Evaluación de rendimiento y eficiencia energética de sistemas heterogéneos para bioinformática

Enzo Rucci

Facultad de Informática

Universidad Nacional de La Plata

Director y Codirector (UNLP): Ing. Armando De Giusti y Dr. Marcelo Naiouf

Director y Codirector (UCM): Dr. Carlos García Sanchez y Dr. Guillermo Botella Juan

Tesis presentada para obtener el grado de

Doctor en Ciencias Informáticas

Diciembre 2015

Resumen

El problema del consumo energético se presenta como uno de los mayores obstáculos para el diseño de sistemas que sean capaces de alcanzar la escala de los Exaflops. Por lo tanto, la comunidad científica está en la búsqueda de diferentes maneras de mejorar la eficiencia energética de los sistemas HPC. Una tendencia reciente para incrementar el poder computacional y al mismo tiempo limitar el consumo de potencia de estos sistemas consiste en incorporarles aceleradores y coprocesadores, como pueden ser las GPUs de NVIDIA y AMD o los coprocesadores Xeon Phi de Intel. Por otra parte, las FPGAs aparecen como una opción promisoria para HPC debido a su capacidad de cómputo creciente, su bajo consumo energético y al desarrollo de nuevas herramientas que facilitan su programación. Estos sistemas híbridos que emplean diferentes recursos de procesamiento se denominan sistemas heterogéneos y son capaces de obtener mejores cocientes FLOPS/Watt.

Entre las áreas que se ven afectadas por los problemas actuales de los sistemas HPC se encuentra la bioinformática, debido al crecimiento exponencial que ha experimentado la información biológica en los últimos años y a que cuenta con un número creciente de aplicaciones que requieren de cómputo de altas prestaciones para alcanzar tiempos de respuesta aceptables. Una de esas aplicaciones es el alineamiento de secuencias, la cual es considerada una operación fundamental en las ciencias biológicas debido a su utilización en la mayoría de sus especialidades. El alineamiento consiste en comparar dos o más secuencias biológicas y su propósito es detectar qué regiones comparten una historia evolutiva común. El algoritmo SW es un método popular para el alineamiento local de secuencias que ha sido utilizado como base para otros algoritmos posteriores y como patrón con el cual comparar otras técnicas de alineamiento. Sin embargo, debido a que su complejidad computacional es cuadrática, en la práctica se emplean diversas heurísticas que permiten reducir el tiempo de ejecución a costo de una pérdida en la sensibilidad de los resultados.

De manera de procesar el creciente volumen de información biológica con tiempos de respuesta aceptables, resulta necesario desarrollar nuevas herramientas computacionales que sean capaces de acelerar primitivas claves y algoritmos elementales eficientemente en términos de rendimiento y consumo energético. Por ese motivo, esta tesis se planteó como objetivo general *evaluar el rendimiento y la eficiencia energética de sistemas para cómputo de altas prestaciones al acelerar el alineamiento de secuencias biológicas mediante el método de Smith-Waterman.*

En primer lugar, se estudiaron las posibles formas de paralelizar el algoritmo SW y se describieron las implementaciones existentes sobre diferentes plataformas de procesamiento: CPU, GPU, FPGA y Xeon Phi. El análisis para cada dispositivo incluyó la evolución temporal de sus implementaciones así como también una descripción detallada de los aportes, las limitaciones, los resultados y la experimentación realizada en cada uno de los trabajos. Este análisis fue posible gracias al estudio de las diferentes arquitecturas y modelos de programación además del de los distintos algoritmos para alineamiento de secuencias biológicas

realizado previamente.

A continuación, se desarrollaron nuevas soluciones algorítmicas para sistemas heterogéneos. Como las GPUs son el acelerador dominante en la comunidad de HPC al día de hoy y existe vasta investigación científica sobre el uso de esta clase de aceleradores para el alineamiento de secuencias, se decidió priorizar el desarrollo de implementaciones para sistemas heterogéneos basados en Xeon Phi y basados en FPGA. Al inicio de esta tesis no existían implementaciones disponibles para el alineamiento de secuencias basadas en Xeon Phi. En sentido opuesto, sí existían antecedentes en la aplicación de FPGAs para el procesamiento de secuencias biológicas, aunque estas implementaciones fueron desarrolladas con HDLs tradicionales y en su mayoría poseen una o más limitaciones que restringen su uso en el mundo real.

El trabajo experimental se inició con los sistemas heterogéneos basados en Xeon Phi. Como punto de partida, se desarrollaron y optimizaron implementaciones para las CPUs y los Xeon Phi en forma individual antes de combinarlos en una implementación híbrida. Las implementaciones desarrolladas fueron compiladas en una única herramienta a la cual se la llamó SWIMM. A partir del análisis del estado del arte realizado inicialmente, se identificó a SWIPE como la herramienta más rápida para búsquedas de similitud en CPU. Mediante los experimentos realizados, se mostró que la versión SSE de SWIMM es equiparable con SWIPE mientras que la versión AVX2 logró superarlo ampliamente alcanzado diferencias de hasta 1.4×. Cabe destacar que, hasta donde llega el conocimiento del tesista, esta es la primera implementación SW para el conjunto de instrucciones AVX2. En cuanto a los Xeon Phi, al inicio de esta investigación no existían implementaciones disponibles para este coprocesador. Sin embargo, en el año 2014 aparecieron SWAPHI y XSW 2.0. Mientras que SWAPHI sólo explota el coprocesador, XSW 2.0 es capaz de aprovechar la potencia del *host* en simultáneo. A través de los experimentos realizados, se mostró que la versión KNC de SWIMM resulta competitiva con SWAPHI, siendo capaz de superarlo para secuencias medianas y largas. Complementariamente, la versión híbrida de SWIMM con distribución dinámica (SSE+KNC) superó notablemente a su alternativa XSW 2.0, logrando aceleraciones de hasta 3.9×.

El siguiente paso en el trabajo experimental consistió en el desarrollo de soluciones algorítmicas para sistemas heterogéneos basados en aceleradores FPGA. A diferencia de las implementaciones disponibles en la literatura, se decidió explorar los beneficios de utilizar una tecnología innovadora, como lo es OpenCL en el ámbito de las FPGAs, en lugar de HDLs tradicionales como VHDL o Verilog. En primer lugar, se exploró el rendimiento y el consumo de recursos de diferentes *kernels* de forma de encontrar la configuración más beneficiosa. Posteriormente, se desarrolló una versión híbrida empleando el código de SWIMM. Estas implementaciones también fueron compiladas en una única herramienta a la cual se la llamó OSWALD. Hasta donde llega el conocimiento del tesista, ésta es la primera implementación utilizando OpenCL sobre FPGAs para búsquedas de similitud. Además, en base al análisis del estado del arte realizado inicialmente, es una de las pocas implementaciones que es completamente funcional y general para esta clase de sistemas, además de facilitar la portabilidad por el empleo de OpenCL. Desafortunadamente, la ausencia del código fuente impide una comparación con otras implementaciones basadas en FPGA y aunque un análisis teórico es posible, los diferentes dispositivos, tecnologías y enfoques empleados no sólo complican una comparación directa sino que también dificultan hacerla en forma justa.

La última etapa del trabajo experimental se basó en relacionar el rendimiento alcanzado con el consumo de potencia de los sistemas empleados previamente. De acuerdo al análisis realizado, se puede afirmar que los sistemas basados únicamente en CPU son capaces de

obtener un buen balance entre rendimiento y consumo de potencia a partir de la explotación de multihilado e instrucciones vectoriales, destacándose aquellos que disponen de las extensiones AVX2. Por otra parte, la incorporación de aceleradores al cómputo del *host* demostró mejorar el rendimiento global del sistema en todos los casos, aunque la proporción de mejora varió de acuerdo al acelerador elegido. Desafortunadamente no ocurrió lo mismo en cuanto a la eficiencia energética. Los Xeon Phi no representan una buena opción para este tipo de aplicación. La ausencia de capacidades vectoriales para enteros de bajo rango es la causa principal del pobre rendimiento de este coprocesador, lo que provoca que el incremento en el consumo energético sea mucho mayor que la ganancia de rendimiento. En sentido opuesto, las GPUs sí pueden explotar este tipo de datos y es por ello que se puede aprovechar su gran poder computacional para acelerar los alineamientos. Aunque el aumento en el consumo de potencia es elevado, la mejora lograda por el uso de esta clase de aceleradores resulta superior, lo que se traduce en una mayor eficiencia energética. En el caso de las FPGAs, su capacidad de reconfiguración hace posible adaptar el hardware a los requerimientos del algoritmo SW. Si bien el rendimiento alcanzable de estos aceleradores puede ser inferior al correspondiente a las GPUs, su bajo consumo de potencia lleva a mayores tasas de eficiencia energética. A diferencia de otras evaluaciones de rendimiento y eficiencia energética en el contexto de SW, en esta tesis se han empleado secuencias de consulta y bases de datos representativas de la comunidad bioinformática, potentes arquitecturas orientadas a HPC y eficientes implementaciones que han demostrado ser las más rápidas de su clase. Por ese motivo se considera que la evaluación presentada puede ser de mayor utilidad en el mundo real.

De acuerdo a los resultados obtenidos y a las contribuciones realizadas, se espera que esta tesis aporte a una mayor adopción de SW por parte de la comunidad bioinformática y a un procesamiento más eficiente de los alineamientos de secuencias biológicas en términos de rendimiento y consumo energético.

Índice general

Índice de figuras

Índice de tablas

Algoritmos

Abreviaturas

ADN	Ácido desoxirribonucleico
ALM	Módulo de lógica adaptativa
AP	*Adaptive Profile*
APU	Unidad de Procesamiento Acelerada
AVX	*Advanced Vector eXtensions*
CPU	Unidad central de procesamiento
CU	*Compute Unit*
CUPS	*Cell Updates Per Second*
DSP	Procesador digital de señales
E/S	Entrada/Salida
FLOPS	Operaciones en punto flotante por segundo
FPGA	*Field Programmable Gate Array*
GCN	*Graphics Core Next*
GPC	Cluster de procesamiento gráfico
GPU	Unidad de procesamiento gráfico
HBM	Memoria de Alto Ancho de Banda
HDL	Lenguaje de descripción de hardware
HPC	Cómputo de alto rendimiento
ILP	Paralelismo a nivel de instrucciones
KNC	*Knights Corner*
NR	*Non-Redundant*
QP	*Query Profile*
SDK	Kit de desarrollo de software
SIMD	*Simple Instruction Multiple Data*
SM	*Streaming Multiprocessor*
SMC	*System Management Controller*
SMM	*Maxwell Streaming Multiprocessor*
SP	*Score Profile*
SSE	*Streaming SIMD Extensions*
SW	Smith-Waterman
TDP	Potencia de diseño térmico
VLIW	*Very Long Instruction Word*
VPU	Unidad de procesamiento vectorial

Capítulo 1

Introducción

En primer lugar, se presenta la motivación de esta tesis (Sección 1.1). Luego se enuncian los objetivos y la metodología a emplear (Sección 1.2), los alcances y limitaciones (Sección 1.3), las contribuciones (Sección 1.4) y las publicaciones derivadas de esta investigación (Sección 1.5). Por último, se describe la organización del documento (Sección 1.6).

1.1. Motivación

Tradicionalmente el objetivo principal de la comunidad de cómputo de altas prestaciones (HPC, por sus siglas en inglés) ha sido mejorar el rendimiento y ocasionalmente el cociente precio/rendimiento, donde rendimiento hace referencia a la velocidad de procesamiento. Un reflejo de este objetivo es el ranking TOP500 [10], que utiliza la métrica número de operaciones en punto flotante por segundo (FLOPS, por sus siglas en inglés) para listar las 500 supercomputadoras más potentes del mundo [11]. Al observar este ranking se puede apreciar la mejora en la velocidad de procesamiento de las supercomputadoras en las últimas dos décadas. En particular, mientras que la supercomputadora más potente en 1993 alcanzaba 59.7 Gflops, la correspondiente a 2015 es capaz de llegar a 33.9 Pflops, lo que representa una mejora de casi $6{\times}10^5$ veces.

La constante búsqueda por mejorar la velocidad de procesamiento llevó a las empresas fabricantes de hardware a desarrollar supercomputadoras que consumen gigantescas cantidades de energía eléctrica y que producen tanto calor que requieren de grandes instalaciones refrigeradas que aseguren un correcto funcionamiento. Como regla general, por cada megawatt (MW) de potencia consumida por una supercomputadora actual, se requieren otros 0.7 MW de refrigeración para disipar el calor generado. Este dato no es insignificante ya que cada MW de potencia cuesta aproximadamente US$ 1 millón por año [12]. De acuerdo con el último listado del TOP500[1], la supercomputadora más poderosa de la actualidad, Tianhe-2, consume 17.8 MW, lo que representa un costo mínimo aproximado de US$ 30 millones por año para funcionar.

