* e incorporar el relleno o *zero-padding* necesario para evitar efectos de divergencia y tratamiento incorrecto de *lmers*.

En tercer lugar, y como se puede observar a través de la Imagen 34, se procederá al modelo de *hitting* donde cada hilo de ejecución será responsable de procesar su *lmer* asignado. Este proceso comenzará con la indexación del correspondiente *lmer* procedente del flujo de secuencias de entrada. Este proceso de indexación empleará la misma función *hash* usada durante la fase de pre-procesado de la base de datos. Posteriormente, cada hilo de ejecución escribirá el valor **“1”** dentro del flujo de datos en caso de existencia de dicho *lmer* dentro de la base de datos o un **“0”** en caso contrario. Desde el punto de

vista de la estructura de datos utilizada y como se puede observar en la Imagen 35, este flujo de información procedente de la base de datos es representado mediante una matriz bidimensional donde las filas representan todas las secuencias de la base de datos y cada columna su correspondiente *Imer*.

Imagen 35 - Matriz de hitting

Una vez finalizada la ejecución de todos los hilos existentes y sincronizada en una última etapa mediante el uso de CUDA *barriers*, se procederá al modelo de reducción en dos fases. El objetivo de esta primera fase de procesamiento consiste en retornar, a partir de la matriz bidimensional obtenida de la fase anterior, un único valor al usuario que identifique la relevancia de la cadena correspondiente dentro de la base de datos de referencia. Como se puede observar en la Imagen 36, el primer paso de esta etapa está basado en la ejecución en paralelo de todos los hilos de ejecución existentes en bloques acorde con el tamaño del CUDA *warp* y su objetivo estará focalizado la obtención del sumatorio de todos los elementos contenidos en cada CUDA *warp.* Llegados a este punto, una consideración adicional a tener en cuenta es la política de asignación de CUDA *warps* en función del flujo de datos entrante y particionado mediante su modelo de procesamiento correspondiente. Un correcto esquema

de particionado es clave de cara a garantizar una paralelización completa entre los diferentes hilos de ejecución y, por tanto, reducir el efecto de divergencia entre ellos.

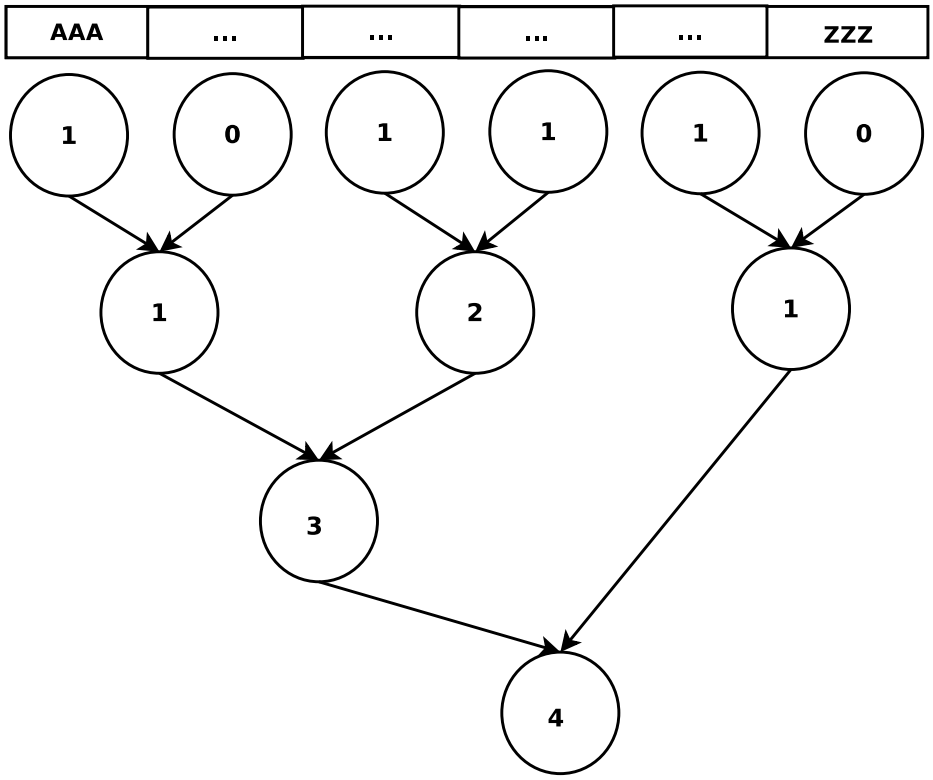

Imagen 36 - Arquitectura de Reducción de Secuencias

Finalmente, una vez que todos los CUDA *warps* han finalizado sus ejecuciones correspondientes, la segunda etapa del proceso de reducción entrará en escena con el objetivo de obtener el mínimo valor de puntuación o porcentaje de verosimilitud. A continuación, se retornará la secuencia evaluada en formato FASTA en caso que ésta supere un valor introducido por el usuario como parámetro de entrada.

6.3. Consideraciones adicionales

Las arquitecturas de altas prestaciones basadas en GPU son arquitecturas de carácter complejo y que poseen características particulares que pueden afectar de manera significativamente en el rendimiento.

La primera consideración es la política de procesamiento de datos. La penalización, en términos de tiempo, producida por las transferencias de datos entre GPU y CPU es uno de los principales cuellos de botella en aplicaciones basadas en el paralelismo, y más concretamente, en aquellas cuyos algoritmos manejan grandes cantidades de datos.

Con el fin de reducir este efecto, la solución propuesta ha diseñado un modelo de tratamiento de información donde la memoria global de la GPU es dividida en dos secciones principales de tal manera que, mientras la primera sección es procesada, la segunda sección va cargando datos de manera asíncrona mediante el uso de CUDA *streams*. Este tipo de transferencias de datos está disponible desde la arquitectura NVidia Fermi. Por consiguiente, y de acuerdo con la política de procesamiento de datos propuesta, el sistema propuesto es capaz de reducir las penalizaciones en transferencias de datos a la primera transferencia exclusivamente.

El segundo aspecto a tener en cuenta está centrando en la política de *hitting*. Este procedimiento tiene por objetivo hacer frente a dos de los principales problemas existentes en las aplicaciones desarrolladas en ámbitos paralelizables, coalescencia y divergencia. El efecto de la coalescencia se define como la combinación de múltiples accesos a memoria dentro de la misma operación. Este concepto está directamente relacionado con los principios de localidad de los datos dentro de la GPU y tiene un impacto directo en el rendimiento de las aplicaciones, especialmente en sus operaciones de entrada/salida.

Como se puede observar en la Imagen 38, la situación ideal en este tipo de situaciones se correspondería con accesos contiguos a memoria donde los hilos no tendrían que hacer saltos aleatorios en memoria para acceder a los datos.

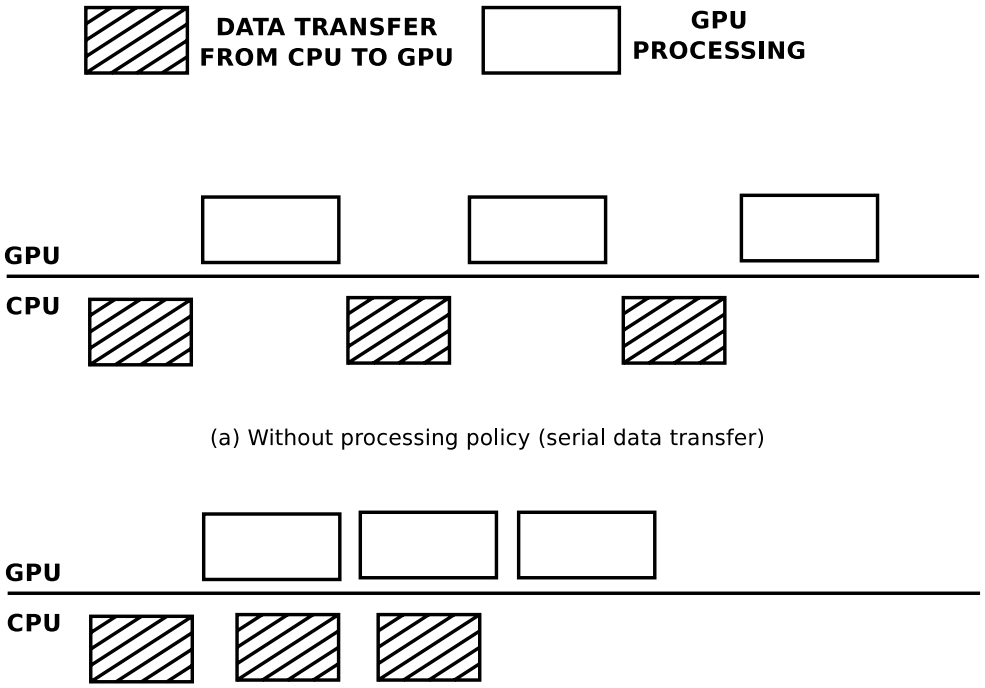

Imagen 37 - Comparación sobre Políticas de Procesamiento

Sin embargo y como se muestra en la Imagen 39, existen una serie de situaciones que pueden llevar a impedir la coalescencia sobre una aplicación:

- Memoria no secuencial
- Accesos a memoria dispersos
- Accesos a memoria desalineados

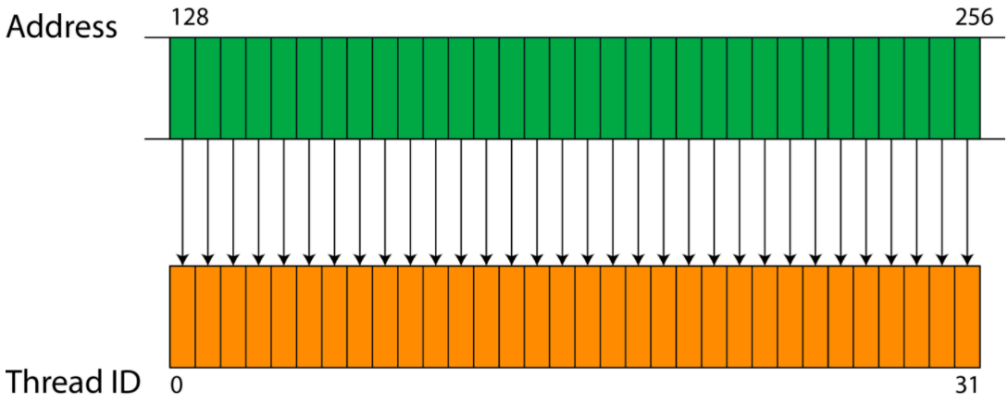

Imagen 38 - Acceso a memoria coalescente

Imagen 39 - Acceso a memoria no coalescente

Para resolver la problemática de la coalescencia, se necesita llevar a cabo accesos a bloques de memoria contiguos por los hilos de ejecución de la GPU y minimizar la penalización causada por los fallos de caché L1 y L2. A continuación se muestran dos bloques de código CUDA con ejemplos de accesos coalescentes y no coalescentes respectivamente.

```
shmem[threadIdx.x]=gmem[blockIdx.x*blockDim.x +
threadIdx.x];
```

```
stride=4;
shmem[threadIdx.x]=gmem[stride*blockIdx.x*blockDim.x +
threadIdx.x*stride];
```

Con el objetivo de comprender el efecto de la divergencia, se debe partir de los conceptos descritos en la arquitectura detallada de la solución propuesta, donde los hilos existentes están agrupados en bloques de tamaño fijo denominados *warps* para su ejecución en un procesador CUDA. A continuación, todos los hilos deberán seguir la misma línea de ejecución, es decir, ejecutando en paralelo la misma instrucción en el mismo instante de tiempo y sin producirse divergencia alguna. Sin embargo, la sentencia condicional más utilizada en programación *if-then-else*, es el principal factor que contribuye a la generación de divergencia dentro de una GPU, ya que producirá una ramificación dentro de la propia línea de ejecución que repercutirá de manera directa sobre los hilos existentes.