En un mundo con recursos energéticos limitados y una constante demanda por mayor poder computacional, el problema del consumo energético se presenta como uno de los mayores obstáculos para el diseño de sistemas que sean capaces de alcanzar la escala de los Exaflops. Por lo tanto, la comunidad científica está en la búsqueda de diferentes maneras de mejorar la eficiencia energética de los sistemas HPC [13]. El surgimiento del ranking

[1] Noviembre de 2015

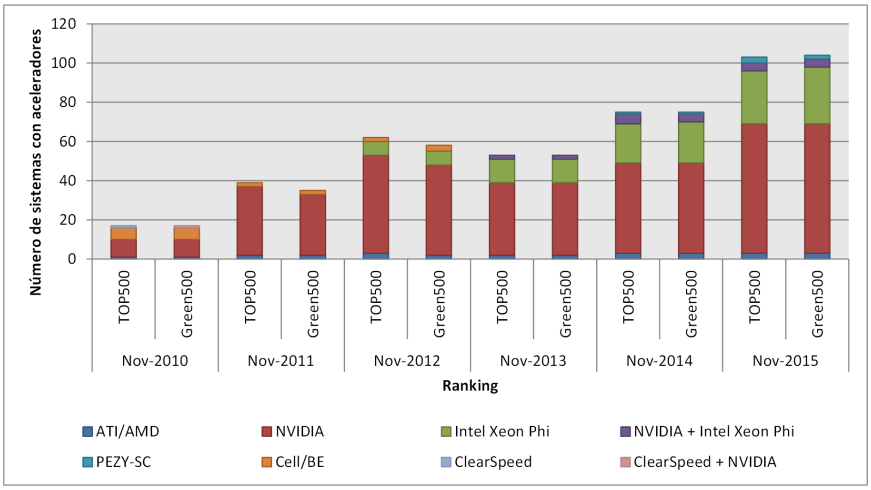

Figura 1.1: Número de sistemas con aceleradores de los rankings TOP500 y Green500 en sus ediciones de noviembre entre 2010 y 2015.

Green500 [14], que lista las 500 supercomputadoras más eficientes energéticamente (medida como FLOPS/Watts), es un reflejo de la importancia de esta búsqueda.

Una estrategia reciente para incrementar el poder computacional de los sistemas HPC consiste en incorporarles aceleradores y coprocesadores, como pueden ser la unidades de procesamiento gráfico (GPU, por sus siglas en inglés) de AMD y NVIDIA o los coprocesadores Xeon Phi de Intel. Más allá de que estos dispositivos no comparten las mismas características, la estrategia se basa en utilizarlos para las secciones de código que requieren cómputo intensivo y dejar las tareas *livianas* a la unidad central de procesamiento (CPU, por sus siglas en inglés), como la entrada/salida (E/S) y las comunicaciones. Estos sistemas híbridos que emplean diferentes recursos de procesamiento se denominan arquitecturas heterogéneas y son capaces de obtener picos de rendimiento muy superiores a los de las CPUs. En los últimos años, ésta estrategia se ha ido consolidando por una capacidad adicional de los aceleradores y coprocesadores: el incremento en el poder de cómputo se logra al mismo tiempo que se limita el consumo de potencia, lo que permite obtener mejores cocientes FLOPS/Watt. Una vez más el TOP500 sirve para dar evidencia a esta tendencia mientras que el Green500 la refuerza. La Figura 1.1 ilustra el número de supercomputadoras que emplean tecnología de aceleradores y coprocesadores del TOP500 y del Green500 en sus ediciones de noviembre entre 2010 y 2015. Como se puede apreciar, ambos rankings contaban con 17 sistemas con aceleradores en noviembre de 2010. Sin embargo, cinco años más tarde, este número se ha incrementado a 104 para el TOP500 y a 103 para el Green500.

Uno de los primeros aceleradores en ser incorporados a los sistemas de cómputo de altas prestaciones fueron las GPUs. En sus inicios estos dispositivos fueron pensados para el procesamiento gráfico y sus diseños estuvieron orientados a este tipo de aplicaciones. En la última década, se introdujeron algunas modificaciones a las arquitecturas de estos dispositivos y se desarrollaron librerías que permiten evitar el uso de primitivas de gráficos para programarlos. Estas modificaciones posibilitaron una amplia extensión de su uso. La clave del poder computacional de las GPUs se encuentra en su arquitectura: cientos o miles de núcleos pequeños y simples que ejecutan instrucciones en orden y operan en grupo como un procesador vectorial sumado a una memoria principal con alto ancho de banda la convierten en un arquitectura masivamente paralela capaz de lograr un pico de rendimiento muy por

encima al de las CPUs. Sin embargo, los algoritmos deben ser codificados de forma tal que reflejen su arquitectura para lograr alto rendimiento. En ese sentido, tanto la arquitectura de las GPUs como sus modelos de programación difieren sustancialmente de los correspondientes a las CPUs, que es donde la comunidad HPC se ha focalizado tradicionalmente. En consecuencia, el programador se ve obligado a aprender conocimientos específicos de las arquitecturas y de su programación, lo que representa una tarea ardua.

Una alternativa a las GPUs son los coprocesadores Xeon Phi, los cuales fueron desarrollados por Intel para aplicaciones que requieren de cómputo de altas prestaciones. Se diseñaron como un complemento de las CPUs aunque pueden operar en forma individual, lo que los diferencia de las placas gráficas [15]. Su arquitectura se basa en la de los procesadores Xeon pero con un mayor número de núcleos y de capacidades vectoriales más amplias. Al compartir los mismos modelos de programación que los procesadores Xeon, no resulta necesario aprender un nuevo lenguaje o herramienta para programarlos, con lo cual se reduce el tiempo requerido para desarrollar o portar una aplicación en este tipo de dispositivos. Por haber sido desarrollado recientemente, su utilidad y rendimiento son evaluados en la actualidad en diferentes áreas de aplicación, como pueden ser álgebra lineal [16], predicción del clima [17] o diferentes tipos de simulación [18].

Las *Field Programmable Gate Arrays* (FPGAs) son otro tipo de acelerador basado en circuitos integrados reconfigurables. Aunque tradicionalmente fueron utilizadas para el procesamiento digital de señales, su uso ha crecido en diferentes áreas [19] y probablemente se expanda aun más considerando que Intel compró recientemente Altera, una de las principales empresas fabricantes de FPGAs [20]. HPC es una de las áreas que se ha interesado en estos aceleradores debido principalmente a la evolución en su capacidad de cómputo y a su bajo consumo energético. Si bien tanto la frecuencia del reloj como el pico de rendimiento suelen ser más bajos que los correspondientes a las CPUs y a las GPUs, la capacidad de configurar el hardware para que se adapte al problema específico a resolver le da la posibilidad a las FPGAs de obtener mejores rendimientos, además de ser en general más eficientes desde el punto de vista energético [21]. La programación de estas placas se realiza a través de lenguajes de descripción de hardware (HDL, por sus siglas en inglés), como pueden ser Verilog o VHDL. Desafortunadamente, los HDLs son engorrosos, propensos a errores y requieren tener que mantener una explícita noción del tiempo, lo que implica una complejidad adicional [22]. En la actualidad, las principales empresas fabricantes de FPGAs (Altera y Xilinx) trabajan en el desarrollo de herramientas que permitan disminuir el esfuerzo de programación de estos dispositivos; en particular, a través de OpenCL, lo que resulta más familiar para los programadores HPC [23, 24].

Más allá del tipo de acelerador utilizado, la programación de sistemas heterogéneos representa un verdadero desafío. Para lograr aplicaciones de alto rendimiento, los programadores deben enfrentar una serie de dificultades como pueden ser: estudiar características específicas de cada arquitectura, contemplar aplicaciones con múltiples niveles de paralelismo y aplicar técnicas de programación y optimización particulares para cada una de ellos, lograr una distribución del trabajo y un balance de carga entre los diferentes dispositivos de procesamiento que permita obtener buen rendimiento y afrontar el surgimiento de nuevos lenguajes y modelos de programación junto a la ausencia de un estándar y de herramientas afianzadas para este tipo de sistemas.

Una de las áreas que se ve afectada por los problemas actuales de los sistemas HPC es la bioinformática, ya que cuenta con un número creciente de aplicaciones que requieren de cómputo de altas prestaciones para alcanzar tiempos de respuesta aceptables. Esta situación se debe principalmente al crecimiento exponencial que ha experimentado la información

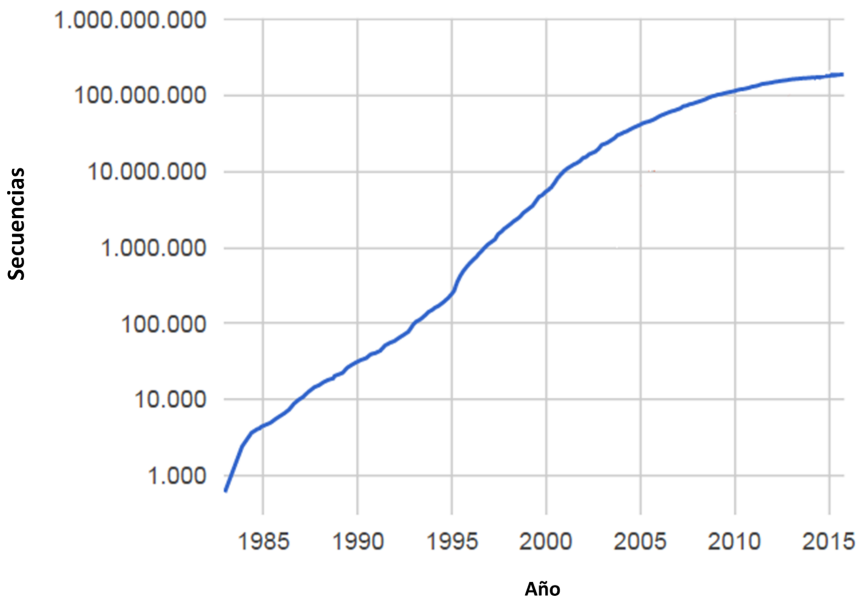

Figura 1.2: Crecimiento de la base de datos GenBank entre 1985 y 2015 (extraído de [1])

biológica en los últimos años, como se puede apreciar en la Figura 1.2, la cual muestra el crecimiento de la base de datos genómicos GenBank del Centro Nacional Americano para la Información Biotecnológica (NCBI, por sus siglas en inglés) [25] desde 1985 a 2015.

El origen de la bioinformática se relaciona con el progreso de las tecnologías informáticas junto al desarrollo de técnicas de secuenciación de ácido desoxirribonucleico (ADN) cada vez más rápidas. En la actualidad, la bioinformática es un área de investigación interdisciplinaria que actúa como puente entre diferentes campos de las ciencias, como la biología, la bioquímica, la informática, la matemática, la física y la estadística. Su objetivo final consiste en descubrir la información relevante que se encuentra oculta dentro de enormes volúmenes de datos y obtener una visión más clara de la biología de los organismos [26].

Una operación fundamental en cualquier investigación biológica es el alineamiento de secuencias, la cual consiste en comparar dos o más secuencias biológicas, como pueden ser las de ADN o las de proteínas. El propósito de esta operación es detectar qué regiones dentro de una secuencia comparten una historia evolutiva común. El alineamiento de secuencias resulta esencial en el análisis filogenético [27], el perfilado de enfermedades genéticas [28], la identificación y cuantificación de regiones conservadas o unidades funcionales [29, 30], y el perfilado y predicción de secuencias ancestrales [31]. También resulta importante en el desarrollo de nuevas drogas y en la investigación forense criminal, y es por ello que actualmente la comunidad científica realiza un gran esfuerzo en este campo de la bioinformática [32].

Los métodos para el alineamiento de secuencias de a pares se pueden clasificar como globales o locales, donde *de a pares* significa considerar sólo dos secuencias por alineamiento. Los alineamientos globales calculan la similitud considerando toda la longitud de las secuencias y son útiles cuando las secuencias son muy parecidas entre sí. El algoritmo de Needleman-Wunsch [33] es capaz de computar el alineamiento global óptimo entre dos secuencias. A diferencia de los alineamientos globales, los alineamientos locales consideran la similitud entre pequeñas regiones de las dos secuencias. Esto permite exponer de mejor manera las semejanzas entre secuencias desiguales y, a su vez, conducir a resultados con más

sentido biológico [34]. El algoritmo de Smith-Waterman (SW) [35] es un método popular para el alineamiento local de secuencias. Este método ha sido utilizado como base para otros algoritmos posteriores y como patrón con el cual comparar otras técnicas de alineamiento. Sin embargo, su complejidad computacional es cuadrática, lo que lo hace inviable para procesar grandes conjuntos de datos biológicos. En la actualidad, se emplean diversas heurísticas que permiten reducir el tiempo de ejecución a costo de una pérdida en la sensibilidad de los resultados. Por fortuna, el proceso de alineamiento exhibe cierto paralelismo inherente que puede ser explotado para reducir el costo computacional de SW sin perder precisión.

De manera de procesar el creciente volumen de información biológica con tiempos de respuesta aceptables, es necesario desarrollar nuevas herramientas computacionales que sean capaces de acelerar primitivas claves y algoritmos elementales, especialmente aquellos que son costosos computacionalmente como es el caso del algoritmo SW. En ese sentido, el empleo de paralelismo se vuelve fundamental y para ello resulta indispensable contar con una evaluación integral de los diferentes sistemas HPC que considere el rendimiento alcanzable y el consumo energético asociado.

1.2. Objetivos y metodología

El objetivo general de esta tesis consiste en evaluar el rendimiento y la eficiencia energética de sistemas para cómputo de altas prestaciones al acelerar el alineamiento de secuencias biológicas mediante el algoritmo SW. Los objetivos específicos son los siguientes:

- Describir y comparar las propuestas científicas para el alineamiento de secuencias biológicas utilizando sistemas para cómputo de altas prestaciones con el objeto de caracterizarlas.
- Identificar los factores que inciden en el rendimiento y la eficiencia energética de los sistemas para cómputo de altas prestaciones al acelerar el alineamiento de secuencias.
- Diseñar y desarrollar soluciones algorítmicas para aquellos sistemas de cómputo de altas prestaciones que no tengan implementaciones disponibles o, en caso contrario, que superen a las existentes en cuanto a aceleración.
- Medir y analizar el rendimiento y la eficiencia energética de las herramientas seleccionadas y de las propuestas desarrolladas.