En lo que respecta al esquema de programación en CUDA, este proceso de ramificación será serializado de tal manera que, en primer lugar, se ejecutarán

aquellos hilos cuya expresión condicional sea afirmativa y, en segundo lugar, aquellos cuya expresión condicional sea negativa. Como se puede observar en la Imagen 40, este efecto tendrá consecuencias linealmente relevantes respecto a la forma de programar y el número de expresiones condicionales utilizadas.

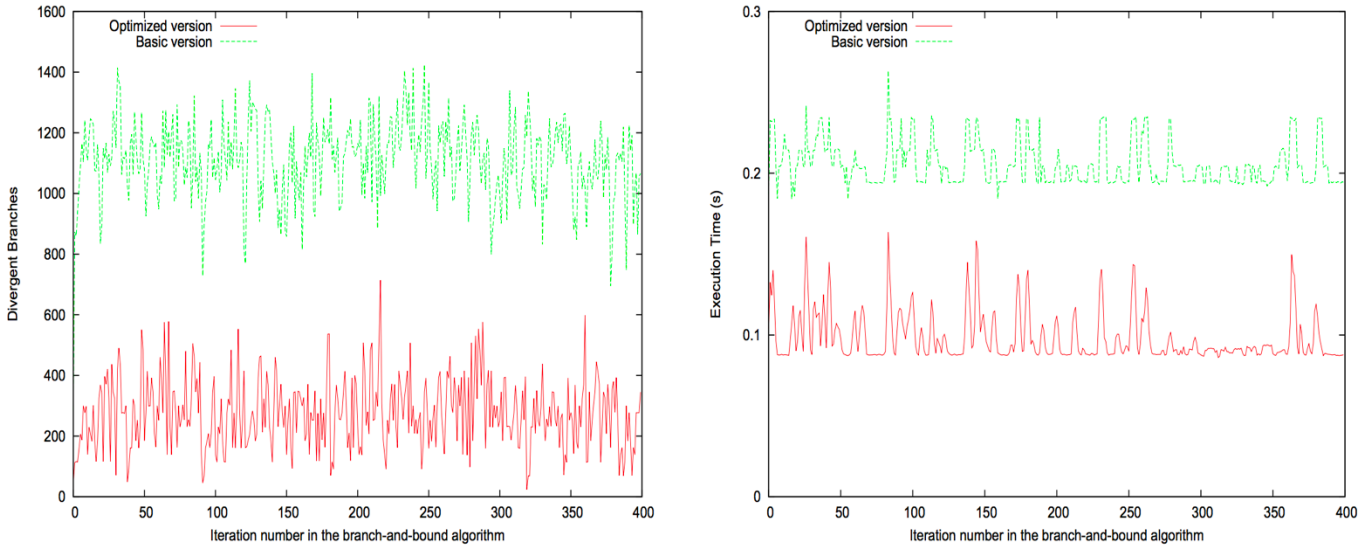

Imagen 40 - Aplicación divergente vs. Aplicación no divergente

La reducción del efecto de la divergencia se ha conseguido sincronizando las ejecuciones de los hilos existentes mediante una manipulación previa de los datos con técnicas *zero-padding* en los flujos de datos. Esta opción ha sido considerada respecto a otras alternativas, como por ejemplo la inclusión de barreras a nivel de programación o *barriers* que sincronicen los diferentes hilos de ejecución, pero el factor de añadir complejidad subyacente a nivel de diseño e implementación sobre la solución propuesta ha sido determinante por descartar otras alternativas en favor de la técnica seleccionada.

La tercera y última consideración a tener en cuenta, especialmente para grandes cantidades de datos, es la decisión de diseño de usar memoria global en lugar de memoria compartida, constante y memoria de texturas o viceversa. La principal diferencia entre estas múltiples memorias es que, mientras las memorias compartidas, constantes o de texturas son memorias internas de la propia GPU, la memoria global se corresponde con una zona mapeada de

memoria del disco duro. Por consiguiente, existen diferencias notorias en términos de tiempo de acceso a los datos que deben ser tenidas en cuenta para optimizar el rendimiento del sistema. Como se puede observar en la Imagen 41, las diferencias entre tiempos de acceso son cada vez más evidentes a medida que el tamaño de las secuencias de aminoácidos aumentan.

Esta decisión va a venir determinada principalmente por el número de restricciones a nivel de *capacity planning*, tales como el tamaño de memoria existente y penalizaciones de tiempo en transferencias de datos entre CPU y GPU.

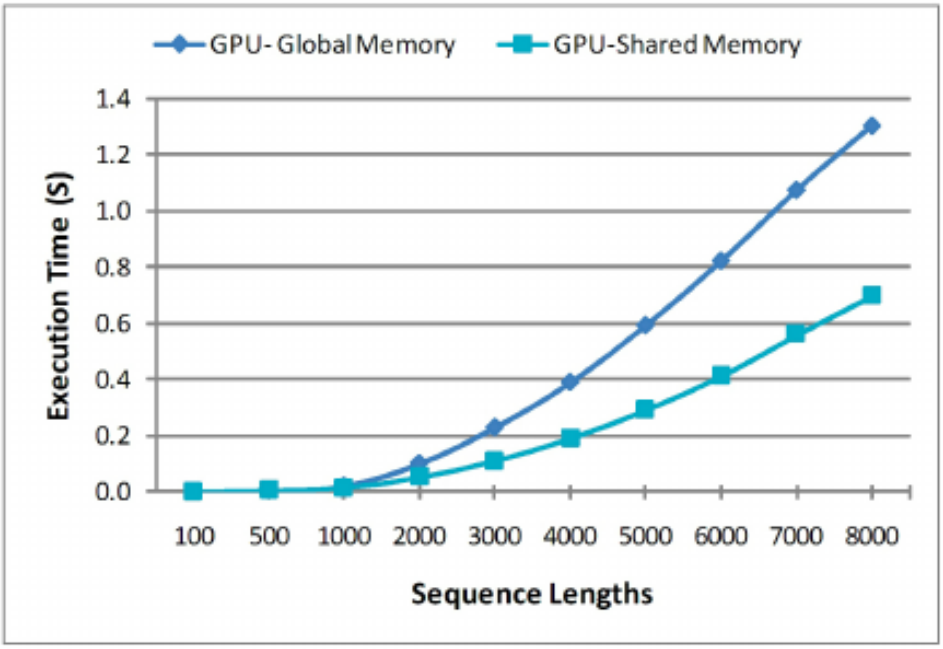

Imagen 41 - GPU Memoria Global vs. GPU Memoria Compartida

Capítulo 7 Evaluación

El siguiente capítulo tiene por objetivo describir el proceso de evaluación llevado a cabo para validar la solución propuesta en términos de precisión y rendimiento. Este proceso ha sido dividido en dos bloques principales y acordes con el tipo de alineamiento llevado a cabo: proteínas y metagenomas.

En primer lugar, del lado del alineamiento de proteínas se han utilizado las siguientes familias procedentes de diferentes filogenias:

- *Anaplasma Marginale (AMA)*
    - Este genoma de la familia de las proteo-bacterias es un patógeno procedente de las garrapatas y sus enfermedades [86]. Su genoma consta de 9.000 secuencias de proteínas.
- *Escherichia Coli (ECS)*
    - Este genoma de la familia de las proteo-bacterias es un patógeno procedente de los humanos [87]. Su genoma consta de 5.000 secuencias de proteínas.
- *Buchnera aphidicola (BAP)*
    - Este genoma es una bacteria simbiótica del grupo de las proteo-bacterias y consta de 1.000 secuencias [88].
- *Pseudomonas putida (PPU)*
    - Este genoma es una bacteria saprofita existente en la tierra que ha sido certificado como anfitrión bioseguro para el clonado de genes externos y procedente del grupo de las proteo-bacterias [89].

En segundo lugar, y formando parte de los alineamientos entre metagenomas, se han utilizado las siguientes familias:

- *GUT* microbiota o microbiota intestional
    - Este metagenoma se corresponde con el conjunto de microorganismos que residen en los intestinos humanos y que

permiten al cuerpo a digerir ciertos alimentos o combatir contra amenazas externas, ayudando así al sistema inmune.

- SOIL
    - Este metagenoma está compuesto por diferentes microorganismos que habitan en el medio ambiente del suelo y contribuyen de manera directa a la riqueza de nutrientes del mismo.
- WFALL1
    - Este metagenoma está formado por un conjunto de microorganismos no cultivados y extraídos a partir de un ADN de ballenas fallecidas.

Además, el proceso de evaluación ha sido completado con otras familias aleatorias de proteo-bacterias y virus poli-proteínicos, pero con la consideración que han sido seleccionadas por tener cadenas de aminoácidos de gran longitud con el objetivo de analizar diferentes comportamientos dentro de la GPU. Todas estas familias de proteínas han sido ejecutadas contra la base de datos de referencia *Genbank non-redundant*.

Este esquema de validación, tanto a nivel de precisión como a nivel de rendimiento, se corresponde con un modelo propuesto por los centros de investigación de referencia implicados en la presente tesis doctoral y que determina la fiabilidad de sus resultados.
El proceso de validación de la solución propuesta consiste en un procedimiento detallado que tiene por objetivo determinar si los resultados obtenidos son correctos en dos niveles precisión y rendimiento [85].

7.1. Pruebas de precisión

La primera fase de este proceso de evaluación ha consistido en la validación de la funcionalidad principal mediante la precisión de los resultados en la solución propuesta. Asimismo, este proceso de validación detallado tiene por objetivo

certificar la compatibilidad del sistema propuesto con la aplicación de referencia, en nuestro caso particular, BLAST.

El diseño de este proceso se ha basado en la comparación entre los resultados esperados, a partir de una ejecución previa del algoritmo de referencia con todas las secuencias sin filtrar, y los resultados obtenidos de la ejecución del algoritmo de referencia con las secuencias filtradas por la solución propuesta. Sin embargo y como ya detallamos en capítulos anteriores, la solución propuesta permite definir, mediante el uso de diferentes parámetros, el nivel de detalle del proceso de filtrado. Este nivel de detalle es representado a través del porcentaje de verosimilitud entre secuencias y ha sido considerado durante esta primera fase de evaluación.