Esta investigación doctoral se plantea como un estudio proyectivo que pretende dar solución a la necesidad de contar con implementaciones aceleradas del alineamiento de secuencias mediante el algoritmo SW en sistemas de cómputo de altas prestaciones junto a evaluaciones de rendimiento que además consideren la eficiencia energética. Para poder cumplir con los diferentes objetivos se realizarán las siguientes actividades:

- Se llevará a cabo un relevamiento de la bibliografía existente en la temática a partir de publicaciones científicas en bases de datos especializadas.
- Se seleccionarán secuencias de consulta y bases de datos biológicas que sean representativas de las utilizadas por la comunidad científica para ser usadas como *benchmarks*.
- Se instalarán las herramientas disponibles para acelerar el alineamiento de secuencias biológicas en uno o más entornos para cómputo de altas prestaciones. En caso de

existir más de una, se evaluarán las mismas de forma de poder seleccionar aquellas más representativas teniendo en cuenta sus características distintivas y la aceleración lograda.

- Se diseñarán soluciones algorítmicas al problema del alineamiento de secuencias biológicas para aquellos sistemas de cómputo de altas prestaciones que no tengan implementaciones disponibles o, en caso de que sí las tengan, que sean capaces de superar a las mismas en cuanto a aceleración.
- Se medirá el rendimiento y la eficiencia energética de las herramientas existentes seleccionadas y de las propuestas propias desarrolladas.
- Se evaluarán los resultados obtenidos y se realizará un análisis integral de los mismos.

1.3. Alcances y limitaciones

Con respecto al caso de estudio seleccionado, el alineamiento de secuencias biológicas contempla tanto secuencias de ADN como de proteínas. Se decidió focalizar en el alineamiento de proteínas en lugar del de ADN por dos razones. En primer lugar, alinear secuencias de proteínas en lugar de secuencias de ADN resulta ventajoso desde el punto de vista biológico en la mayoría de los casos, ya que se reduce el nivel de redundancia de los datos y resulta más fácil lograr significación estadística, entre otras ventajas [36, 37]. En segundo lugar, el alineamiento de secuencias de proteínas es más complejo que el correspondiente a ADN debido a un alfabeto más grande y al uso de matrices de sustitución, por lo que resolver el alineamiento de secuencias de ADN resulta relativamente simple una vez resuelto el primero.

En relación a la explotación de sistemas heterogéneos, las GPUs son el acelerador dominante en la comunidad de HPC al día de hoy por su alto rendimiento y bajo costo de adquisición y existe vasta investigación científica sobre el uso de esta clase de acelerador para el alineamiento de secuencias como se describe en la Sección 2.3. Por ese motivo se decidió priorizar el desarrollo de nuevas soluciones algorítmicas para sistemas heterogéneos basados en coprocesadores Xeon Phi y basados en FPGAs. Los coprocesadores Xeon Phi fueron introducidos recientemente y, al inicio de esta tesis, no existían implementaciones disponibles para el alineamiento de secuencias. En cuanto a las FPGAs, el estudio de su aplicación para el procesamiento de secuencias biológicas sí cuenta con antecedentes aunque estas implementaciones fueron desarrolladas con HDLs tradicionales y en su mayoría poseen una o más limitaciones que restringen su uso en el mundo real (Sección 2.3.2.3). El reciente desarrollo de herramientas basadas en OpenCL por parte de las principales empresas fabricantes de FPGAs, que disminuyen su costo de programación y que favorecen la portabilidad de sus programas, representa uno de los motivos del estudio de esta clase de acelerador para el alineamiento de secuencias. Por último, también existen otras clases de aceleradores, como los procesadores digitales de señales (DSP, por sus siglas en inglés) de Texas Instruments o las arquitecturas basadas en procesadores ARM. Sin embargo, su uso está mucho menos extendido en comparación a los anteriores y es por eso que no son considerados en este trabajo.

1.4. Contribuciones

Las principales contribuciones de esta tesis son las siguientes:

- Una descripción global del estado del arte de la implementación de algoritmos para alineamientos locales de secuencias biológicas en sistemas de cómputo de altas prestaciones. Esta contribución puede aportar a futuros estudios en la temática.

- Un conjunto de técnicas de programación y optimización para algoritmos paralelos que computen alineamientos de secuencias mediante el método SW. Estas técnicas pueden ser reutilizadas en implementaciones para las próximas generaciones de los procesadores, aceleradores y coprocesadores. Incluso también podrían ser usadas para el desarrollo de implementaciones a problemas con características similares teniendo en cuenta que el algoritmo SW sigue el paradigma de programación dinámica, el cual representa uno de los 13 *Dwarfs* de la programación paralela [38].

- Una herramienta para realizar búsquedas de similitud en bases de datos biológicas en sistemas heterogéneos basados en coprocesadores Xeon Phi llamada SWIMM. Esta herramienta trabaja con bases de datos biológicas reales y permite configurar los parámetros de ejecución. Además, admite tres modos de ejecución: (1) cómputo en el *host*, (2) cómputo en los coprocesadores y (3) cómputo concurrente en el *host* y los coprocesadores. Para ello, se vale de multihilado y de las instrucciones vectoriales SSE y AVX2 en el *host* y de las KNC en los coprocesadores. Se ha mostrado que el rendimiento con los diferentes modos de ejecución resulta competitivo o incluso superior al de otras opciones disponibles para la comunidad científica, como lo son SWIPE [4], SWAPHI [39] o XSW [40]. Por otro lado, hasta donde sabe el autor, la implementación con AVX2 es la primera de su clase. Para beneficio de la comunidad científica, SWIMM se encuentra disponible en un repositorio web público [2].

- Una herramienta para realizar búsquedas de similitud en bases de datos biológicas en sistemas heterogéneos basados en FPGAs llamada OSWALD. Hasta donde llega el conocimiento del tesista, ésta la primera implementación utilizando OpenCL sobre FPGAs para búsquedas de similitud. OSWALD ofrece dos modos de ejecución: (1) cómputo en las FPGAs y (2) cómputo concurrente en el *host* y las FPGAs. Además, es una de las pocas implementaciones que es completamente funcional y general para esta clase de sistemas, además de ser portable por el empleo de OpenCL. Al igual que la herramienta anterior, OSWALD procesa bases de datos biológicas reales, permite la configuración de los parámetros de ejecución y se encuentra disponible en un repositorio web público para beneficio de la comunidad científica [3].

- Una evaluación integral de la utilización de sistemas heterogéneos para computar alineamientos de secuencias biológicas considerando no sólo la aceleración lograda sino también la eficiencia energética. Esta evaluación permite identificar cuál tipo de sistema es capaz de lograr la mayor aceleración, cuál es el que consume menos potencia y cuál es el más eficiente energéticamente. Siendo el alineamiento de secuencias una operación fundamental, representativa y muy utilizada en la bioinformática, esta evaluación le puede ser útil a cualquier centro de investigación y desarrollo de esta área al momento de seleccionar un sistema HPC para sus aplicaciones de acuerdo con las prioridades que posea.

[2] SWIMM se encuentra disponible *online* en https://github.com/enzorucci/SWIMM
[3] OSWALD se encuentra disponible *online* en https://github.com/enzorucci/OSWALD

1.5. Publicaciones

Esta tesis doctoral está avalada por las siguientes publicaciones ordenadas según su fecha de publicación:

- "Smith-Waterman Algorithm on Heterogeneous Computing: A Case of Study". Enzo Rucci, Armando De Giusti, Marcelo Naiouf, Carlos García Sanchez, Guillermo Botella Juan, Manuel Prieto-Matías. *Proceedings of 2014 IEEE International Conference on Cluster Computing (CLUSTER)*. Septiembre de 2014. Madrid, España. ISBN: 978-1-4799-5547-3. Págs. 323-330.

- "An Energy-Aware Performance Analysis of SWIMM: Smith-Waterman Implementation on Intel's Multicore and Manycore". Enzo Rucci, Carlos García Sanchez, Guillermo Botella Juan, Armando De Giusti, Marcelo Naiouf, Manuel Prieto-Matías. *Concurrency and Computation: Practice and Experience*. ISSN: 1532-0626, 1532-0634 (online). Vol. 27. Págs. 5517-5537. Julio de 2015. DOI:10.1002/cpe.3598.

- "Smith-Waterman Protein Search with OpenCL on FPGA". Enzo Rucci, Armando De Giusti, Marcelo Naiouf, Carlos García Sanchez, Guillermo Botella Juan, Manuel Prieto-Matías. *Proceedings of 2015 IEEE Symposyum on Parallel and Distributed Processing with Applications (ISPA)*. 20 al 22 de Agosto de 2015. Helsinki, Finlandia. ISBN: 978-1-4673-7952-6. Págs. 208-213. DOI: 10.1109/Trustcom.2015.634.

- "OSWALD: OpenCL Smith-Waterman on Altera FPGA for Large Protein Databases". Enzo Rucci, Carlos García Sanchez, Guillermo Botella Juan, Armando De Giusti, Marcelo Naiouf, Manuel Prieto-Matías. *International Journal of High-Performance Computing Applicactions*. ISSN: 1094-3420, 1741-2846 (online). *En prensa.*

- "State-of-the-art in Smith-Waterman Protein Database Search on HPC Platforms". Enzo Rucci, Armando De Giusti, Marcelo Naiouf, Carlos García Sanchez, Guillermo Botella Juan, Manuel Prieto-Matías. *Big Data Analytics in Genomics*. Capítulo 6. Editor: Ka-Chun Wong. Springer (New York), 2017. *En prensa.*

A continuación se citan publicaciones previas del tesista relacionadas con la temática de la tesis:

- "DNA Sequence Alignment: hybrid parallel programming on multicore cluster", Enzo Rucci, Armando De Giusti, Franco Chichizola, Marcelo Naiouf, Laura De Giusti, *Proceedings of the International Conference on Computers, Digital Communications and Computing (ICDCCC '11)*, Vol. 1, Nikos Mastorakis, Valeri Mladenov, Badea Lepadatescu, Hamid Reza Karimi, Costas G. Helmis (Editors), WSEAS Press, September 15-17, 2011, Barcelona, Spain, ISBN: 978-1-61804-030-5, pp. 183-190.

- "Parallel Smith-Waterman Algorithm for DNA Sequences Comparison on Different Cluster Architectures", Enzo Rucci, Armando E. De Giusti, Franco Chichizola, *Proceedings of the 2011 International Conference on Parallel and Distributed Processing Techniques and Applications*, Vol. 1, Hamid R. Arabnia (Editor), Minoru Ito, Kazuki Joe, Hiroaki Nishikawa, Hiroshi Ishii, Fernando G. Tinetti, Ashu M. G. Solo, George A. Gravvanis (Associate Editors), CSREA Press, July 18-21, 2011, Las Vegas Nevada, USA, ISBN: 1-60132-193-7, pp. 666-672.

- "Parallel pipelines for DNA sequence alignment on a cluster of multicores. A comparison of communication models.". Enzo Rucci, Franco Chichizola, Marcelo Naiouf, Laura De Giusti, Armando De Giusti. *Journal of Communication and Computer (JCC)*. David Publishing Company, California (EEUU) 2012, vol. 9, núm. 12, pp. 1364-1371. ISSN: 1548-7709.

- "Power Characterisation of Shared-Memory HPC Systems". Javier Balladini, Enzo Rucci, Armando De Giusti, Marcelo Naiouf, Remo Suppi, Dolores Rexachs, Emilio Luque. *Computer Science & Technology Series – XVIII Argentine Congress of Computer Science Selected Papers*. Editores: Armando De Giusti, Guillermo Simari, Patricia Pesado. ISBN 978-987-1985-20-3. Págs. 53-65. Editorial de la Universidad de La Plata (edulp). La Plata (Argentina). 2013.

1.6. Organización de la tesis

A continuación se detalla la organización del resto del documento:

- En el Capítulo 2 se estudia la fuerte relación entre el alineamiento de secuencias biológicas y el HPC. En primer lugar, se explica el origen de la bioinformática y la importancia del alineamiento de secuencias en las ciencias biológicas. También se presentan y se analizan comparativamente los diferentes algoritmos para alineamiento de secuencias biológicas seguido de una descripción de las bases de datos biológicas más utilizadas por la comunidad científica. A continuación, se reseña la evolución de los procesadores y se caracterizan las arquitecturas heterogéneas actuales en la comunidad HPC, además de presentar los diferentes modelos y herramientas para programación paralela más populares. Por último, se estudian las dependencias de datos del algoritmo SW y las formas posibles de paralelizarlo seguido de una descripción del estado del arte de las implementaciones SW sobre diferentes plataformas de procesamiento paralelo: CPU, GPU, FPGA y Xeon Phi.

- En el Capítulo 3 se introduce la herramienta SWIMM para acelerar el algoritmo SW sobre sistemas heterogéneos basados en coprocesadores Xeon Phi. En primer lugar, se describen las técnicas de programación y optimización empleadas en las diferentes implementaciones. Posteriormente, se detallan los datos usados y los experimentos realizados para evaluar el rendimiento de las distintas implementaciones en forma individual y en forma comparativa con otras soluciones SW.

- En el Capítulo 4 se presenta la herramienta OSWALD para acelerar el algoritmo SW sobre sistemas heterogéneos basados en aceleradores FPGA. Al igual que en el capítulo anterior, se describen al inicio las técnicas de programación y optimización empleadas en las diferentes implementaciones desarrolladas. A continuación, se detallan los datos usados y los experimentos realizados para evaluar el rendimiento de las distintas propuestas en forma individual y en forma comparativa con otras soluciones SW.

- En el Capítulo 5 se detalla la metodología de medición de consumo de potencia empleada en cada arquitectura particular utilizada junto a la serie de experimentos realizados. Posteriormente, se evalúa la eficiencia energética de las diferentes propuestas desarrolladas y de otras implementaciones de referencia seleccionadas.

- En el Capítulo 6 se resume el trabajo realizado en esta investigación y se presentan las conclusiones. Finalmente se mencionan algunas de las posibles líneas de trabajo futuro que se pueden derivar de esta tesis.

Capítulo 2

Alineamiento de secuencias biológicas de alto rendimiento

El alineamiento de secuencias es una operación fundamental en cualquier investigación biológica. El costo computacional de los algoritmos junto al crecimiento exponencial de los datos biológicos hacen necesario el empleo de HPC para alcanzar tiempos de respuesta aceptables. Es por ello que este capítulo se centra en la fuerte relación entre el alineamiento de secuencias biológicas y el HPC. En la Sección 2.1 se explica la importancia del alineamiento de secuencias en las ciencias biológicas y se presentan y se analizan comparativamente los diferentes algoritmos seguido de una descripción de las bases de datos biológicas más utilizadas por la comunidad científica. A continuación, en la Sección 2.2 se reseña la evolución de los procesadores y se caracterizan las arquitecturas heterogéneas actuales en la comunidad HPC, además de presentar los diferentes modelos y herramientas para programación paralela más populares. Luego, en la Sección 2.3 se estudian las dependencias de datos del algoritmo SW y las formas posibles de paralelizarlo seguido de una descripción del estado del arte de las implementaciones SW sobre diferentes plataformas de procesamiento paralelo: CPU, GPU, FPGA y Xeon Phi. Por último, en la Sección 2.4 se presenta un resumen del capítulo.

2.1. Alineamiento de secuencias biológicas

En esta sección se explica el origen de la bioinformática y la importancia del alineamiento de secuencias en las ciencias biológicas (Sección 2.1.1). Luego, se presentan y se analizan comparativamente los diferentes algoritmos para alineamiento de secuencias biológicas (Secciones 2.1.2, 2.1.4 y 2.1.5) seguido de una descripción de las bases de datos biológicas más utilizadas por la comunidad científica (Sección 2.1.6).

2.1.1. Bioinformática y alineamiento de secuencias biológicas

Durante las últimas tres décadas, la biología molecular se ha visto revolucionada debido no sólo al desarrollo de técnicas de secuenciación de ADN cada vez más rápidas sino también al progreso de las tecnologías informáticas, las cuales permiten almacenar, manipular y analizar enormes cantidades de información en forma cada vez más eficiente. El término *bioinformática* se comenzó a utilizar a mediados de los años 80 para referirse a aquellas aplicaciones de computadora que resuelven problemas biológicos [26].

En la actualidad, la bioinformática es un área de investigación interdisciplinaria que

actúa como puente entre diferentes campos de las ciencias, como la biología, la bioquímica, la informática, la matemática, la física y la estadística. Su objetivo final consiste en descubrir la información importante que se encuentra oculta dentro de enormes volúmenes de datos y obtener una visión más clara de la biología de los organismos [41].

Una operación fundamental en cualquier investigación biológica es el alineamiento de secuencias, la cual consiste en comparar dos o más secuencias biológicas. El propósito de esta operación es determinar las regiones de similitud entre las secuencias para identificar las relaciones funcionales, estructurales y evolutivas que existen entre ellas. El alineamiento de secuencias resulta esencial en el análisis filogenético [27], el perfilado de enfermedades genéticas [28], la identificación y cuantificación de regiones conservadas o unidades funcionales [29, 30], y el perfilado y predicción de secuencias ancestrales [31]. También resulta importante en el desarrollo de nuevas drogas y en la investigación forense criminal, y es por ello que actualmente la comunidad científica realiza un gran esfuerzo en este campo de la bioinformática [32].

Cuando se alinean dos secuencias, se asume que ambas son *homólogas*. Las secuencias homólogas comparten un ancestro común y sus diferencias relativas son producto de mutaciones sufridas a lo largo del tiempo. Estas mutaciones se pueden manifestar de varias formas: *sustituciones* donde un símbolo es reemplazado por otro, *inserciones* donde un nuevo símbolo en insertado en una secuencia y *eliminaciones* donde un símbolo es eliminado de una secuencia. El alineamiento de secuencias permite determinar el grado de homología. Para ello, las secuencias son emparejadas de forma tal que el grado de similitud es maximizado [42].

2.1.2. Algoritmos para el alineamiento de secuencias biológicas

Las secuencias biológicas se representan simbólicamente mediante una sucesión de letras mayúsculas que representan la estructura real o hipotética de una molécula o cadena de ADN o de proteínas. El alfabeto de las secuencias de ADN se compone de 4 letras relativas a adenina (`A`), citosina (`C`), guanina (`G`) y timina (`T`). En el caso de las secuencias de proteínas, el alfabeto consiste de 23 letras, las cuales se corresponden con los aminoácidos.

Los algoritmos de alineamiento operan directamente sobre las letras de las secuencias. Como se mencionó anteriormente, la idea se basa en emparejar las secuencias de forma tal que el grado de similitud es maximizado. Para las secuencias de ADN, esto significa hacer coincidir los nucleótidos iguales. En el caso de las secuencias de proteínas, los aminoácidos son emparejados si son idénticos o si uno deriva de otro a través de sustituciones probables de ocurrir. Las mutaciones por sustitución son consideradas al emparejar las secuencias directamente. Sin embargo, para manejar las mutaciones por inserción y por eliminación, resulta necesario introducir la noción de *gaps*. Los *gaps* se denotan mediante el símbolo '-' y son insertados en las secuencias para aumentar el número de coincidencias [42].

En la Figura 2.1 se ilustra un ejemplo de alineamiento entre dos secuencias de ADN con y sin uso de *gaps* (extraído de [34]).

```
T A C C A G T       T A C C A G T - -
C C C G T A A       C - C C - G T A A
   sin gaps              con gaps
```

Figura 2.1: Alineamiento simple entre dos secuencias de ADN

Se puede notar en la figura anterior que el alineamiento con *gaps* expone de mejor manera

las similitudes entre ambas secuencias. Sin embargo, éste no es el único alineamiento posible. Otra opción podría ser la que se muestra en la Figura 2.2.

```
T  A  C  C  A  G  T  -  -
-  -  C  C  C  G  T  A  A
```

Figura 2.2: Alineamiento simple entre dos secuencias de ADN (segunda opción)

Como múltiples alineamientos son posibles entre dos secuencias, aún con casos sencillos como el ilustrado, se requiere de alguna técnica de clasificación que permita determinar objetivamente el mejor alineamiento para cualquier par de secuencias. Una manera sencilla de lograrlo consiste en asignar puntajes a las parejas del alineamiento. Un esquema simple podría ser +1 para las coincidencias, -1 para las no coincidencias y -2 para los *gaps*. A este tipo de esquema se lo llama de *penalización de gap lineal*. Un esquema más avanzado es aquel que emplea una *penalización de gap por afinidad*, donde se distinguen mediante puntajes diferentes la inserción de un nuevo *gap* de la continuación de uno ya existente. Este segundo esquema otorga en general una mayor penalización al inicio de un nuevo *gap* y suele ser el más utilizado debido a que tiene mayor sentido biológico [43]. Usando el esquema de puntuación lineal, el primer alineamiento con *gaps* obtiene un puntaje de (-1-2+1+1-2+1+1-2-2)=-5 mientras que el segundo alineamiento también (-2-2+1+1-1+1+1-2-2)=-5. En este caso, ambos alineamientos con *gaps* son igual de buenos. Sin embargo, esto no significa automáticamente que ambos tienen la misma relevancia biológica. Para determinar cuan relevante es un puntaje de alineamiento, se pueden utilizar métodos probabilísticos: la idea consiste en evaluar si la probabilidad de un alineamiento que logra el puntaje en cuestión es lo suficientemente pequeña. Por otra parte, resulta importante notar que de usar un esquema de penalización de *gaps* por afinidad, el segundo alineamiento hubiera obtenido el mejor puntaje ya que tiene dos *gaps* en lugar de tres [42].

El esquema descrito en el párrafo anterior resulta útil tanto para secuencias de proteínas como de ADN. En el caso de las secuencias de proteínas, se suelen utilizar las familias de matrices PAM y BLOSUM, las cuales asignan puntajes a las sustituciones de aminoácidos de acuerdo a su similitud evolutiva. La Figura 2.3 muestra una de las matrices BLOSUM (BLOSUM62). El caso de las secuencias de ADN es más simple ya que se suele utilizar un valor fijo para las coincidencias y otro para las no coincidencias.

2.1.2.1. Identidad y similitud

Realizar un alineamiento de secuencias significa generar un emparejamiento entre dos secuencias según un modelo matemático. El modelo describe, en términos generales, el concepto de alineamiento de dos secuencias y sus parámetros: penalizaciones por *gaps*, impacto de las diferentes longitudes de las secuencias, efecto de la complejidad del alfabeto, entre otros. Una elección adecuada de parámetros minimizará el número de *gaps*, mientras que su relajación permitirá, teóricamente, el alineamiento de cualquier par de secuencias arbitrarias. El hecho de realizar un alineamiento de dos secuencias no se debe tomar como prueba, por sí mismo, de que exista una relación entre ellas [26].

	A	R	N	D	C	Q	E	G	H	I	L	K	M	F	P	S	T	W	Y	V	B	Z	X	*
A	4	-1	-2	-2	0	-1	-1	0	-2	-1	-1	-1	-1	-2	-1	1	0	-3	-2	0	-2	-1	0	-4
R	-1	5	0	-2	-3	1	0	-2	0	-3	-2	2	-1	-3	-2	-1	-1	-3	-2	-3	-1	0	-1	-4
N	-2	0	6	1	-3	0	0	0	1	-3	-3	0	-2	-3	-2	1	0	-4	-2	-3	3	0	-1	-4
D	-2	-2	1	6	-3	0	2	-1	-1	-3	-4	-1	-3	-3	-1	0	-1	-4	-3	-3	4	1	-1	-4
C	0	-3	-3	-3	9	-3	-4	-3	-3	-1	-1	-3	-1	-2	-3	-1	-1	-2	-2	-1	-3	-3	-2	-4
Q	-1	1	0	0	-3	5	2	-2	0	-3	-2	1	0	-3	-1	0	-1	-2	-1	-2	0	3	-1	-4
E	-1	0	0	2	-4	2	5	-2	0	-3	-3	1	-2	-3	-1	0	-1	-3	-2	-2	1	4	-1	-4
G	0	-2	0	-1	-3	-2	-2	6	-2	-4	-4	-2	-3	-3	-2	0	-2	-2	-3	-3	-1	-2	-1	-4
H	-2	0	1	-1	-3	0	0	-2	8	-3	-3	-1	-2	-1	-2	-1	-2	-2	2	-3	0	0	-1	-4
I	-1	-3	-3	-3	-1	-3	-3	-4	-3	4	2	-3	1	0	-3	-2	-1	-3	-1	3	-3	-3	-1	-4
L	-1	-2	-3	-4	-1	-2	-3	-4	-3	2	4	-2	2	0	-3	-2	-1	-2	-1	1	-4	-3	-1	-4
K	-1	2	0	-1	-3	1	1	-2	-1	-3	-2	5	-1	-3	-1	0	-1	-3	-2	-2	0	1	-1	-4
M	-1	-1	-2	-3	-1	0	-2	-3	-2	1	2	-1	5	0	-2	-1	-1	-1	-1	1	-3	-1	-1	-4
F	-2	-3	-3	-3	-2	-3	-3	-3	-1	0	0	-3	0	6	-4	-2	-2	1	3	-1	-3	-3	-1	-4
P	-1	-2	-2	-1	-3	-1	-1	-2	-2	-3	-3	-1	-2	-4	7	-1	-1	-4	-3	-2	-2	-1	-2	-4
S	1	-1	1	0	-1	0	0	0	-1	-2	-2	0	-1	-2	-1	4	1	-3	-2	-2	0	0	0	-4
T	0	-1	0	-1	-1	-1	-1	-2	-2	-1	-1	-1	-1	-2	-1	1	5	-2	-2	0	-1	-1	0	-4
W	-3	-3	-4	-4	-2	-2	-3	-2	-2	-3	-2	-3	-1	1	-4	-3	-2	11	2	-3	-4	-3	-2	-4
Y	-2	-2	-2	-3	-2	-1	-2	-3	2	-1	-1	-2	-1	3	-3	-2	-2	2	7	-1	-3	-2	-1	-4
V	0	-3	-3	-3	-1	-2	-2	-3	-3	3	1	-2	1	-1	-2	-2	0	-3	-1	4	-3	-2	-1	-4
B	-2	-1	3	4	-3	0	1	-1	0	-3	-4	0	-3	-3	-2	0	-1	-4	-3	-3	4	1	-1	-4
Z	-1	0	0	1	-3	3	4	-2	0	-3	-3	1	-1	-3	-1	0	-1	-3	-2	-2	1	4	-1	-4
X	0	-1	-1	-1	-2	-1	-1	-1	-1	-1	-1	-1	-1	-1	-2	0	0	-2	-1	-1	-1	-1	-1	-4
*	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	1

Figura 2.3: Matriz de sustitución BLOSUM62

2.1.3. Clasificación de algoritmos

Los algoritmos para alineamientos de secuencias se pueden clasificar de acuerdo a dos propiedades. En primer lugar, si el algoritmo produce un alineamiento global o uno local. En segundo lugar, si el algoritmo se basa en programación dinámica o en heurísticas. Una representación esquemática de la clasificación se puede ver en la Figura 2.4.