A modo de resultado, el porcentaje de verosimilitud entre los diferentes alineamientos ha sido analizado desde dos puntos de vista diferentes. La primera aproximación se ha focalizado en la existencia de pequeños fragmentos de secuencias dentro de un genoma completo de referencia, es decir, se ha comprobado la inclusión de sub-secuencias dentro de una secuencia mayor. Ejemplos de genomas como *Anaplasma Marginale* (AMA) o *Escherichia Coli* (ECS) fueron seleccionados para ser comparados respecto a sus propios fragmentos de secuencias, como por ejemplo AMA1981 o ECS2480, y establecer una verdad fundamental o *ground-truth* que determine un correcto funcionamiento, a nivel de precisión de los resultados, en la solución propuesta. En todas las pruebas llevadas a cabo durante esta primera aproximación, la precisión de la solución propuesta ha sido de un 100% con una inclusión total de los fragmentos de secuencia respecto a su propio genoma de referencia. Además, una consideración adicional fue observada durante esta fase ya que nuevos alineamientos, anteriormente no considerados por el algoritmo de referencia, fueron detectados dada la reducción del espacio de búsqueda entre las secuencias de entrada y la base de datos de referencia. Este fenómeno ya ha sido descrito en capítulos anteriores analizando las diferentes líneas de investigación existentes y, aunque todavía no existen evidencias suficientes para

determinar que implica una mejora global de precisión del algoritmo de referencia, aporta el valor suficiente para que investigadores tengan un mayor volumen de datos para realizar tal discriminación.

Tras haber observado un buen comportamiento de la solución propuesta en la fase anterior, un segundo de nivel de pruebas fue llevado a cabo con el objetivo de tener una aproximación más realista y completar el ciclo de validación, en términos de precisión, en la solución propuesta. Esta aproximación, más cercana a los entornos científicos actuales, está basada en la búsqueda del mejor alineamiento entre dos genomas diferentes y completos. Por ello, ejemplos de genomas como *Escherichia Coli* (ECS), *Buchnera aphidicola* (BAP) o *Pseudomonas putida* (PPU) y la aplicación de referencia BLAST fueron seleccionados y obteniendo porcentajes de precisión del 70% en el escenario peor.

Como conclusión, tanto el aumento de precisión en la primera fase de pruebas como las diferencias en los porcentajes de precisión de la segunda fase tienen una explicación basada en términos del espacio de búsqueda y su relación respecto al esquema de puntuación utilizado para determinar la relevancia de un alineamiento. Por consiguiente y como comentamos anteriormente, a mayor espacio de búsqueda, menor precisión de los algoritmos y sus correspondientes resultados. Concretamente, algunos algoritmos de detección de homologías basados en procesadores *multicore*, como por ejemplo MPIBLAST, han demostrado esta dependencia estadística dividiendo el conjunto de datos original en bloques de menor tamaño para un procesamiento individual y en paralelo. En todos estos casos, se ha reducido el espacio de búsqueda y, por tanto, se ha aumentado la precisión de sus resultados.

7.2. Pruebas de rendimiento

La segunda fase del proceso de evaluación sobre la solución propuesta ha consistido en las pruebas de rendimiento. Estas pruebas tenían por objetivo

relacionar los resultados previos de las pruebas de precisión con un conjunto de indicadores de rendimiento seleccionados, como por ejemplo el tiempo de ejecución.

Las arquitecturas hardware empleadas durante estas pruebas se han dividido en dos escenarios distintos. Tal como se puede observar en la Imagen 42, el primer escenario está formado por un procesador *multicore* Intel Xeon E5506 con 8 *cores* físicos y con las funciones de *hyperthreading* deshabilitadas, 24 GBytes de memoria RAM y la tarjeta gráfica NVidia GTX590 formada a su vez por dos tarjetas gráficas interconectadas a través de un interfaz llamado SLI y 512 CUDA *cores* por GPU.

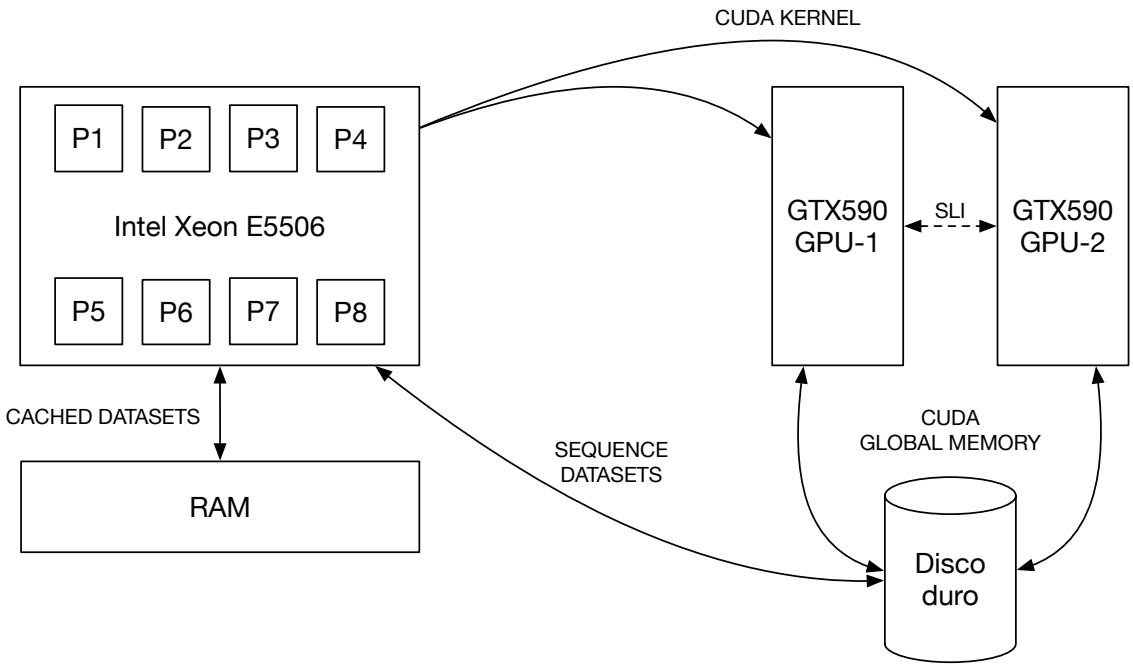

Imagen 42 – Primer escenario de evaluación

A continuación, y como se puede observar en la Imagen 43, el segundo escenario está formado por un procesador *multicore* Intel Xeon E5-2630 con 12 *cores* físicos y funciones de *hyperthreading* habilitadas, conexiones PCI Express 3.0, 32GB de memoria RAM y una tarjeta gráfica NVidia Tesla K40c, procedente de la familia Kepler, con 876MHz, 12GBytes de memoria RAM DDR5 y 2.880 CUDA *cores.* Además, este último escenario se encuentra ejecutándose bajo un

sistema operativo GNU/Linux Fedora 18 con arquitectura de 64bits y un *kernel* 3.11.10-100, CUDA *runtime* 6.0 y *compute capability* 3.5.

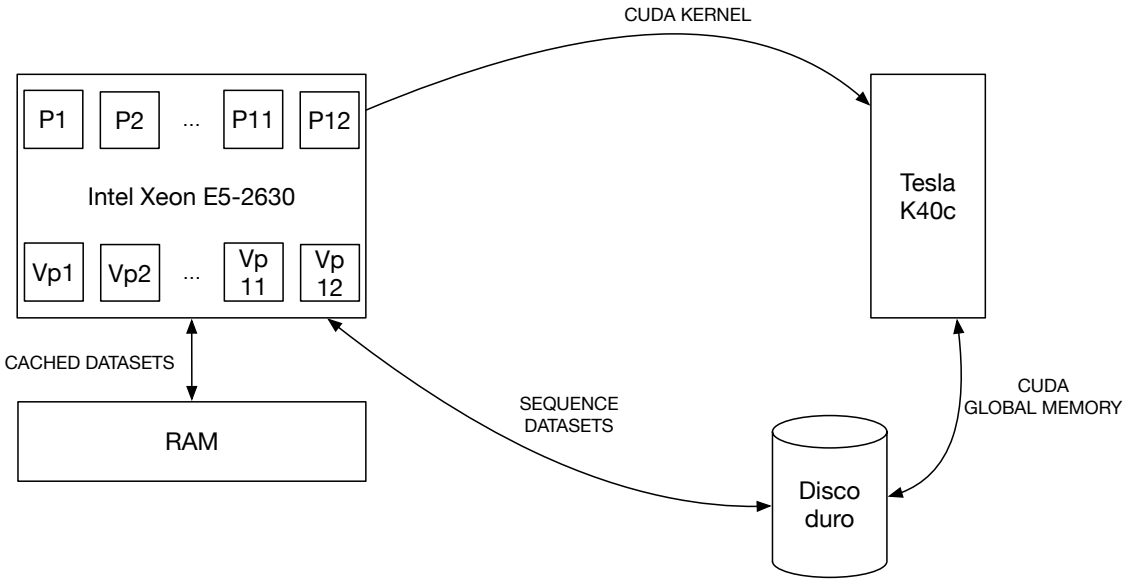

Imagen 43 - Segundo escenario de evaluación

En primer lugar, la tecnología propietaria de Intel *hyperthreading*, procedente de las arquitecturas *Simultaneous Multithreading* (SMT) de Intel, es capaz de convertir un procesador o *core* físico en dos lógicos. De esta manera y a partir que dicha funcionalidad es activada desde la BIOS y soportada dentro del *kernel* del sistema operativo, los recursos del procesador físico son compartidos, como por ejemplo cachés L2-L3 o buses, y el estado de la arquitectura duplicado por los dos procesadores o *cores* lógicos. Como se puede observar en la Imagen 44, esta arquitectura duplicada estará formada por una máquina de control de estados y registros de interrupciones o *Advanced Programmable Interrupt Controller* (APIC). La premisa inicial en la que se basa esta tecnología es que el proceso de duplicación permite al procesador o *core* físico ejecutar instrucciones desde diferentes hilos o *threads* en paralelo, por lo que permite un mejor aprovechamiento de utilización del procesador y mayor rendimiento generalizado. En términos generales, la activación de la funcionalidad *hyperthreading* puede mejorar el rendimiento de las aplicaciones con un elevado nivel de paralelismo. Sin embargo, existen una serie de consideraciones a tener

en cuenta que pueden afectar a la propia aportación de dicha funcionalidad sobre el comportamiento general del sistema y su rendimiento:

- Aquellas aplicaciones que lleven a cabo procesamientos exhaustivos o complejos no podrán aprovechas las ventajas del *hyperthreading* ya que los recursos de computación son compartidos por el resto de procesadores lógicos y altamente utilizados por lo que no será posible una paralelización de los procesos.
- Aquellas aplicaciones que usen de manera recurrente el acceso a caché pueden ver su rendimiento afectado, incluso negativamente, al ser un recurso compartido por el conjunto de procesadores lógicos. Esto puede provocar un elevado número de fallos de caché con su correspondiente penalización a nivel de rendimiento.
- Aquellas aplicaciones que usen de manera intensiva las comunicaciones a través de los diferentes buses o E/S del procesador físico pueden ver afectado el comportamiento de sus ejecuciones al ser éstos un cuello de botella potencial por los diferentes procesadores lógicos.