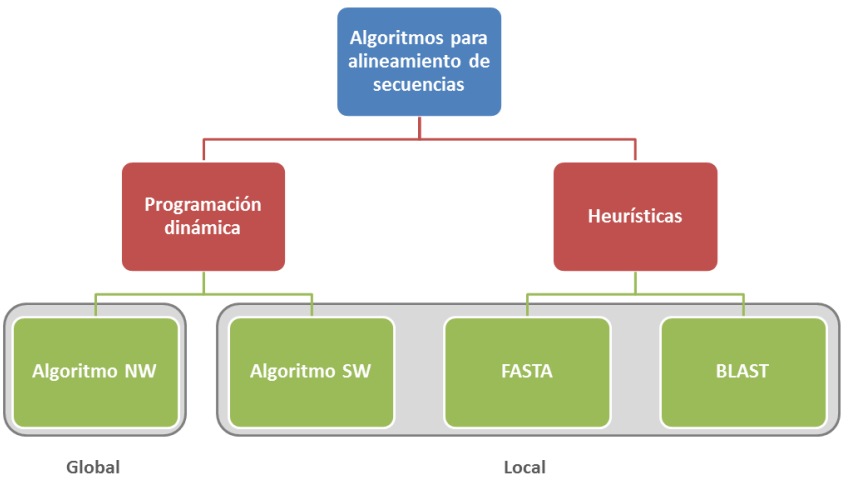

Figura 2.4: Clasificación de algoritmos para alineamiento de secuencias

2.1.3.1. Alineamientos globales y locales

Los alineamientos de secuencias se pueden clasificar como globales o locales. Los alineamientos globales calculan la similitud considerando toda la longitud de las secuencias y son útiles cuando las secuencias son muy parecidas entre sí. A diferencia de los alineamientos globales, los alineamientos locales consideran la similitud entre pequeñas regiones de las dos secuencias. Esto permite exponer de mejor manera las similitudes entre secuencias desiguales y, a su vez, dotarla de resultados con mayor sentido biológico. Es por ello que los alineamientos globales casi no son utilizados [34].

En la Figura 2.5 se muestra un alineamiento global y otro local para un mismo par de secuencias de proteínas. Se puede notar en la figura que la inserción de *gaps* en el alineamiento global apunta a aumentar la similitud desde el principio hasta el fin de las secuencias. En cambio, en el alineamiento local, la inserción de *gaps* beneficia la concentración de regiones de similitud.

global	F	T	F	T	A	L	I	L	L	A	V	A	V
	F	-	-	T	A	L	-	L	L	A	-	A	V
local	F	T	F	T	A	L	I	L	L	-	A	V	A	V
	-	-	F	T	A	L	-	L	L	A	A	V	-	-

Figura 2.5: Alineamiento global y local para un mismo par de secuencias de proteínas

2.1.3.2. Programación dinámica y heurísticas

Los algoritmos para alineamiento de secuencias también se pueden clasificar de acuerdo a la metodología empleada en su resolución. La programación dinámica se basa en la estrategia *divide y vencerás* y resuelve un problema dividiéndolo en subproblemas más pequeños. La combinación de las soluciones individuales a los subproblemas permite obtener la solución al problema general. Los algoritmos basados en programación dinámica siempre encuentran la solución óptima. Desafortunadamente, deben examinar todas las formas posibles de resolver un problema, lo que suele ser computacionalmente costoso.

Los algoritmos basados en heurísticas son más *rápidos* que los basados en programación dinámica. Este aumento de velocidad se debe a una reducción en la precisión de los resultados. Los algoritmos basados en heurísticas no siempre encuentran la solución óptima, ya que sólo examinan las formas más probables de resolver un problema.

2.1.4. Algoritmos basados en programación dinámica

Los algoritmos de alineamiento basados en programación dinámica más populares son el de Needleman-Wunsch para alineamiento globales y el de Smith-Waterman para alineamiento locales.

2.1.4.1. Algoritmo Needleman-Wunsch

En 1970, Saul Needleman y Christian Wunsch propusieron un método para alinear dos secuencias de proteínas [33], conocido como algoritmo Needleman-Wunsch. Este algoritmo siempre encuentra el alineamiento global óptimo entre dos secuencias. Sin embargo, como los

alineamiento globales suelen ser de poco interés biológico, el algoritmo Needleman-Wunsch es poco utilizado.

2.1.4.2. Algoritmo Smith-Waterman

En 1981, Temple Smith y Michael Waterman propusieron un método para alineamientos locales de secuencias [35]. Este algoritmo, conocido como algoritmo SW, es una variación del algoritmo Needleman-Wunsch y ha sido utilizado como base para otros algoritmos posteriores y como patrón con el cual comparar otras técnicas de alineamiento.

El algoritmo SW computa el alineamiento óptimo entre dos secuencias siguiendo un enfoque matricial y se puede dividir en dos fases: (1) cómputo de la matriz de similitud (o también llamada matriz de alineamiento), para obtener el puntaje de alineamiento óptimo; y (2) retroceso, para obtener el alineamiento óptimo.

1. ***Cómputo de la matriz de similitud:*** dadas dos secuencias $q = q_1 q_2 q_3 \dots q_m$ y $d = d_1 d_2 d_3 \dots d_n$, se construye una matriz de similitud H de $(m+1)\times(n+1)$ elementos. Las relaciones de recurrencia para completar la matriz, con las modificaciones de Gotoh para admitir un modelo de penalización de *gap* por afinidad [44], se detallan a continuación:

$$H_{i,j} = max \begin{cases} 0 \\ H_{i-1,j-1} + SM(q_i, d_j) \\ E_{i,j} \\ F_{i,j} \end{cases} \tag{2.1}$$

$$E_{i,j} = max \begin{cases} H_{i,j-1} - G_{oe} \\ E_{i,j-1} - G_e \end{cases} \tag{2.2}$$

$$F_{i,j} = max \begin{cases} H_{i-1,j} - G_{oe} \\ F_{i-1,j} - G_e \end{cases} \tag{2.3}$$

Los residuos de la secuencia q, usualmente llamada *secuencia de consulta*, etiquetan las filas mientras que los residuos de la secuencia d, usualmente llamada *secuencia de la base de datos*, etiquetan las columnas como se muestra en la Figura 2.20. $H_{i,j}$ representa el puntaje de alineamiento para las subsecuencias de q y d terminadas en los residuos q_i y d_j, respectivamente. $E_{i,j}$ y $F_{i,j}$ son los puntajes de los alineamientos de las mismas subsecuencias de q y d pero terminadas en un *gap* en q y en d, respectivamente. SM es la matriz de sustitución que define los puntaje de sustitución entre todos los pares de residuos. G_{oe} es la suma de las penalizaciones por inserción y extensión de *gap*, mientras que G_e es la penalización por extensión de *gap*. $H_{i,j}$, $E_{i,j}$ y $F_{i,j}$ son inicializadas en 0 cuando $i = 0$ o $j = 0$. El puntaje de alineamiento óptimo S es el máximo puntaje de alineamiento de la matriz H.

2. ***Retroceso:*** el alineamiento óptimo se calcula realizando un proceso de retroceso desde la posición donde se encuentra el puntaje de alineamiento óptimo S hasta llegar a una posición donde el valor sea 0, siendo éste punto el inicio del alineamiento.

El algoritmo SW tiene complejidad temporal cuadrática. El puntaje de alineamiento óptimo se puede calcular linealmente en cuanto a espacio. Por último, para obtener el alineamiento óptimo entre las secuencias resulta necesario almacenar la matriz H completamente, por lo que su complejidad espacial se vuelve cuadrática.

Ejemplo de SW La Figura 2.6 muestra un ejemplo de alineamiento entre las secuencias de proteínas MEVKPKLY y SETAGKVI de acuerdo al método SW. El esquema de puntuación considerado es el siguiente: BLOSUM62 se emplea como matriz de puntuación mientras que las penalizaciones por inserción y extensión de *gap* se puntúan con 10 y 2, respectivamente. El segmento de alineamiento óptimo se muestra debajo de la matriz de similitud siendo el puntaje igual a 8.

	-	S	E	T	A	G	K	V	I
-	0	0	0	0	0	0	0	0	0
M	0	0	0	0	0	0	0	1	1
E	0	0	**5**	0	0	0	1	0	0
V	0	0	0	**5**	0	0	0	5	3
K	0	0	1	0	**4**	0	5	0	2
P	0	0	0	0	0	**2**	0	3	0
K	0	0	1	0	0	0	**7**	0	0
L	0	0	0	0	0	0	0	**8**	2
Y	0	0	0	0	0	0	0	0	7

```
E  T  A  G  K  V
|  -  -  -  |  -
E  V  K  P  K  L
```

Figura 2.6: Ejemplo de alineamiento entre secuencias MEVKPKLY y SETAGKVI de acuerdo al método SW

SW en la práctica El algoritmo SW siempre encuentra el alineamiento óptimo entre dos secuencias. El segmento de alineamiento óptimo es justamente lo que le interesa a los investigadores pero para obtenerlo se necesitan algunos pasos previos. Cuando un investigador tiene una porción de secuencia biológica, ya sea de ADN o de proteínas, usualmente la alinea contra una base de datos de secuencias conocidas. Por ejemplo, la base de datos de proteínas Swiss-Prot, la cual contiene cientos de miles de secuencias provenientes de humanos y animales (ver Sección 2.1.6.1). Como el alineamiento óptimo sólo tiene valor para las secuencias de mayor puntaje, en primer lugar se suele calcular sólo los puntajes de alineamiento entre la secuencia de interés (o de consulta) y todas las secuencias de la base de datos. A este proceso se lo conoce como *búsqueda biológica, búsqueda de similitud* o también *búsqueda SW*, y su resultado es el ranking de las secuencias más parecidas con respecto a la secuencia de consulta (es decir, las secuencias de mayor puntaje). Luego, cuando resulta necesario, se puede calcular los alineamientos óptimos entre la secuencia de consulta y las secuencias de la base de datos más parecidas [45]. De esta forma, se reducen los requerimientos de memoria de las búsquedas que, a su vez, impactan también en el tiempo de ejecución requerido.

2.1.5. Algoritmos basados en heurísticas

Los algoritmos basados en heurísticas más populares son BLAST [46] y FASTA [47], ambos para alineamientos locales. La clave del éxito de estos métodos consiste en el uso de técnicas de indexación para detectar pequeñas regiones de alineamiento de alta similitud. Estos alineamientos iniciales son luego extendidos (con ciertas restricciones) y sus puntajes se utilizan para priorizar aquellos pares de mayor similitud. La mayoría de las implementaciones basadas en heurísticas emplean un algoritmo SW modificado para examinar los candidatos a mejores alineamientos y luego determinar los puntajes y alineamientos finales [48]. No existe una regla clara que defina cuándo usar BLAST y cuándo FASTA; aunque BLAST suele ser más rápido, FASTA puede ser más sensitivo en determinados casos [34].

2.1.6. Bases de datos biológicas

El desarrollo del Proyecto Genoma Humano [49] provocó un crecimiento exponencial de la información biológica. En la actualidad, las secuencias de ADN y de proteínas son almacenadas, organizadas e indexadas en bases de datos donde cada secuencia cuenta con un identificador único llamado *número de acceso*. Las bases de datos biológicas son considerablemente útiles para los investigadores, ya que les dan acceso a grandes cantidades de información. Esto resulta crítico en múltiples áreas, como por ejemplo, las búsquedas de similitud: si se cuenta con pocas secuencias con las cuales comparar, las búsquedas de similitud tienen poco sentido.

2.1.6.1. Colaboración Internacional de Bases de Datos de Secuencias de Nucleótidos

La *Colaboración Internacional de Bases de Datos de Secuencias de Nucleótidos* [50] es un proyecto colaborativo entre los tres proveedores de bases de datos biológicas más grandes del mundo: el NCBI [25], el Banco de Datos de ADN de Japón [51] y el Laboratorio de Biología Molecular Europeo [52]. El objetivo consiste en que todos tengan disponible la mayor información posible y es por eso que intercambian diariamente la información de sus bases de datos. Las bases de datos más populares se describen a continuación.

GenBank GenBank [1] es una base de datos de secuencias de ADN mantenida por el NCBI. Los archivos de GenBank se dividen en grupos; algunas de estas divisiones se basan en criterios filogenéticos, mientras que otras se basan en el enfoque técnico que fue utilizado para generar la información de las secuencias. En agosto de 2015, GenBank contaba con más de 180 millones de secuencias.