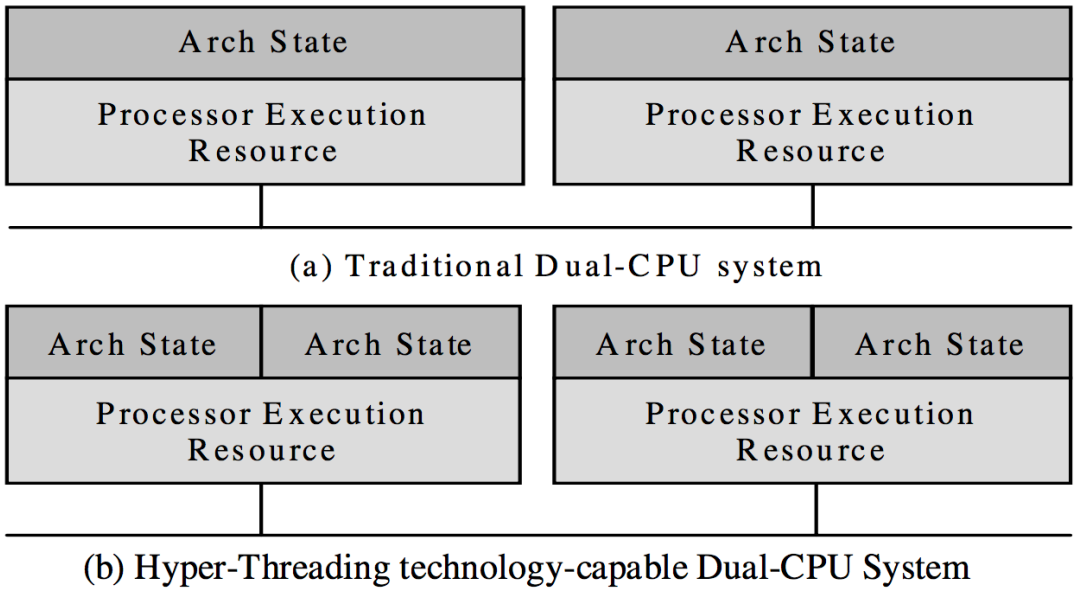

Imagen 44 - Comparativa hyperthreading

En segundo lugar, la interfaz de interconexión *NVidia's Scalable Link Interconnect* (SLI) tiene por objetivo incrementar la potencia computacional de una arquitectura basada en GPU mediante la interconexión de múltiples tarjetas en paralelo. De esta manera, aquellas aplicaciones basadas en CUDA podrán beneficiarse con mejoras de rendimiento significativas gracias a los diferentes algoritmos existentes de balanceo de carga que permitan dinámicamente ejecutar este tipo de acciones de distribución de tareas a bajo nivel. Actualmente, esta interfaz de interconexión ha sido renombrada en entornos profesionales como NVLink.

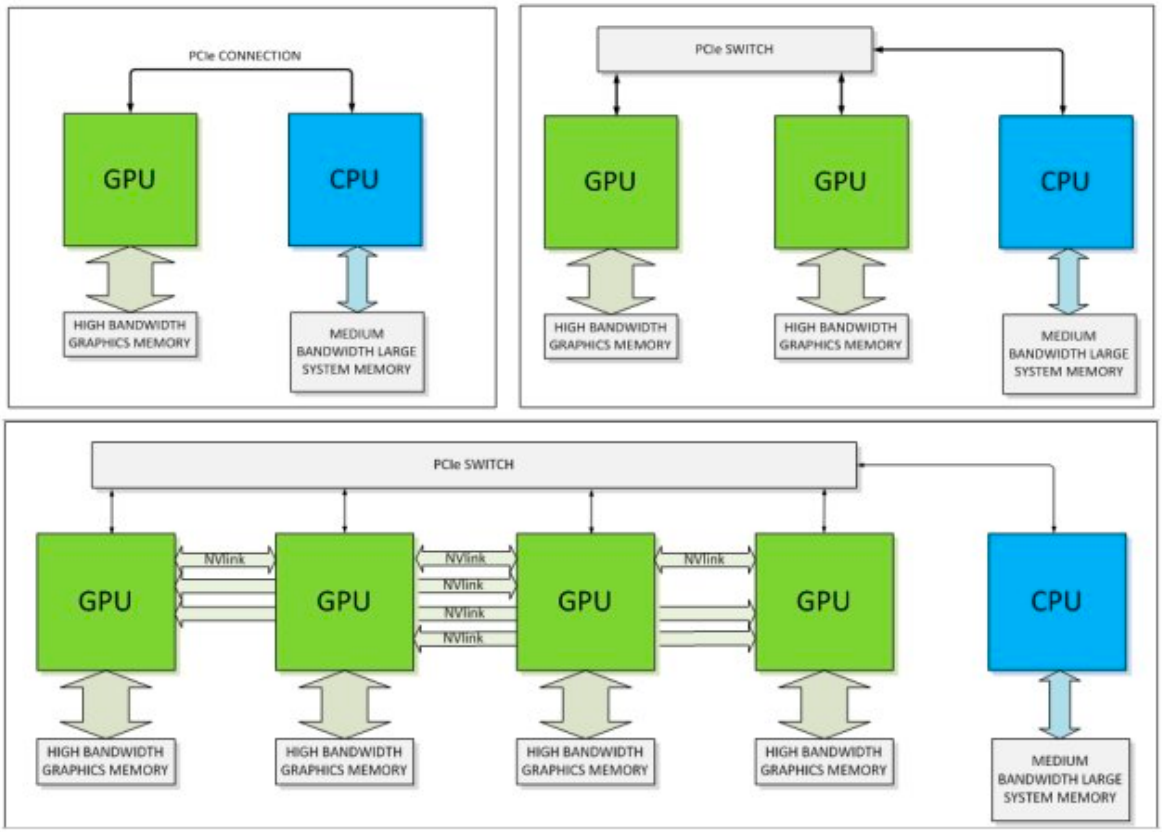

Imagen 45 - Arquitectura NVLink

Tal y como se puede observar en Imagen 45, NVLink se define como una interfaz de interconexión escalable, de bajo consumo energético y alto ancho de banda de transferencias en esquemas GPU-GPU y GPU-CPU con tasas de 80GBytes por segundo.

A continuación, en la sección de resultados, se presentan las mejoras obtenidas a partir de los dos escenarios prácticos propuestos y las familias de aminoácidos evaluadas frente a la base de datos de referencia *GenBank non-redundant*. En ambos casos, la solución propuesta ha alcanzado una mejora significativa respecto al algoritmo original con aceleraciones hasta cuatro veces mayor (4x).

7.3. Resultados

Una vez obtenidos los resultados a partir del proceso de evaluación, tendremos información suficiente para certificar la compatibilidad de la solución propuesta con la aplicación de referencia y, por tanto, otorgar una cierta validez y credibilidad a la solución propuesta respecto a otras alternativas existentes. Como resultado, estas pruebas han concluido con altos niveles de precisión:

- 70% en el caso peor para coincidencias exactas entre genomas completos.
- 100% para coincidencias de sub-secuencias sobre su genoma de referencia.

A modo de aclaración, y en lo que respecta al valor obtenido en el caso peor para coincidencias exactas entre genomas completos, las diferencias existentes entre los resultados obtenidos del algoritmo de referencia y aquellos obtenidos de la solución propuesta se han debido a ligeras variaciones entre los espacios de búsqueda de ambas soluciones que han resultado en diferencias poco significativas sobre los alineamientos obtenidos. Estos valores han sido validados por diferentes centros de investigación de relevancia que han concluido en las mismas afirmaciones.

Desde el punto de vista de la estructura de los datos, la base de datos de secuencias utilizada fue *GenBank non-redundant* con 3.163.461.953 aminoácidos y 9.230.955 secuencias. Usando como elemento común la base de

datos de referencia, se realizarán tres evaluaciones distintas sobre los escenarios prácticas descritos en el apartado anterior.

En primer lugar, se ha llevado a cabo una comparación entre diferentes familias de proteínas y la base de datos de referencia sobre el primer escenario práctico que describimos en la Imagen 42. Como podemos observar en la Tabla 2, existe una relativa diferencia entre los tiempos de ejecución de los alineamientos en las diversas familias evaluadas. Esto es debido a que, en este caso particular, ya no estamos filtrando secuencias individuales sino un conjunto de mayor tamaño con una elevada dispersión en las longitudes de cada una de las secuencias que conforman la familia correspondiente. Por consiguiente, este efecto aumenta la varianza en los tiempos de ejecución de los alineamientos y, por tanto, en su correspondiente aceleración. No obstante, en todos los casos evaluados se ha obtenido una mejora significativa con aceleraciones entre 2x y 5x.

Base de datos	**Secuencia de entrada**	**Número de secuencias**	**Tiempo BLAST (seg.)**	**Tiempo BLAST + Prefiltro (seg.)**	**Aceleración**
nr	AMA	9000	4740	1980	2.39
nr	ECS	5000	2820	540	5.22
nr	BAP	1000	420	240	1.75

Tabla 2 - Prestaciones en proteínas sobre el primer escenario

En segundo lugar, se ha llevado a cabo el mismo proceso de evaluación y sobre el mismo escenario práctico, pero en este caso sobre diferentes familias de metagenomas.

Base de datos	Secuencia de entrada	Número de secuencias	Tiempo BLAST (seg.)	Tiempo BLAST + Prefiltro (seg.)	Aceleración
nr	GUT	10000	12180	6120	2
nr	WFALL1	30000	23520	16200	1.5

Tabla 3 - Prestaciones en metagenomas sobre el primer escenario

Como podemos observar en la Tabla 3, el comportamiento de la solución propuesta difiere entre proteínas y metagenomas, experimentando niveles de aceleración inferiores en este segundo caso. Aunque no existe una certeza clara sobre el factor o factores que pueden influir en esta degradación del rendimiento, todas las investigaciones llevadas a cabo hasta la fecha apuntan a la complejidad estructural de los propios metagenomas como causante principal.

En tercer y último lugar, la comparación entre los conjuntos de datos de entrada y base de datos se ha focalizado en dos diferentes familias, virus y proteobacterias, y en relación al escenario práctico descrito en la Imagen 43. La razón de seleccionar estas familias es porque sus secuencias aportan longitudes suficientemente largas para llevar a cabo el escenario de caso peor para la gestión de memoria por la GPU.

Base de datos	Secuencia de entrada	Longitud	Tiempo BLAST (seg.)	Tiempo BLAST + Prefiltro (seg.)	Aceleración
Virus Poliproteínico					
nr	BAK61626.1	3161	779	243	3.21
nr	BAK61626.1	3161	779	325	2.40
nr	ABD34305.1	743	548	156	3.51
nr	ABD34305.1	743	548	256	2.14
nr	AAA45466.1	2225	700	199	3.52
nr	AAA45466.1	2225	700	311	2.25
Proteobacteria					
nr	AHW02111.1	2435	718	175	4.10
nr	AHW02111.1	2435	718	311	2.31
nr	AAD11553.1	542	522	150	3.48
nr	AAD11553.1	542	522	188	2.78
nr	AAO08121.1	1976	696	173	4.02
nr	AAO08121.1	1976	696	307	2.27

Tabla 4 - Prestaciones en proteínas sobre el segundo escenario

Tal como indica la Tabla 4, la solución propuesta es capaz de alcanzar una mejora de rendimiento significativa con aceleraciones de hasta 4x. Esta aceleración ha sido alcanzada mediante la comparación de tiempos de ejecución del pre-procesado de la base de datos (NCBI *formatdb*) más la aplicación de referencia NCBI BLASTP y la solución propuesta compuesta por el pre-procesamiento de la base de datos y el modelo de pre-filtrado. Esta mejora de rendimiento confirma el uso del sistema propuesto como herramienta para mejorar los tiempos de ejecución de aplicaciones de referencia en analizar secuencias. Sin embargo, estos resultados y su correspondencia a nivel de rendimiento son dependientes de la filogenia molecular de cada secuencia. Por consiguiente y como comentaremos en capítulos posteriores, queda pendiente como trabajo futuro extender el proceso de evaluación para más familias de secuencias y establecer relaciones entre el proceso de filtrado y su filogenia.

Base de datos	Secuencia de entrada	Longitud	Umbral de verosimilitud (%)	Porcentaje de filtrado (%)	Aceleración
Virus Poliproteínico					
nr	BAK61626.1	3161	95	91.30	3.21
nr	BAK61626.1	3161	60	83.79	2.40
nr	ABD34305.1	743	95	98.20	3.51
nr	ABD34305.1	743	60	87.17	2.14
nr	AAA45466.1	2225	95	95.10	3.52
nr	AAA45466.1	2225	60	83.83	2.25
Proteobacteria					
nr	AHW02111.1	2435	95	97.20	4.10
nr	AHW02111.1	2435	60	83.98	2.31
nr	AAD11553.1	542	95	98.64	3.48
nr	AAD11553.1	542	60	51.39	2.78
nr	AAO08121.1	1976	95	97.35	4.02
nr	AAO08121.1	1976	60	84.04	2.27

Tabla 5 - Evaluación de Prestaciones en filtrado

En segunda instancia, la Tabla 5 muestra la relevancia de la solución propuesta como eliminador de secuencias irrelevantes. Este proceso tiene por objetivo descartar aquellas secuencias de la base de datos con un porcentaje de verosimilitud inferior a un cierto valor introducido como parámetro. Gracias a ello, es posible ayudar a otras aplicaciones de referencia a acelerar sus algoritmos sin necesidad de re-implementar las fases de procesamiento.

Sin embargo y de acuerdo con la ley de Amdahl's, el rendimiento de la solución propuesta está directamente relacionada con la probabilidad de verosimilitud introducida como parámetro. Por consiguiente, esta entrada determina el número de secuencias que la aplicación de referencia analizará. En relación con lo comentado, esta tercera etapa de la fase de evaluación estará formada por dos escenarios diferentes. En primer lugar, el escenario de caso mejor, con aceleraciones de 4x, define altos porcentajes de verosimilitud (95%) y

descartando un gran volumen de secuencias (90%) de la base de datos. En segundo lugar, se encuentra el escenario de caso peor, con aceleraciones de 2x, reportando porcentajes de verosimilitud relativamente bajos (60%) y descartando valores promedios de secuencias procedentes de la base de datos (50%).

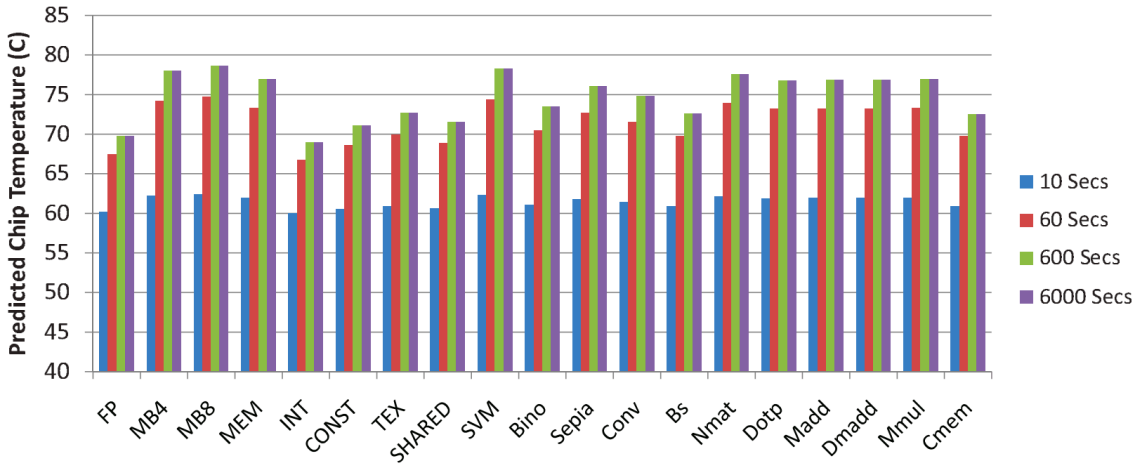

Imagen 46 - Pruebas de temperatura sobre GPU

Asimismo, durante las pruebas también se ha evaluado la temperatura y condiciones físicas de la GPU. De acuerdo con investigaciones previas [90] y como se puede observar en la Imagen 46, los valores de temperaturas obtenidos durante las diferentes etapas de la fase de evaluación han estado muy cerca del valor del reposo (57ºC), significando un comportamiento estándar que aporta datos suficientes para demostrar que la solución propuesta es capaz de alcanzar ahorros de energía relevantes respecto a otras soluciones alternativas de grandes prestaciones.

Capítulo 8 Conclusiones

El vertiginoso incremento, tanto de las fuentes de información como de sus propios datos procedentes de secuencias de proteínas, nucleótidos o metagenomas está llevando a replantearse el actual, y tradicional, esquema de secuenciación y procesamiento de los datos. Este nuevo paradigma, denominado *genome big data*, ha desembocado en la aparición de los actuales sistemas de secuenciación de nueva generación (NGS). Estos sistemas han revolucionado los actuales algoritmos de detección de homologías hasta el punto de dejarlos completamente obsoletos. De esta manera y como se puede observar en la Imagen 47, aplicaciones de detección de homologías tradicionales como BLAST [5], Smith-Waterman [3] o BLAT [84] han sido substituidos por otros algoritmos más eficaces como RAPSearch2 [91].

Database	Query			Running time (min)				
	Dataset	Number of reads	Read length (nt)	BLAST[a]	RAPSearch	RAPSearch2 (# threads)		
						1	4	8
IMG (1.6G)	SRR020796[b]	1 164 805	72	95 800	1170	587	170	100
	4440037[c]	188 445	100	9240	378	120	36	22
	TS28[d]	622 554	200	67 000	3872	1341	331	242
	TS50[d]	312 665	329	39 200	4105	1512	385	281
NCBI NR (3.2G)	SRR020796			271 000	2910	1229	362	250
	4440037			25 680	889	335	110	58
	TS28			177 900	8471	3019	859	518
	TS50			103 600	9195	3545	901	644

Imagen 47 - BLAST vs. RAPSearch vs. RAPSearch2

A consecuencia de este cambio de paradigma, múltiples y diversas líneas de investigación están llevándose a cabo en la actualidad con el objetivo de mejorar estos algoritmos tradicionales en términos de rendimiento y precisión. Sin embargo, todas estas líneas de investigación llevan consigo una serie de características comunes y están clasificadas en tres categorías principalmente:

- Métodos predictivos.
- Métodos basados en *hardware* de altas prestaciones (HPC).

- Métodos basados en el modelado o filtrado.

Lo más relevante de esta categorización es que, ninguno de estos métodos es excluyente respecto al resto, por lo que pueden complementarse a la perfección con el objetivo de obtener el mejor resultado posible.

En nuestro caso particular, se ha optado por la combinación de un método de filtrado aplicado a *hardware* de altas prestaciones. La naturaleza altamente paralelizable de las GPU y la flexibilidad del lenguaje de programación CUDA, ha convertido a NVidia como el mejor candidato existente para la implementación del sistema propuesto. Dicho sistema es capaz de secuenciar grandes cantidades de aminoácidos, nucleótidos o metagenomas a bajo coste, mediante el uso de *hardware* de propósito general, y con mejoras significativas de rendimiento alcanzando tiempos hasta cuatro veces mayor que con ejecuciones de algoritmos convencionales.

Al contrario que otras soluciones existentes hasta la fecha y basadas en GPU o FPGA [15][16][17], la solución propuesta es completamente independiente del algoritmo original dado que no está basado en la re-implementación de todas sus etapas de procesamiento. Como resultado de ello, se ha obtenido una solución flexible que puede ser empleada con otros algoritmos de detección de homologías, aparte de la aplicación de referencia utilizado durante la fase de evaluación, BLAST, e incluso con futuras versiones del mismo que sean desarrolladas a futuro.

Por último y a diferencia de investigaciones llevadas a cabo hasta la fecha, el proceso de evaluación se ha llevado a cabo desde diferentes tipos de familias de proteínas, conformando un volumen de datos lo suficientemente significativo para certificar la solución en entornos profesionales y llevar su uso a la explotación industrial como describiremos en el siguiente capítulo.

Todo el código fuente, documentación e instrucciones de instalación están disponibles en Github [30] como dominio público y bajo licenciamiento MIT.

Capítulo 9 Divulgación de resultados

La divulgación de los resultados obtenidos en la presente tesis doctoral se ha estructurado en dos bloques principales: diseño e implementación de la solución propuesta. Además, de manera complementaria a estas fases, se han incluido otros elementos como el sistema de almacenamiento distribuido o los mecanismos de transferencia de secuencias a través de una red de comunicaciones (Internet) para completar el producto comercial.

La fase de diseño, comprendida a lo largo de los capítulos séptimo, octavo y noveno, estuvo centrada especialmente en el diseño de la solución propuesta. Estas aportaciones tenían por objetivo definir el concepto de *hit* y su correspondiente principio de proximidad para que, a través del modelo teórico de Karlin-Altschul, se pudiera fundamentar el diseño en un modelo teórico básico. De esta manera, los resultados obtenidos a partir de las siguientes aportaciones generaron las siguientes publicaciones:

- Retamosa, G., de Pedro, L., Gonzalez, I., & Tamames, J. (2014). High performance genomic sequencing: a filtered approach. In *8th International Conference on Practical Applications of Computational Biology & Bioinformatics (PACBB 2014)* (pp. 129-136). Springer International Publishing.
  Tasa de aceptación 2014: 39/61 (64%)
- Retamosa, G., de Pedro, L., Gonzalez, I., Aracil, J. & Tamames, J. (2012). naudit Filter: using graphic processors to accelerate homology searching. In *11th Spanish Symposium on Bioinformatics (JBI 2012)*, Barcelona, Spain.

En segundo lugar, se procedió con la fase de implementación comprendida en los mismos capítulos que en la fase anterior, pero en este caso, centrado en el análisis y descripción de la implementación en el sistema propuesto. Estas aportaciones se focalizaron en el análisis de las diferentes arquitecturas

posibles, así como en las diferentes problemáticas que se fueron encontrando durante el ciclo de vida del desarrollo del sistema propuesto. Además, consideraciones adicionales fueron incorporadas a nivel de optimización de la aplicación de referencia BLAST dentro del sistema de computación de altas prestaciones GPU, como por ejemplo los efectos de coalescencia y divergencia. Finalmente, el trabajo de investigación fue completado con el proceso de evaluación, en términos de precisión y rendimiento, por los centros de referencia CNB-CSIC y CSISP, y generando las siguientes publicaciones:

- Retamosa, G., de Pedro, L., González, I., & Tamames, J. (2016). Prefiltering Model for Homology Detection Algorithms on GPU. *Evolutionary Bioinformatics Online*, *12*, 313.
  Journal Impact Factor: 1.404. Journal Rank: 30/56 (T2 - Q3). Category: Mathematical and Computational Biology (obtained from the 2015 JCR).