UniprotKB UniprotKB es una base de datos que sólo contiene información de secuencias de proteínas y que es mantenida por el consorcio Uniprot [53]. UniprotKB se divide en dos secciones: Swiss-Prot y TrEMBL. En la primera, las secuencias son revisadas y anotadas manualmente, por lo que se considera a Swiss-Prot el *gold standard* de las bases de datos de proteínas. En cambio, en la segunda, las secuencias son anotadas en forma automática mediante métodos computacionales. Si bien la calidad de SwissProt es altamente superior en relación a TrEMBL, el número de secuencias que ofrece es mucho menor. Por ejemplo, en agosto de 2015 Swiss-Prot contaba con más de 500 mil secuencias mientras que TrEMBL poseía más de 50 millones. Más allá de contar con un número limitado de secuencias, se considera a Swiss-Prot la base de datos de proteínas de referencia.

RefSeq La base de datos RefSeq [54] es mantenida por el NCBI y contiene información tanto de secuencias de proteínas como de nucleótidos. Al igual que SwissProt, su contenido es revisado y anotado manualmente. A diferencia de GenBank, la cual puede contener múltiples variantes del mismo registro, RefSeq ofrece un único registro para cada molécula natural biológica para la mayoría de los organismos vivos. En agosto de 2015, RefSeq contaba con más de 77 millones de secuencias.

NR y Environmental NR La mayoría de las bases de datos intentan ser *no redundantes*. Sin embargo, el concepto de redundancia varía de acuerdo a cada base de datos debido a la complejidad de la información biológica. En general, lo que buscan es evitar la ocurrencia de duplicados. La base de datos Non-Redundant (NR) del NCBI compila las proteínas de varias bases de datos, entre ellas, Swiss-Prot. Cuando se encuentran dos secuencias idénticas, sólo una es mantenida. Se puede decir que NR posee prácticamente todas las secuencias de proteínas que existen.

Environmental NR es otra base de datos no redundante mantenida por el NCBI. Esta base de datos se compone de secuencias de proteínas traducidas que provienen directamente del ambiente y es por eso que todos los organismos se encuentran mezclados. El tamaño de Environmental NR es mucho menor en relación a NR. En agosto de 2015, mientras que Environmental NR contaba con más de 6 millones de secuencias, NR disponía de más 10 veces ese tamaño (aproximadamente 69 millones de secuencias).

2.2. Sistemas heterogéneos

En los últimos años, la computación heterogénea se ha consolidado como una de las estrategias para mejorar la eficiencia energética de los sistemas de cómputo de altas prestaciones. En esta sección se reseña la evolución de los procesadores detallando cómo el problema termo-energético cambió el paradigma tradicional de diseño (Sección 2.2.1). A continuación, se caracterizan las arquitecturas heterogéneas actuales en la comunidad HPC, además de analizar sus ventajas y desventajas (Sección 2.2.2). Por último, la Sección 2.2.3 presenta diferentes modelos de programación y describe las herramientas para programación paralela más populares.

2.2.1. Evolución de los procesadores

De acuerdo a la Ley de Moore, el número de transistores en el chip de un procesador se duplica cada 18-24 meses. Tradicionalmente la mejora de rendimiento de los procesadores estuvo guiada por el aumento tanto en el número de transistores en el chip como en la frecuencia del reloj. Esto permitió la implementación de técnicas avanzadas de procesamiento que se traducían en mejoras en el rendimiento de las aplicaciones [55]. La Figura 2.7 exhibe algunas de las técnicas más destacadas y su relación con el pico de rendimiento y la frecuencia de reloj. Entre las técnicas más destacadas se encuentran:

- *Pipelining*: al dividir las instrucciones en etapas resulta posible ejecutar etapas de distintas instrucciones en forma simultánea, lo que incrementa el número de instrucciones ejecutadas por ciclo de reloj. Este es el ejemplo más elemental de paralelismo a nivel de instrucciones (ILP, por sus siglas en inglés).

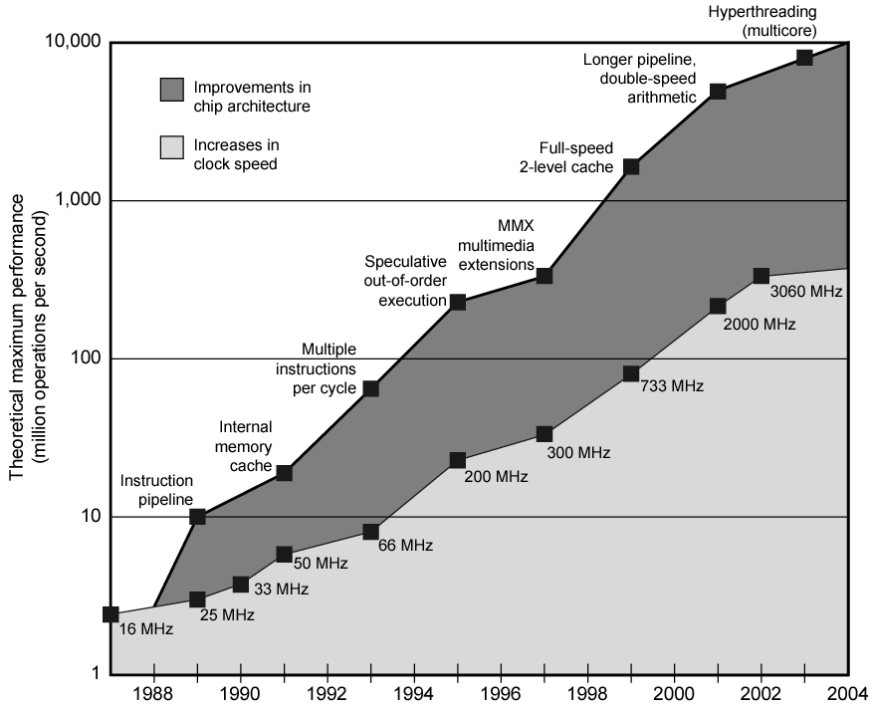

Figura 2.7: Evolución de los procesadores de acuerdo al pico de rendimiento, la frecuencia de reloj y las mejoras implementadas entre 1988 y 2004 (extraído de [2])

- Arquitectura superescalar: los procesadores superescalares con múltiples unidades funcionales son capaces de ejecutar más de una instrucción por ciclo de reloj. Para ello cuentan con un planificador a nivel de hardware que analiza el programa a ejecutar y extrae aquellas instrucciones que pueden ser ejecutadas simultáneamente.

- Instrucciones *Single Instruction Multiple Data* (SIMD): este tipo de instrucciones permite aplicar la misma operación sobre un conjunto de datos diferentes al mismo tiempo. Tantos los procesadores actuales de Intel como los de AMD cuentan con instrucciones SIMD. En los procesadores Intel, se conocen como *SIMD Streaming Extensions* (SSE) para 128 bits o *Advanced Vector Extensions* (AVX) para 256 bits. Por su parte, AMD inicialmente implementó su propio conjunto (llamadas *3dNow!*) aunque en la actualidad se ha inclinado por incluir las SSE y las AVX.

- Ejecución fuera de orden: siempre y cuando no haya dependencias, las instrucciones son reorganizadas para obtener una ejecución más eficiente.

- Predicción de saltos y ejecución especulativa: el predictor asume el valor de verdad que tendrán futuras instrucciones condicionales y avanza con la ejecución aún sin saber si el trabajo será necesario. Si el predictor acierta, entonces se evita la demora en la cual se habría incurrido por hacer el trabajo después de saber que hay que hacerlo. En sentido contrario, si el predictor no acierta, se deshacen los cambios y se ignora el trabajo realizado.

- *Simultaneous Multithreading*: mediante la duplicación de registros y otros componentes para mantener el estado de un procesador, la CPU simula tener múltiples núcleos y

permite la ejecución paralela de múltiples flujos de instrucciones (también llamados procesos lógicos o hilos hardware) sin importar si pertenecen al mismo programa o no. En los procesadores Intel, esta técnica se ha implementado bajo el nombre comercial *Hyper-Threading*.

- Memorias caché más grandes: este tipo de memoria permite mitigar la diferencia existente entre la velocidad del procesador y la de la memoria principal, lo que se conoce como *memory wall* [56]. Si bien una caché muy grande resulta más lenta que una pequeña, su tamaño ha aumentado a través del tiempo.

A principios de la década del 2000, este proceso de mejora llegó a su límite debido a dos causas principales [57]:

- Resultaba difícil extraer más ILP de los programas debido a limitaciones de los compiladores, a la limitada cantidad de paralelismo intrínseco disponible o a que la predicción de saltos lo volvía imposible. A este fenómeno se lo llamó *ILP wall*.

- No fue posible continuar incrementando la frecuencia del reloj de los procesadores ya que el consumo de energía y su disipación en forma de calor alcanzó limites insostenibles para el funcionamiento correcto de los circuitos integrados.

En consecuencia, fue necesario buscar otras alternativas en el diseño de los procesadores que permitieran continuar incrementando su rendimiento. En lugar de seguir aumentando la complejidad de la organización interna del chip, las empresas fabricantes optaron por integrar dos o más núcleos computacionales (también llamados *cores*) más simples en un sólo chip. Si bien estos núcleos son más limitados y menos veloces, al combinarlos permiten mejorar el rendimiento global del procesador y al mismo tiempo hacerlo más eficiente energéticamente [58]. A este tipo de procesadores se los denominó *multicore*. La Figura 2.8 refleja este proceso mostrando la evolución de los procesadores Intel de acuerdo al número de transistores, frecuencia de reloj, consumo de potencia e ILP extraíble entre 1970 y 2010. Se puede apreciar que, si bien el número de transistores continuó en aumento, la frecuencia de reloj, el consumo de potencia y el ILP extraíble se mantuvieron casi constantes desde principios de 2000. Aunque la gráfica sólo alcanza hasta el año 2010, se puede afirmar que las tendencias se mantienen en la actualidad [59].

Si el proceso de mejora tradicional de los procesadores hubiera continuado, se estima que la densidad de potencia de un procesador habría superado la correspondiente a la de un reactor nuclear para el año 2010 [60]. A la limitación impuesta por el problema energético se la denominó *power wall*. Este fenómeno no sólo afectó a las empresas fabricantes sino también a los programadores de aplicaciones. Con la introducción de los procesadores *multicore* se volvió necesario que las aplicaciones exploten paralelismo explícito para poder aprovechar la potencia del hardware subyacente; en particular, tanto paralelismo de datos como de tareas. El paralelismo de datos consiste en procesar múltiples datos al mismo tiempo. Por otra parte, el paralelismo de tareas se basa en ejecutar múltiples tareas independientes en paralelo.

A partir de su origen, los procesadores *multicore* han sofisticado su diseño en las sucesivas familias de procesadores de propósito general. Los primeros procesadores de este tipo eran prácticamente dos procesadores mononúcleo en la misma oblea. Las siguientes generaciones han incrementado el número de núcleos e incorporado niveles de caché L2 y L3, los cuales son compartidos por todos los núcleos o por grupos de ellos. La Figura 2.9 ilustra dos ejemplos de arquitecturas diferentes de procesadores de cuatro núcleos por parte de AMD y de Intel.

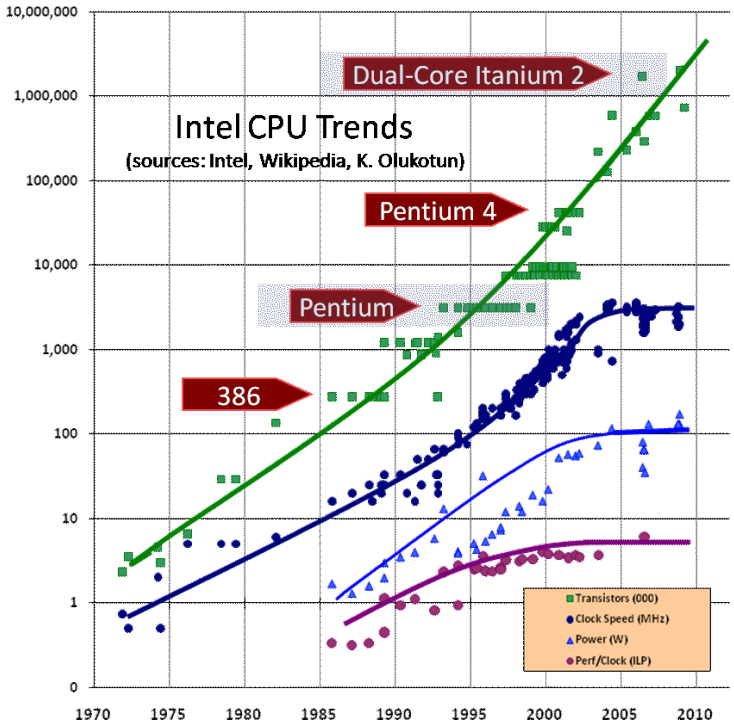

Figura 2.8: Evolución de los procesadores Intel de acuerdo al número de transistores, frecuencia de reloj, consumo de potencia e ILP extraíble entre 1970 y 2010 (extraído de [3])

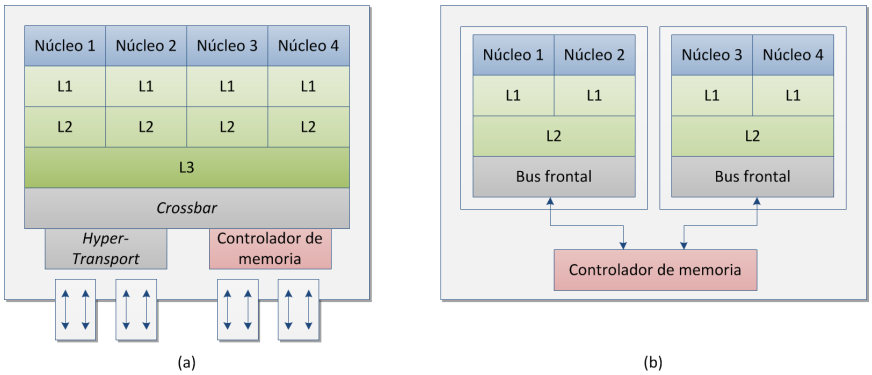

Figura 2.9: Ejemplos de diferentes arquitecturas de procesadores de cuatro núcleos. (a) AMD Opteron. (b) Intel Xeon.