Finalmente, y como se detalla en el capítulo de aplicaciones industriales de la tesis doctoral, el desarrollo de la solución comercial llevó a cabo un estudio del estado del arte sobre soluciones de almacenamiento distribuido y mecanismos de transferencia de grandes volúmenes de información a través de las redes de comunicaciones. En ambos casos, las aproximaciones abordadas fueron evaluadas con herramientas de código abierto (*open-source*) y con el objetivo de determinar su rendimiento efectivo y sin repercutir en costes adicionales a la empresa. A consecuencia de dicho estudio, se generaron las siguientes publicaciones:

- Retamosa de Ágreda, G., & Aracil, J. (2012). Performance evaluation of open-source software for traffic traces manipulation and analysis. *Infocommunications journal*.
  Journal Impact Factor: 0.101. Journal Rank: T3 – Q4. Category: Computer Science (obtained from the 2012 SJR).

Capítulo 10 Aplicaciones industriales

Desde el punto de vista académico, y como hemos ido observando a través de los diferentes capítulos de la tesis doctoral, el presente trabajo de investigación ha contribuido con el diseño e implementación de un sistema de pre-filtrado de altas prestaciones optimizada para GPU. El objetivo de dicho sistema consiste en mejorar, tanto a nivel de rendimiento como a nivel de precisión, los resultados procedentes de las aplicaciones de detección de homologías tradicionales, como por ejemplo BLAST.

Todas estas contribuciones han sido publicadas y contrastadas a través de diversas publicaciones en revistas y conferencias internacionales de alto impacto que corroboran su validez y relevancia dentro del ámbito de la biología computacional y la bioinformática.

Sin embargo, la aplicación de la tesis doctoral no se ha restringido únicamente al ámbito académico, sino que también ha sido utilizada en entornos industriales mediante procesos de transferencia tecnológica y en actual comercialización por la empresa Naudit High Performance Computing and Networking S.L. En lo que respecta a los procesos de transferencia tecnológica, se han llevado a cabo varios procesos con centros de investigación de referencia como el Centro Nacional de Biotecnología (CNB-CSIC) y el Centro Superior de Investigación y Salud Pública de Valencia (CSISP). La utilización de la solución propuesta dentro de los centros de investigación mencionados ha contribuido en la reducción de tiempos de espera sobre ejecuciones en las aplicaciones de detección de homologías, como por ejemplo BLAST, utilizados en diversas investigaciones llevadas a cabo en cada uno de los centros de investigación.

De manera complementaria a los procesos de transferencia tecnológica llevados a cabo con diversos centros de referencia que ha servido para certificar que la solución es apta para una explotación comercial, diferentes aportaciones como el desarrollo del sistema de almacenamiento distribuido y el sistema de

visualización, extracción y tratamiento de grandes volúmenes de información han sido llevados a cabo para cerrar el producto comercial. En el primer caso, este sistema fue utilizado dentro del servicio de genómica del Parque Científico de Madrid. En el segundo caso, se trata de un servicio profesional ofrecido por Naudit High Performance Computing and Networking S.L. y que actualmente se encuentra desplegado en empresas multinacionales como BBVA, Telefónica o Correos.

En todos estos casos, el proceso de transferencia tecnológica y comercialización ha sido llevado a cabo a través la empresa Naudit High Performance Computing and Networking S.L. Naudit es una empresa de base tecnológica creada como una *spin-off* de dos universidades públicas: la Universidad Autónoma de Madrid (UAM) y la Universidad Pública de Navarra (UPNA). Además, Naudit forma parte de la iniciativa de Campus de Excelencia Internacional de la UAM.

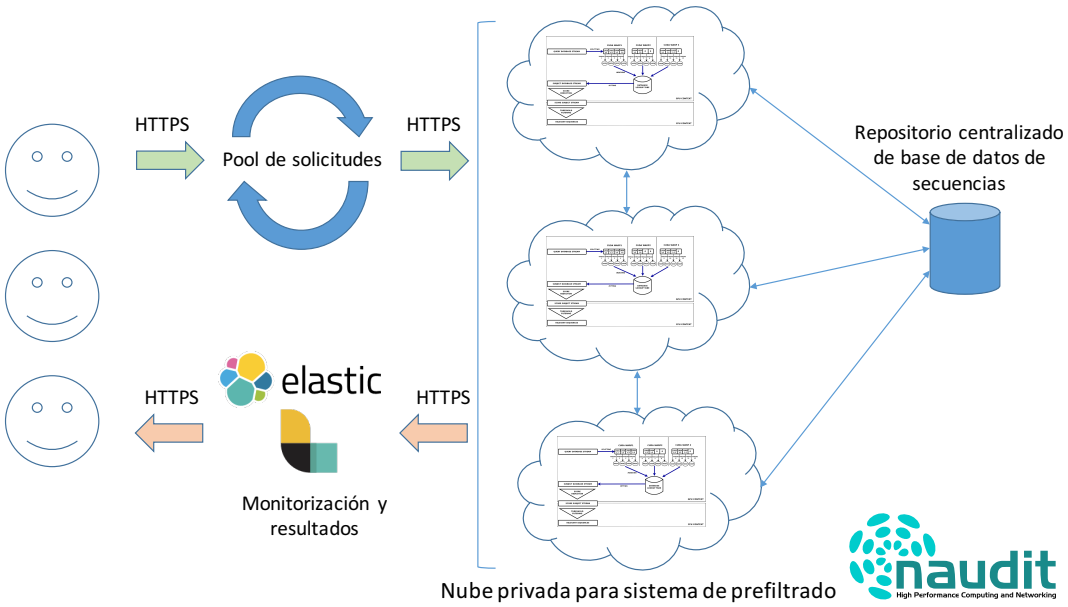

Imagen 48 - Arquitectura comercial Naudit HPCN S.L.

La Imagen 48 representa la arquitectura desarrollada en la empresa Naudit High Performance Computing and Networking para la comercialización del sistema de pre-filtrado de secuencias propuesto en la presente tesis doctoral. Tal y como se

puede observar en dicha arquitectura, filosofías como *big data*, *cloud computing* y *next-generation sequencing* confluyen en la actual solución donde prevalecen la utilización de procesos de desarrollo propios con tecnologías *open-source* frente a licencias de productos comerciales.

El procedimiento principal de secuenciación comienza dentro de un frontal web que, mediante diversas interfaces de usuario, solicita la información necesaria para el proceso de secuenciación, como por ejemplo la base de datos de referencia, el conjunto de secuencias de entrada y el porcentaje de verosimilitud. Una vez realizada la solicitud, se enviará mediante servicios REST la información asociada a un gestor de colas o *pool* de eventos que procesará la información de manera secuencial por solicitud de usuario y enviará a la nube privada de cada usuario los datos asociados al procesamiento.

Llegados a este punto, es importante destacar la importancia de las redes de comunicaciones en este proceso y su relación con el rendimiento del sistema. Por ello, se han llevado a cabo diferentes estudios sobre herramientas de evaluación en redes de comunicaciones que, mediante el análisis de tráfico, permitan determinar cuellos de botella en la transmisión de datos que afecten al rendimiento global del sistema [92]. En nuestro caso particular, el procedimiento de transmisión de datos seleccionado ha sido en *streaming* que, junto con los procedimientos de transmisión de datos de la GPU, permitirá reducir los tiempos de espera entre fases y mitigar posibles efectos de congestión en las comunicaciones.

Una vez recibida la información asociada al procesamiento en la nube privada, se llevará a cabo el proceso de pre-filtrado siguiendo la metodología descrita en capítulos anteriores y compuesta de las siguientes tareas principales[22], [23]:

- Pre-formateo de las secuencias de entrada.
- *map-reduce* entre secuencias de entrada y secuencias de la base de datos:

    - [*map*] Búsqueda de coincidencias entre ambos bloques de secuencias.
    - [*reduce*] Cálculo del valor final promedio de coincidencia.
- Filtrado de secuencias por el parámetro de entrada correspondiente al porcentaje de verosimilitud.

En lo que respecta al sistema de almacenamiento de secuencias utilizado, una serie de factores técnicos han sido tenidos en cuenta de cara a obtener el mejor esquema de datos para nuestra solución:

- **Escalabilidad**. La configuración en modo *cluster* de elasticsearch permite incorporar nodos principales de manera dinámica y sin afectar al resto de datos del sistema. Cada uno de los datos introducidos se repartirán entre los diferentes nodos del *cluster* aumentando su paralelismo.
- **Seguridad**. El esquema de seguridad en elasticsearch provee mecanismos a nivel de transporte, tanto a nivel interno del *cluster* como a nivel externo dentro de su API REST de comunicaciones con servicios de terceros.
- **Tolerancia a fallos**. La configuración en modo *cluster* de elasticsearch permite definir nodos *replica* que entren en escena en caso de caer sus nodos principales correspondientes. Este proceso se realiza dinámicamente al existir mecanismos de comunicación internos entre los nodos del *cluster*.
- **Flexibilidad**. El esquema de datos utilizado en elasticsearch es *schema-free*, de la misma manera que las bases de datos no relacionales, por lo que los datos del sistema pueden evolucionar en cualquier dirección, como por ejemplo con nuevos campos, sin afectar al comportamiento general de la solución.
- **Bajo coste**. Las soluciones basadas en elasticsearch son soluciones *open-source* por lo que, salvo casos muy concretos donde se requieran funcionalidades adicionales, no requieren licenciamientos con costes adicionales.

Finalmente, se retornarán los resultados obtenidos en un cuadro de mando integrado con diversas opciones de *alerting*, *reporting* y diferentes opciones de visualización de grandes volúmenes de información basados en los sistemas *open-source* elasticsearch, kibana y grafana.

Por último, los resultados dentro del presente trabajo de investigación han sido aplicados en los siguientes escenarios:

- **Secuenciación de proteínas:** Dentro de las múltiples investigaciones llevadas a cabo por los centros de investigación implicados en el ámbito de las proteínas, la solución propuesta ha contribuido en la reducción del tamaño sobre los conjuntos de datos de entrada de los experimentos, reduciendo así su tiempo de ejecución.
- **Secuenciación de nucleótidos:** De la misma manera que en el caso de uso anterior, pero dentro del ámbito de los nucleótidos, la solución propuesta ha contribuido en la reducción del tamaño sobre los conjuntos de datos de entrada de los experimentos, reduciendo así su tiempo de ejecución y aumentando la precisión del algoritmo tradicional.
- **Secuenciación de metagenomas:** Siguiendo el mismo ejemplo de casos anteriores, la solución propuesta ha obtenido una serie de mejoras significativas en el alineamiento de secuencias mediante el algoritmo BLASTX en términos de rendimiento y precisión.

Capítulo 11 Trabajos futuros

Gracias al vertiginoso avance en las técnicas de secuenciación de nueva generación (NGS) y exponencial aumento del volumen de datos en términos de secuencias de proteínas, nucleótidos o metagenomas, las posibilidades de trabajos futuros son muy extensas y prometedoras.

En primer lugar, un posible trabajo futuro estaría centrado en un estudio comparativo entre la solución propuesta, basado en técnicas de filtrado, respecto a las técnicas predictivas existentes hasta la fecha. Asimismo, un trabajo futuro, derivado de la línea de investigación actual, sería el propio comportamiento y posible mejora en términos de precisión y rendimiento de todas estas técnicas de predicción en combinación con *hardware* de altas prestaciones (HPC). De esta manera, podríamos alcanzar un punto de vista más global y objetivo de todo el ecosistema de detección de homologías mediante *hardware* de altas prestaciones y seleccionar el método que mejor relación rendimiento-precisión ofrezca.

En segundo lugar, otro posible trabajo futuro estaría centrado en la integración de la solución propuesta con los actuales métodos de modelado de secuencias según su filogenia. De esta manera y mediante las técnicas *hardware* de altas prestaciones (HPC) utilizadas hasta la fecha, evaluaríamos el comportamiento en términos de precisión y rendimiento de los procesos de alineamiento entre secuencias (*sequence-sequence*) respecto a los procesos de alineamiento entre perfiles o modelos de secuencias (*profile-profile*) y con el objetivo de determinar qué método ofrece una mejor relación rendimiento-precisión.

En tercer y último lugar, queda pendiente como parte de trabajos futuros se llevaría a cabo este mismo proceso de evaluación con otros modelos de GPU, como por ejemplo NVidia GTX Titan procedente de la arquitectura de GPU Kepler con 837MHz, 6GB DDR5 de memoria RAM y 2.688 CUDA *cores*. Las investigaciones preliminares previas indican que este tipo de GPU es capaz de

proporcionar rendimientos similares a la tarjeta evaluada (NVidia Tesla K40c) pero con un ahorro económico significativo. Asimismo, dada la acelerada evolución y demanda de las GPU en la actualidad, este mismo trabajo futuro podría extenderse a su aplicación en entornos móviles, como por ejemplo tabletas o teléfonos móviles inteligentes, con el fin de cerrar el ciclo de secuenciación de nueva generación y altas prestaciones al usuario final de la aplicación. Al contrario que estudios previos, esta comparación puede proporcionar un valor añadido a la línea de investigación actual ofreciendo una perspectiva diferente del lado del *hardware* convencional y computación de alto rendimiento en el ámbito de la biomedicina.

Bibliografía

[1] J. Manyika *et al.*, «Big data: The next frontier for innovation, competition, and productivity | McKinsey & Company». [En línea]. Disponible en: http://www.mckinsey.com/business-functions/digital-mckinsey/our-insights/big-data-the-next-frontier-for-innovation. [Accedido: 11-may-2017].

[2] IBM, P. Zikopoulos, y C. Eaton, *Understanding Big Data: Analytics for Enterprise Class Hadoop and Streaming Data*, 1st ed. McGraw-Hill Osborne Media, 2011.

[3] T. F. Smith y M. S. Waterman, «Identification of common molecular subsequences», *J. Mol. Biol.*, vol. 147, n.º 1, pp. 195-197, 1981.

[4] D. J. White, «Dynamic programming», RPRT, 1969.

[5] S. F. Altschul, W. Gish, W. Miller, E. W. Myers, y D. J. Lipman, «Basic local alignment search tool», *J. Mol. Biol.*, vol. 215, n.º 3, pp. 403-410, 1990.

[6] W. R. Pearson, «Flexible Sequence Similarity Searching with the FASTA3 Program Package», en *Bioinformatics Methods and Protocols*, New Jersey: Humana Press, 1999, pp. 185-219.

[7] M. Sniedovich, *Dynamic Programming: Foundations and Principles, Second Edition*. CRC Press, 2010.

[8] A. Jacob, J. Lancaster, J. Buhler, B. Harris, y R. D. Chamberlain, «Mercury BLASTP», *ACM Trans. Reconfigurable Technol. Syst.*, vol. 1, n.º 2, pp. 1-44, jun. 2008.

[9] K. Muriki, K. D. Underwood, y R. Sass, «RC-BLAST: Towards a Portable, Cost-Effective Open Source Hardware Implementation», en *19th IEEE International Parallel and Distributed Processing Symposium*, 2005, p. 196b-196b.

[10] D. Lavenier, L. Xinchun, y G. Georges, «Seed-based genomic sequence comparison using a FPGA/FLASH accelerator», en *2006 IEEE International Conference on Field Programmable Technology*, 2006, pp. 41-48.

[11] E. Sotiriades y A. Dollas, «A General Reconfigurable Architecture for the BLAST Algorithm», *J. VLSI Signal Process. Syst. Signal Image Video Technol.*, vol. 48, n.º 3, pp. 189-208, ago. 2007.

[12] I. T. Li *et al.*, «160-fold acceleration of the Smith-Waterman algorithm using a field programmable gate array (FPGA)», *BMC Bioinformatics*, vol. 8, n.º 1, p. 185, 2007.

[13] X. Jiang, X. Liu, L. Xu, P. Zhang, y N. Sun, «A Reconfigurable Accelerator for Smith–Waterman Algorithm», *IEEE Trans. Circuits Syst. II Express Briefs*, vol. 54, n.º 12, pp. 1077-1081, dic. 2007.

[14] J. Allred, J. Coyne, W. Lynch, V. Natoli, J. Grecco, y J. Morrissette, «Smith-Waterman implementation on a FSB-FPGA module using the Intel Accelerator Abstraction Layer», en *2009 IEEE International Symposium on Parallel & Distributed Processing*, 2009, pp. 1-4.

[15] W. Weiguo Liu, B. Schmidt, y W. Muller-Wittig, «CUDA-BLASTP: Accelerating BLASTP on CUDA-Enabled Graphics Hardware», *IEEE/ACM Trans. Comput. Biol. Bioinform.*, vol. 8, n.º 6, pp. 1678-1684, nov. 2011.

[16] P. D. Vouzis y N. V. Sahinidis, «GPU-BLAST: using graphics processors to accelerate protein sequence alignment», *Bioinformatics*, vol. 27, n.º 2, pp. 182-188, ene. 2011.

[17] S. Xiao, H. Lin, y W. Feng, «Accelerating Protein Sequence Search in a Heterogeneous Computing System», en *2011 IEEE International Parallel & Distributed Processing Symposium*, 2011, pp. 1212-1222.

[18] Y. Liu *et al.*, «CUDASW++: optimizing Smith-Waterman sequence database searches for CUDA-enabled graphics processing units», *BMC Res. Notes*, vol. 2, n.º 1, p. 73, 2009.

[19] Y. Liu *et al.*, «CUDASW++2.0: enhanced Smith-Waterman protein database search on CUDA-enabled GPUs based on SIMT and virtualized SIMD abstractions», *BMC Res. Notes*, vol. 3, n.º 1, p. 93, 2010.

[20] S. A. Manavski *et al.*, «CUDA compatible GPU cards as efficient hardware accelerators for Smith-Waterman sequence alignment», *BMC*

Bioinformatics, vol. 9, n.º Suppl 2, p. S10, 2008.

[21] B. Liu, J. Chen, y X. Wang, «Protein remote homology detection by combining Chou's distance-pair pseudo amino acid composition and principal component analysis», *Mol. Genet. Genomics*, vol. 290, n.º 5, pp. 1919-1931, oct. 2015.

[22] B. Liu, J. Chen, y X. Wang, «Application of learning to rank to protein remote homology detection», *Bioinformatics*, vol. 31, n.º 21, pp. 3492-3498, nov. 2015.

[23] J. Chen, R. Long, X. Wang, B. Liu, y K.-C. Chou, «dRHP-PseRA: detecting remote homology proteins using profile-based pseudo protein sequence and rank aggregation», *Sci. Rep.*, vol. 6, sep. 2016.

[24] M. N. Abdul Rahman, M. Y. Mohd. Saman, A. Ahmad, y A. O. Md. Tap, «A filtering algorithm for efficient retrieving of DNA sequence», *Int. J. Comput. Theory Eng.*, vol. 1, n.º 2, 2009.

[25] P. Afratis, E. Sotiriades, G. Chrysos, S. Fytraki, y D. Pnevmatikatos, «A rate-based prefiltering approach to blast acceleration», en *2008 International Conference on Field Programmable Logic and Applications*, 2008, pp. 631-634.

[26] G. Retamosa, L. de Pedro, I. Gonzalez, y J. Tamames, «High Performance Genomic Sequencing: A Filtered Approach», en *8th International Conference on Practical Applications of Computational Biology & Bioinformatics (PACBB 2014)*, Springer International Publishing, 2014, pp. 129-136.

[27] G. Retamosa, L. de Pedro, I. González, J. Aracil, y J. Tamames, «naudit Filter: using graphic processors to accelerate homology searching», presentado en Spanish Symposium on Bioinformatics (JBI), Barcelona (Spain), 2012.

[28] J. H. Park, Y. Qiu, y M. C. Herbordt, «CAAD BLASTn: Accelerated NCBI BLASTn with FPGA prefiltering», en *Proceedings of 2010 IEEE International Symposium on Circuits and Systems*, 2010, pp. 3797-3800.

[29] D. A. Benson *et al.*, «GenBank», *Nucleic Acids Res.*, vol. 41, n.º D1, pp. D36-D42, ene. 2013.

[30] G. Retamosa, «Prefilter BLAST», 2016. [En línea]. Disponible en: https://github.com/gretamosa/prefilter-blast.

[31] S. B. Needleman y C. D. Wunsch, «A general method applicable to the search for similarities in the amino acid sequence of two proteins», *J. Mol. Biol.*, vol. 48, n.º 3, pp. 443-453, mar. 1970.

[32] R. Schwarz, M. Dayhoff, y B. Orcutt, «A model of evolutionary change in proteins», *Atlas Protein Seq. Struct.*, pp. 345–352.

[33] W. Zhou, Z. Cai, B. Lian, J. Wang, y J. Ma, «Protein database search of hybrid alignment algorithm based on GPU parallel acceleration», *J. Supercomput.*, pp. 1-18, abr. 2017.

[34] D. S. Hirschberg, «A Linear Space Algorithm for Computing Maximal Common Subsequences», *Commun ACM*, vol. 18, n.º 6, pp. 341–343, jun. 1975.

[35] L. Bergroth, H. Hakonen, y T. Raita, «A survey of longest common subsequence algorithms», en *Proceedings Seventh International Symposium on String Processing and Information Retrieval. SPIRE 2000*, 2000, pp. 39-48.

[36] Y. Liu, A. Wirawan, y B. Schmidt, «CUDASW++ 3.0: accelerating Smith-Waterman protein database search by coupling CPU and GPU SIMD instructions», *BMC Bioinformatics*, vol. 14, p. 117, 2013.

[37] I. Korf, M. Yandell, y J. Bedell, *BLAST*. O'Reilly, 2003.

[38] I. Kirmitzoglou y V. J. Promponas, «LCR-eXXXplorer: a web platform to search, visualize and share data for low complexity regions in protein sequences», *Bioinformatics*, vol. 31, n.º 13, pp. 2208-2210, jul. 2015.

[39] J. C. Wootton y S. Federhen, «Statistics of local complexity in amino acid sequences and sequence databases», *Comput. Chem.*, vol. 17, n.º 2, pp. 149-163, 1993.

[40] S. F. Altschul, «Amino acid substitution matrices from an information theoretic perspective», *J. Mol. Biol.*, vol. 219, n.º 3, pp. 555-565, 1991.

[41] S. Karlin y S. F. Altschul, «Methods for assessing the statistical

significance of molecular sequence features by using general scoring schemes.», *Proc. Natl. Acad. Sci.*, vol. 87, n.º 6, pp. 2264-2268, mar. 1990.