2.2.1.1. Procesadores de Intel

En la actualidad, Intel ofrece una amplia gama de procesadores, cada uno orientado a diferentes usos:

- Los procesadores Atom se caracterizan por su bajo consumo de potencia, lo que los hace muy eficientes energéticamente. Es por ello que estos procesadores son usados en *netbooks*, *tablets*, *smartphones*, dispositivos para la asistencia sanitaria, autos inteligentes, entre otros.

- Los procesadores Celeron se han destacado por ser los más económicos de los ofrecidos por Intel y se los utiliza en computadoras de escritorio o portátiles. En comparación a otros procesadores del mismo fabricante, suelen contar con menos núcleos y memorias caché más pequeñas.

- Los procesadores de la línea Core en sus 3 variedades (i3, i5 e i7) ofrecen mayor potencia de cómputo en relación a los anteriores y están orientados al uso personal y profesional. Los procesadores i3 son los más modestos; todos sus modelos poseen dos núcleos con Hyper-Threading y memorias caché L3 de hasta 4MB. Los procesadores i5 son más potentes que los anteriores; la mayoría de ellos cuentan con cuatro núcleos físicos aunque existen modelos con dos núcleos e Hyper-Threading. Adicionalmente, cuentan con memorias caché más grandes (L3 de hasta 6MB) y tecnología Turbo Boost[1]. Los procesadores i7 son los más potentes de esta línea y existen modelos con dos, cuatro, seis y ocho núcleos (todos con Hyper-Threading). También cuentan con tecnología Turbo Boost y las memorias cachés más grandes de toda la línea (L3 de hasta 20MB).

- Los procesadores Xeon son los más potentes de los fabricados por Intel y son utilizados en *workstations* y servidores. Estos procesadores fueron pensados para aplicaciones con cargas de trabajo pesadas que requieren eficiencia y confiabilidad en su procesamiento. Entres sus principales ventajas se encuentran la capacidad de soportar múltiples *sockets*, un mayor número de núcleos (algunos de los modelos cuentan con hasta 18 núcleos con Hyper-Threading), memorias caché más grandes (L3 de hasta 45MB) y soporte para la detección y corrección de errores en la memoria RAM.

A partir del 2006, Intel ha empleado un modelo de desarrollo para sus procesadores que alterna entre etapas diferentes y al que ha bautizado *tick tock*. Cada *tick* implica una mejora en la tecnología de fabricación lo que le permite hacer transistores más pequeños. Por otro lado, cada *tock* representa el diseño de una nueva microarquitectura. La Figura 2.10 ilustra el modelo *tick tock* seguido por Intel desde 2006 a la actualidad.

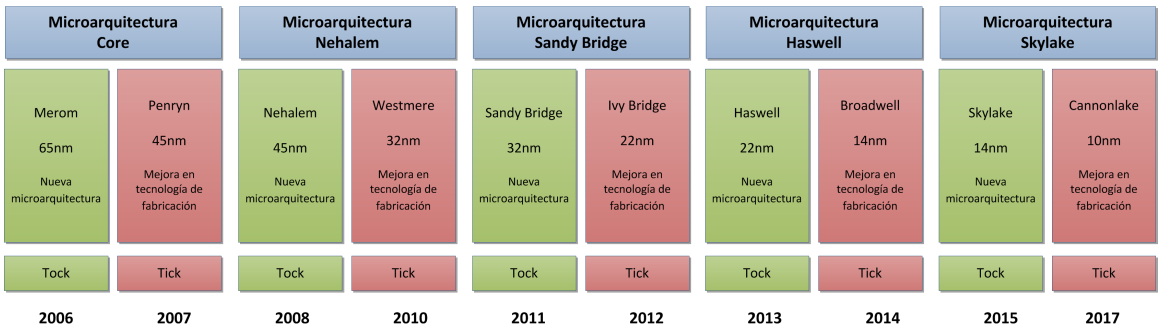

Figura 2.10: Modelo *tick tock* de Intel

En los últimos meses de 2015, Intel ha lanzado los primeros procesadores Skylake y más modelos de esta familia se esperan para 2016 y 2017. En particular, la línea Xeon

[1]La tecnología Turbo Boost de Intel permite incrementar el desempeño de sus procesadores al aumentar automáticamente la frecuencia del reloj de los núcleos por encima de la frecuencia operativa nominal, siemple y cuando no se hayan alcanzado los límites especificados de energía, corriente y temperatura. La activación de esta tecnología y el tiempo durante el cual la misma opera sobre el procesador dependen de la carga de trabajo y del entorno operativo [61].

incorporará mejoras significativas, entre ellas: soporte para un número mayor de *sockets* y más canales de memoria DDR4 por CPU y de buses PCI Express 3. Además integrará el conjunto de instrucciones vectoriales AVX-512 de 512 bits [2] (una extensión a 512 bits de las instrucciones AVX actuales) y dará soporte a FPGAs en forma integrada [62, 63]. Por último, aunque originalmente fuera planificado para 2016, Intel ha reprogramado el siguiente *tick* a 2017 debido a demoras sufridas en el proceso de reducción a 10nm de la tecnología de fabricación [64, 65].

2.2.1.2. Procesadores de AMD

Al igual que Intel, AMD ofrece una amplia gama de procesadores diseñados para diferentes propósitos. En el año 2006, AMD compró la empresa de placas gráficas ATI, lo que le permitió incrementar notablemente su capacidad de producir e innovar hardware gráfico. Como resultado, AMD desarrolló las Unidades de Procesamiento Acelerado (APU, por sus siglas en inglés). Estas unidades combinan una CPU y una GPU en el mismo chip, por lo que las computadoras con esta clase de procesadores no requiere de una placa gráfica integrada en el *motherboard* o de una dedicada. En el caso de las GPUs, el hardware adopta la arquitectura *Graphics Core Next* (GCN) que se describe más adelante en la Sección 2.2.2.1. Las APUs pueden ser una buena opción desde el punto de vista del compromiso precio-rendimiento con buenos cocientes de eficiencia energética y se ofrecen para computadoras de escritorio y *notebooks* [66]. En la actualidad, AMD ofrece los siguientes procesadores:

- En la categoría APU, se encuentran las líneas Sempron, Athlon y A-Series, ordenados ascendentemente de acuerdo a su potencia de cómputo. La línea Sempron se compone de procesadores poco potentes y de bajo costo. A noviembre de 2015, AMD ofrece sólo dos modelos de esta línea con dos y cuatro núcleos. La línea Athlon ofrece APUs un poco más poderosas con dos modelos *quad-core* y memoria caché L2 de 2MB. Las APUs A-Series son los más potentes de esta categoría y a su vez la que ofrece más modelos diferentes. La memoria caché L2 se extiende a 4MB mientras que el número de núcleos de la CPU varía de dos a cuatro y las unidades de cómputo correspondientes a la GPU alternan entre cuatro y ocho.

- En la categoría CPU, se encuentran las líneas Athlon, FX y Opteron. La línea Athlon de CPU ofrece un buen cociente precio-rendimiento y puede ser una buena opción para una computadora de uso personal. Se ofrecen en configuraciones de dos y cuatro núcleos con memoria caché L2 de 1 y 2 MB, respectivamente. Los procesadores FX son los más potentes para el segmento de escritorio y sus modelos poseen ocho núcleos y memoria caché L3 de hasta 8MB. Por otra parte, la línea Opteron está destinada a los servidores y *workstations*. Estos potentes procesadores integran hasta 16 núcleos y 16MB de memoria caché L3.

A diferencia de Intel, AMD no sigue un modelo estipulado para el desarrollo de sus microarquitecturas. La Figura 2.11 muestra las microarquitecturas desarrolladas por AMD a partir de 2011. La familia Fusion se introdujo en el 2011 y corresponde a las APUs que integran CPU y GPU en el mismo chip. La familia Bobcat está orientada al segmento de bajo consumo y de bajo costo. La primera microarquitectura de esta familia se presentó en el 2011 mientras que la segunda (Jaguar) y la tercera (Puma) generación se introdujeron en el 2013

[2] Extensiones AVX-512: https://software.intel.com/en-us/blogs/additional-avx-512-instructions

y 2014, respectivamente. Por otra parte, la familia Bulldozer fue diseñada para computadoras de escritorio y servidores. Su primera microarquitectura se presentó en 2011 y cuenta con tres generaciones posteriores: Piledriver, Steamroller y Excavator en 2012, 2014 y 2015, respectivamente. Por última, la siguiente microarquitectura de AMD se llama Zen y será introducida a fines de 2016. El objetivo principal de Zen consiste en mejorar el rendimiento por núcleo más que aumentar su número o la cantidad de hilos hardware. En particular, algunos reportes preliminares de AMD indican hasta un 40 % más de instrucciones por ciclo de reloj [67]. A diferencia de las familias anteriores, Zen adoptará capacidades SMT y sus procesadores serán fabricados con tecnología de 14nm.

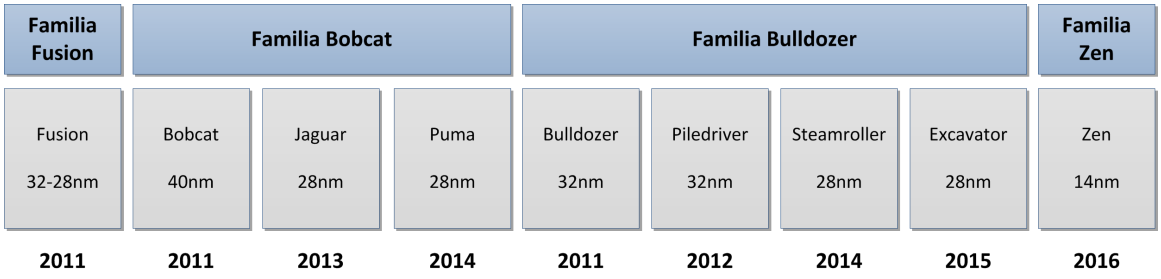

Figura 2.11: Microarquitecturas de procesadores desarrolladas por AMD a partir de 2011

2.2.2. Consolidación de aceleradores en HPC

Como se mencionó en la Sección 1.1, la computación heterogénea se presenta como una de las estrategias más elegidas para mejorar la eficiencia energética de los sistemas HPC. En una computadora heterogénea, dos o más tipos de procesadores, generalmente con diferentes conjuntos de instrucciones, son combinados en una única máquina para proporcionar un ambiente de cómputo flexible para diferentes tipos de aplicaciones. La estrategia consiste básicamente en incorporar tecnología de aceleradores y de coprocesadores a los nodos convencionales con procesadores de propósito general [68].

En la actualidad, existe una amplia variedad de aceleradores y coprocesadores, siendo los más populares las GPUs (mayoritariamente de NVIDIA) seguidos por los coprocesadores Xeon Phi de Intel. Las FPGAs aparecen como una opción promisoria para HPC debido a su capacidad de cómputo creciente, su bajo consumo energético y al desarrollo de nuevas herramientas que facilitan su programación [19]. La reciente compra de Altera por parte de Intel probablemente expanda aun más el uso de esta clase de aceleradores [20].

Si bien cada dispositivo cuenta con sus características propias, existen diversos aspectos que son comunes a todos ellos:

- Complejidad de programación: la programación de estos sistemas heterogéneos se convierte en un verdadero desafío debido a las dificultades que deben afrontar los programadores para lograr alto rendimiento en sus aplicaciones. Algunas de las dificultades pueden ser: contemplar aplicaciones con múltiples niveles de paralelismo y aplicar técnicas de programación y optimización particulares para cada una de ellos, lograr una distribución del trabajo y un balance de carga entre los diferentes dispositivos de procesamiento que permita obtener buen rendimiento y afrontar el surgimiento de nuevos lenguajes y modelos de programación junto a la ausencia de un estándar para este tipo de sistemas.

- Memoria del dispositivo separada: los aceleradores y coprocesadores actuales son incorporados como dispositivos de E/S, por lo que su memoria se encuentra separada de la del *host*. En consecuencia, la administración de la memoria es una de las claves para obtener buen rendimiento. En ese sentido, el esquema de memoria unificada propuesto por NVIDIA para su familia de placas Maxwell representa un avance [69].

- Patrón de acceso a la memoria: las memorias de los dispositivos se encuentran optimizados para tipos de acceso específicos a cada uno, los cuales difieren de los correspondientes a los procesadores de propósito general. Si los datos son accedidos convenientemente, entonces la aplicación puede aprovechar el ancho de banda provisto por el hardware.

Existe otro conjunto de características comunes pero que estrictamente sólo comparten los coprocesadores Xeon Phi y las GPUs. Como las FPGAs tienen la capacidad de reconfigurar su hardware, pueden o no compartir las siguientes características dependiendo de su configuración. Entre estas características se encuentran [70]:

- *Many-cores*: todos los dispositivos cuentan una gran cantidad de núcleos que son capaces emitir instrucciones en una o más unidades funcionales y que comparten recursos con otras unidades. La organización de los núcleos y sus unidades funcionales difiere de acuerdo al dispositivo.