[42] S. Henikoff y J. G. Henikoff, «Amino acid substitution matrices from protein blocks.», *Proc. Natl. Acad. Sci.*, vol. 89, n.º 22, pp. 10915-10919, nov. 1992.

[43] T. A. Tatusova y T. L. Madden, «BLAST 2 Sequences, a new tool for comparing protein and nucleotide sequences», *FEMS Microbiol. Lett.*, vol. 174, n.º 2, pp. 247-250, may 1999.

[44] D. J. Lipman y W. R. Pearson, «Rapid and sensitive protein similarity searches», *Science*, vol. 227, n.º 4693, pp. 1435-1441, mar. 1985.

[45] J. W. Fickett, «Fast optimal alignment», *Nucleic Acids Res.*, vol. 12, n.º 1Part1, pp. 175-179, ene. 1984.

[46] H. Hermjakob *et al.*, «The HUPO PSI's Molecular Interaction format—a community standard for the representation of protein interaction data», *Nat. Biotechnol.*, vol. 22, n.º 2, pp. 177-183, feb. 2004.

[47] C. Pj, F. Cj, G. N, H. Ml, y R. Pm, «The Sanger FASTQ file format for sequences with quality scores, and the Solexa/Illumina FASTQ variants.», *Nucleic Acids Res.*, vol. 38, n.º 6, pp. 1767-1771, abr. 2010.

[48] H. Li *et al.*, «The Sequence Alignment/Map format and SAMtools», *Bioinformatics*, vol. 25, n.º 16, pp. 2078-2079, ago. 2009.

[49] C. Sander y R. Schneider, «Database of homology-derived protein structures and the structural meaning of sequence alignment», *Proteins Struct. Funct. Bioinforma.*, vol. 9, n.º 1, pp. 56-68, ene. 1991.

[50] G. B. Singh, «Multiple Sequence Alignment», en *Fundamentals of Bioinformatics and Computational Biology*, Springer International Publishing, 2015, pp. 143-158.

[51] S. F. Altschul *et al.*, «Gapped BLAST and PSI-BLAST: a new generation of protein database search programs», *Nucleic Acids Res.*, vol. 25, n.º 17, pp. 3389-3402, sep. 1997.

[52] A. A. Schäffer, Y. I. Wolf, C. P. Ponting, E. V. Koonin, L. Aravind, y S. F. Altschul, «IMPALA: matching a protein sequence against a collection of PSI-BLAST-constructed position-specific score matrices», *Bioinformatics*, vol. 15, n.º 12, pp. 1000-1011, dic. 1999.

[53] R. Sadreyev y N. Grishin, «COMPASS: A Tool for Comparison of Multiple Protein Alignments with Assessment of Statistical Significance», *J. Mol. Biol.*, vol. 326, n.º 1, pp. 317-336, feb. 2003.

[54] L. Rychlewski, W. Li, L. Jaroszewski, y A. Godzik, «Comparison of sequence profiles. Strategies for structural predictions using sequence information», *Protein Sci.*, vol. 9, n.º 2, pp. 232-241, ene. 2000.

[55] Y. Yang, E. Faraggi, H. Zhao, y Y. Zhou, «Improving protein fold recognition and template-based modeling by employing probabilistic-based matching between predicted one-dimensional structural properties of query and corresponding native properties of templates», *Bioinformatics*, vol. 27, n.º 15, pp. 2076-2082, ago. 2011.

[56] M. Margelevičius y Č. Venclovas, «Detection of distant evolutionary relationships between protein families using theory of sequence profile-profile comparison», *BMC Bioinformatics*, vol. 11, p. 89, 2010.

[57] R. D. Finn, J. Clements, y S. R. Eddy, «HMMER web server: interactive sequence similarity searching», *Nucleic Acids Res.*, vol. 39, n.º suppl_2, pp. W29-W37, jul. 2011.

[58] M. Remmert, A. Biegert, A. Hauser, y J. Söding, «HHblits: lightning-fast iterative protein sequence searching by HMM-HMM alignment», *Nat. Methods*, vol. 9, n.º 2, pp. 173-175, feb. 2012.

[59] B. Liu, X. Wang, Q. Chen, Q. Dong, y X. Lan, «Using Amino Acid Physicochemical Distance Transformation for Fast Protein Remote Homology Detection», *PLOS ONE*, vol. 7, n.º 9, p. e46633, sep. 2012.

[60] B. Liu *et al.*, «Using distances between Top-n-gram and residue pairs for protein remote homology detection», *BMC Bioinformatics*, vol. 15, n.º Suppl 2, p. S3, 2014.

[61] M. Franceschet, «PageRank: Standing on the Shoulders of Giants», *Commun ACM*, vol. 54, n.º 6, pp. 92–101, jun. 2011.

[62] I. Melvin, J. Weston, C. Leslie, y W. S. Noble, «Rankprop: a web server for protein remote homology detection», *Bioinformatics*, vol. 25, n.º 1, pp. 121-122, ene. 2009.

[63] Z. Qian, P. Bogdan, C. Y. Tsui, y R. Marculescu, «Performance evaluation of multicore systems: From traffic analysis to latency predictions (Embedded tutorial)», en *2013 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2013, pp. 82-84.

[64] M. Ilyas, Q. Javaid, y M. A. Shah, «Use of Symmetric Multiprocessor Architecture to achieve high performance computing», en *2016 22nd International Conference on Automation and Computing (ICAC)*, 2016, pp. 42-47.

[65] C. Lameter, «NUMA (Non-Uniform Memory Access): An Overview», *Queue*, vol. 11, n.º 7, p. 40:40–40:51, jul. 2013.

[66] S. Hauck y A. DeHon, *Reconfigurable Computing: The Theory and Practice of FPGA-Based Computation*. Morgan Kaufmann, 2010.

[67] D. Sanchez-Roman *et al.*, «FPGA acceleration using high-level languages of a Monte-Carlo method for pricing complex options», *J. Syst. Archit.*, vol. 59, n.º 3, pp. 135-143, mar. 2013.

[68] E. El-Araby, S. G. Merchant, y T. El-Ghazawi, «A Framework for Evaluating High-Level Design Methodologies for High-Performance Reconfigurable Computers», *IEEE Trans. Parallel Distrib. Syst.*, vol. 22, n.º 1, pp. 33-45, ene. 2011.

[69] E. El-Araby, S. G. Merchant, y T. El-Ghazawi, «Assessing Productivity of High-Level Design Methodologies for High-Performance Reconfigurable Computers», en *High-Performance Computing Using FPGAs*, W. Vanderbauwhede y K. Benkrid, Eds. Springer New York, 2013, pp. 719-745.

[70] «BLAST: Basic Local Alignment Search Tool». [En línea]. Disponible en: https://blast.ncbi.nlm.nih.gov/Blast.cgi. [Accedido: 12-feb-2017].

[71] C. Camacho *et al.*, «BLAST+: architecture and applications», *BMC Bioinformatics*, vol. 10, p. 421, 2009.

[72] H. Wang, B. C. Ooi, K.-L. Tan, T.-H. Ong, y L. Zhou, «BLAST++: BLASTing queries in batches», *Bioinformatics*, vol. 19, n.º 17, pp. 2323-2324, nov. 2003.

[73] A. E. Darling, L. Carey, y W. Feng, «The Design, Implementation, and Evaluation of mpiBLAST», en *In Proceedings of ClusterWorld 2003*, 2003.

[74] A. Mahram y M. C. Herbordt, «Fast and Accurate NCBI BLASTP: Acceleration with Multiphase FPGA-based Prefiltering», en *Proceedings of the 24th ACM International Conference on Supercomputing*, New York, NY, USA, 2010, pp. 73–82.

[75] L. Wienbrandt, D. Siebert, y M. Schimmler, «Improvement of BLASTp on the FPGA-Based High-Performance Computer RIVYERA», en *Bioinformatics Research and Applications*, 2012, pp. 275-286.

[76] J. Fang, A. L. Varbanescu, y H. Sips, «A Comprehensive Performance Comparison of CUDA and OpenCL», en *2011 International Conference on Parallel Processing*, 2011, pp. 216-225.

[77] K. Karimi, N. G. Dickson, y F. Hamze, «A Performance Comparison of CUDA and OpenCL», *ArXiv10052581 Phys.*, may 2010.

[78] S. R. Eddy, «Where did the BLOSUM62 alignment score matrix come from?», *Nat. Biotechnol.*, vol. 22, n.º 8, pp. 1035-1036, ago. 2004.

[79] J. L. Risler, M. O. Delorme, H. Delacroix, y A. Henaut, «Amino acid substitutions in structurally related proteins a pattern recognition approach», *J. Mol. Biol.*, vol. 204, n.º 4, pp. 1019-1029, dic. 1988.

[80] M. S. Johnson y J. P. Overington, «A Structural Basis for Sequence Comparisons», *J. Mol. Biol.*, vol. 233, n.º 4, pp. 716-738, oct. 1993.

[81] Z. Yang y A. D. Yoder, «Estimation of the Transition/Transversion Rate Bias and Species Sampling», *J. Mol. Evol.*, vol. 48, n.º 3, pp. 274-283, mar. 1999.

[82] L. B. Koski y G. B. Golding, «The Closest BLAST Hit Is Often Not the Nearest Neighbor», *J. Mol. Evol.*, vol. 52, n.º 6, pp. 540-542, jun. 2001.

[83] M. Burrows y D. J. Wheeler, «A block-sorting lossless data compression algorithm», 1994.

[84] W. J. Kent, «BLAT—The BLAST-Like Alignment Tool», *Genome Res.*, vol. 12, n.º 4, pp. 656-664, abr. 2002.

[85] G. Retamosa, L. de Pedro, I. González, y J. Tamames, «Prefiltering Model for Homology Detection Algorithms on GPU», *Evol. Bioinforma. Online*, vol. 12, pp. 313-322, dic. 2016.

[86] K. A. Brayton *et al.*, «Complete genome sequencing of Anaplasma marginale reveals that the surface is skewed to two superfamilies of outer membrane proteins», *Proc. Natl. Acad. Sci. U. S. A.*, vol. 102, n.º 3, pp. 844-849, ene. 2005.

[87] H. Karch, P. I. Tarr, y M. Bielaszewska, «Enterohaemorrhagic Escherichia coli in human medicine», *Int. J. Med. Microbiol.*, vol. 295, n.º 6–7, pp. 405-418, oct. 2005.

[88] «Nutritional Interactions in Insect-Microbial Symbioses: Aphids and Their Symbiotic Bacteria Buchnera», *Annu. Rev. Entomol.*, vol. 43, n.º 1, pp. 17-37, 1998.

[89] P. Cornelis, *Pseudomonas: Genomics and Molecular Biology*. Horizon Scientific Press, 2008.

[90] S. Hong y H. Kim, «An Integrated GPU Power and Performance Model», en *Proceedings of the 37th Annual International Symposium on Computer Architecture*, New York, NY, USA, 2010, pp. 280–289.

[91] Y. Zhao, H. Tang, y Y. Ye, «RAPSearch2: a fast and memory-efficient protein similarity search tool for next-generation sequencing data», *Bioinformatics*, vol. 28, n.º 1, pp. 125-126, ene. 2012.

[92] G. Retamosa de Ágreda y J. Aracil, «Performance evaluation of open-source software for traffic traces manipulation and analysis», 2012.