- Multihilado: los tres dispositivos emplean técnicas de multihilado para ocultar la latencia de ciertas operaciones, como pueden ser las de memoria.

- Vectorización: los tres dispositivos tienen amplias capacidades SIMD.

- Ejecución de instrucciones en orden: comparado a los núcleos de las CPUs convencionales, las unidades de control de estos dispositivos son más sencillas.

- Memorias caché más pequeñas: las memorias caché por núcleo son más pequeñas en estos dispositivos que en los procesadores de propósito general actuales.

A continuación se describen las características más importantes de estos tres tipos de arquitecturas.

2.2.2.1. GPU

Las GPUs fueron originalmente diseñadas para los juegos de computadoras. La creciente demanda por gráficos de alta calidad permitió incrementar la potencia de cálculo de las GPUs y las empresas fabricantes comenzaron a aumentar el grado de programación de las mismas. Esto motivó el surgimiento de nuevas técnicas, lenguajes y herramientas para la programación de GPUs, lo cual permite utilizar a las mismas como arquitecturas paralelas para resolver problemas de propósito general y no sólo los relacionados a computación gráfica.

La gran diferencia de rendimiento que existe entre las CPUs y las GPUs se debe a que sus filosofías de diseño son muy distintas. Las CPUs destinan los recursos de silicio principalmente a memorias caché y a núcleos de compleja organización que permitan explotar ILP. En sentido opuesto, las GPUs emplean la mayor parte del silicio disponible en unidades funcionales. Cada una de ellas tiene un conjunto de núcleos simples que ejecutan instrucciones en orden y operan en grupos como si fueran un procesador vectorial. La simplicidad de los núcleos y su capacidad vectorial, compartiendo la lógica de control, es lo que posibilita

tener miles de núcleos en un sólo chip y picos de rendimientos muy superiores a las CPUs [59].

A pesar de tener tasas de ancho de banda mucho mayores que las CPUs, la latencia de la memoria se encuentra todavía a cientos de ciclos de reloj, por lo que las GPUs emplean técnicas de multihilado con hasta varias decenas de hilos por núcleo para tolerarla. Cada uno de estos hilos tiene sus propios registros, por lo que el planificador de instrucciones puede cambiar rápidamente de un hilo a otro, dependiendo de cual ya dispone de sus datos para computar [71].

Las primeras aplicaciones no gráficas para GPUs eran programadas en término de operaciones gráficas usando lenguajes como OpenGL o DirectX. Como esto resultaba engorroso y propenso a errores, tanto la industria como la academia propusieron varios lenguajes que permiten abstraerse de los gráficos. En la actualidad, los lenguajes de programación de propósito general para GPUs más populares son CUDA (Sección 2.2.3.3), OpenCL (Sección 2.2.3.4) y, en menor medida, OpenACC [72].

Al día de hoy, son 3 las empresas que comparten el mercado de las GPUs, siendo Intel la más grande. Sin embargo, Intel sólo domina el segmento correspondiente a placas integradas y de bajo rendimiento. En el segmento de alto rendimiento, AMD y NVIDIA son los únicos proveedores. Tanto en ámbitos industriales como académicos, NVIDIA supera ampliamente a AMD, en parte debido al desarrollo de CUDA (ver Sección 2.2.3.3) [73].

GPUs de NVIDIA Los elementos claves en el hardware de una GPU de NVIDIA son los diferentes tipos de memoria, los *Streaming Multiprocessors* (SMs) y los *Streaming Processors* (SPs). Una GPU de este fabricante consiste de uno o más SMs. Cada SM cuenta con múltiples SPs, un espacio de memoria dedicado a registros para cada SP y un espacio de memoria compartida sólo accesible por sí mismo. Además, cada SM cuenta con un bus separado para acceder a una memoria global, a una memoria constante y a una memoria de textura. La memoria global se compone de módulos de memoria GDDR. La memoria de textura es una vista especial de la memoria global que resulta útil cuando se requiere interpolación, por ejemplo, cuando se requiere búsquedas en tablas de dos o tres dimensiones. La memoria constante es una memoria de sólo lectura y baja latencia. Al igual que la memoria de textura, es una vista especial de la memoria global [74].

Las GPUs actuales de NVIDIA se basan en la arquitectura Maxwell, la cual fue presentada en el año 2014 [75]. Los cambios introducidos por Maxwell con respecto a su predecesora, Kepler, estuvieron orientados a mejorar la eficiencia energética de las placas. Las GPUs Maxwell se componen de un arreglo de *clusters* de procesamiento gráfico (GPC, por sus siglas en inglés) y de controladores de memoria, entre otros elementos, como se ilustra en la Figura 2.12. Cada GPC a su vez consiste de un número de SMs, que en esta arquitectura se denominan *Maxwell Streaming Multiprocessor* (SMM). El número de GPCs y de SMMs por GPC varía de acuerdo al modelo. Los SMMs utilizan un diseño basado en cuadrante con cuatro bloques de procesamiento de 32 núcleos CUDA (128 núcleos CUDA por SMM), donde cada bloque cuenta con un planificador dedicado capaz de despachar dos instrucciones por ciclo. Además, cada SMM dispone un *PolyMorph Engine*, una unidad de hardware gráfico.

La jerarquía de memoria de los SMMs también ha cambiado en relación a la arquitectura Kepler. En lugar de implementar un bloque de memoria compartida combinado con una memoria caché L1, los SMMs de Maxwell poseen un bloque de memoria compartida dedicado, mientras que los bloques de caché L1 se han combinado con los de textura [76].

NVIDIA ha anunciado que la próxima arquitectura de sus GPUs se llamará Pascal y

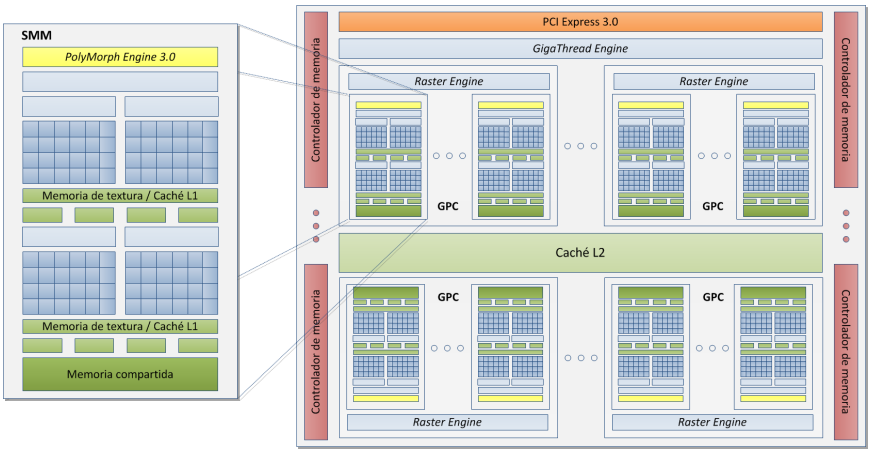

Figura 2.12: Arquitectura Maxwell de NVIDIA

será presentada en 2016. Los principales cambios están orientados a mejorar la organización de la memoria y los buses de interconexión.

GPUs de AMD En el año 2011, AMD presentó la arquitectura GCN, la cual representa la base de sus GPUs actuales tanto para el segmento de sus placas individuales como para el de las integradas con CPUs. Con su introducción, AMD dejó atrás un diseño basado en *Very Long Instruction Word* (VLIW) por otro basado en SIMD. Aunque ambos esquemas comparten similitudes, su ejecución es muy diferente. Mientras que VLIW se basa en extraer el mayor ILP posible, SIMD se apoya principalmente en paralelismo a nivel de hilo. GCN fue pensada para lograr buenos rendimientos no sólo en tareas gráficas sino también en tareas de cómputo general [77].

Una GPU de AMD se compone de una colección de arreglos de *Compute Units* (CUs), como ilustra la Figura 2.13. Un CU es una unidad de cómputo y se la considerada una pieza fundamental en el diseño de las GPUs. Cada CU contiene un conjunto de unidades SIMD, unidades para planificación de instrucciones, una unidad escalar y distintos bloques de memoria caché L1.

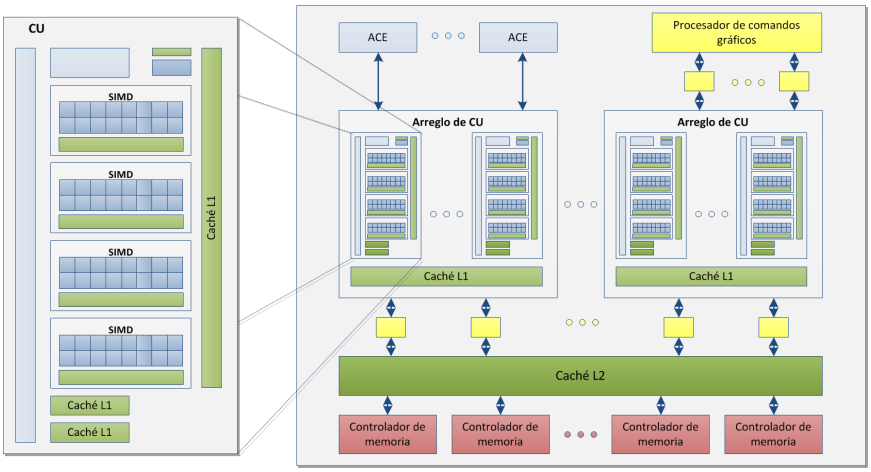

Figura 2.13: Arquitectura *Graphics Core Next* de AMD

Además de las CUs, las GPUs basadas en GCN poseen otros componentes como memoria caché, controladores de memoria, hardware para gráficos y para planificación. Con respecto a la memoria, la caché L2 se encuentra emparejada a los controladores de memoria. En relación al cómputo, GCN introduce los *Asynchronous Compute Engines*, los cuales sirven como procesadores de comandos. Su función principal consiste en aceptar trabajo y despacharlo a los diferentes CUs para su correspondiente procesamiento [78].

Mediante GCN, AMD pretende posicionar a las GPUs a la misma altura que las CPUs en lugar de tratarlas como un dispositivo periférico. Una de las innovaciones más importantes de AMD en los últimos años es su tecnología Memoria de Alto Ancho de Banda (HBM, por sus siglas en inglés), desarrollada en colaboración con la empresa de semiconductores SK Hynix. HBM es una nuevo tipo de memoria RAM que organiza los chips de memoria en forma vertical y apilada, lo que se conoce como *memoria 3D*, y que puede ser aprovechado tanto por las GPUs como las CPUs. Esta organización produce múltiples mejoras: significativo ahorro de espacio y considerables aumentos en la velocidad de comunicación y en la eficiencia energética [79]. HBM ya ha sido incorporada en las GPUs con nombre clave Fiji de la arquitectura GCN, lanzadas al mercado en el segundo semestre de 2015, y más placas de AMD con esta tecnología se esperan para 2016. Además, HBM también será adoptada por NVIDIA para sus placas de la arquitectura Pascal, las cuales serán introducidas en 2016.

2.2.2.2. Xeon Phi

En el año 2012, Intel presentó los coprocesadores Xeon Phi, su propuesta para el segmento de aceleradores en HPC. Este dispositivo es un *many-core* dentro de la arquitectura *Many Integrated Cores* que deriva del proyecto discontinuado Larrabee [80] y del proyecto Teraflops Research Chip. En su actual generación, el Xeon Phi se compone de hasta 61 núcleos Pentium x86 con unidades vectoriales extendidas de 512 bits a través del conjunto de instrucciones Knights Corner (KNC). Además, cada núcleo cuenta con *Hyper-Threading* (cuatro hilos hardware por núcleo) e integra una caché L1 (32 Kb de datos + 32Kb de instrucciones) y una caché L2 asociada completamente coherente (512 Kb de datos e instrucciones). Como se ilustra en la Figura 2.14, una interconexión de alta velocidad en forma de anillo permite transferencias de datos entre todas las memorias caché L2 del coprocesador y el subsistema de memoria. El Xeon Phi soporta hasta 8 controladores de memoria, cada uno con dos canales GDDR5, y se conecta al *host* a través del bus PCIe Gen2.

Desde el punto de vista de la programación, una de las ventajas de esta plataforma es el soporte a modelos de programación paralela tradicionalmente usados en sistemas HPC como OpenMP (Sección 2.2.3.1) o MPI (Sección 2.2.3.2), lo que simplifica el desarrollo de código y mejora la portabilidad sobre otras alternativas de lenguajes de programación específicos para aceleradores como OpenCL o CUDA.

Los coprocesadores Xeon Phi admiten dos modos de ejecución diferentes: nativo y *offload*. El modo nativo consiste en emplear a los Xeon Phi como un sistema de cómputo autónomo, lo que permite ejecutar aplicaciones usando solamente los recursos del coprocesador. Construir una aplicación nativa usualmente requiere pequeñas modificaciones de código. De hecho, muchos códigos HPC escritos para *clusters* de procesadores de propósito general pueden ser ejecutados en este modo con facilidad, simplemente recompilándolos para esta plataforma con el *flag* `-mmic` del compilador. Sin embargo, como la clave del rendimiento del Xeon Phi se basa en el uso eficiente de las unidades vectoriales de los núcleos y de la pequeña caché, un buen desempeño probablemente no sea logrado después de una migración directa de código.

El modo nativo puede resultar ineficiente para algunas aplicaciones con tasas altas de

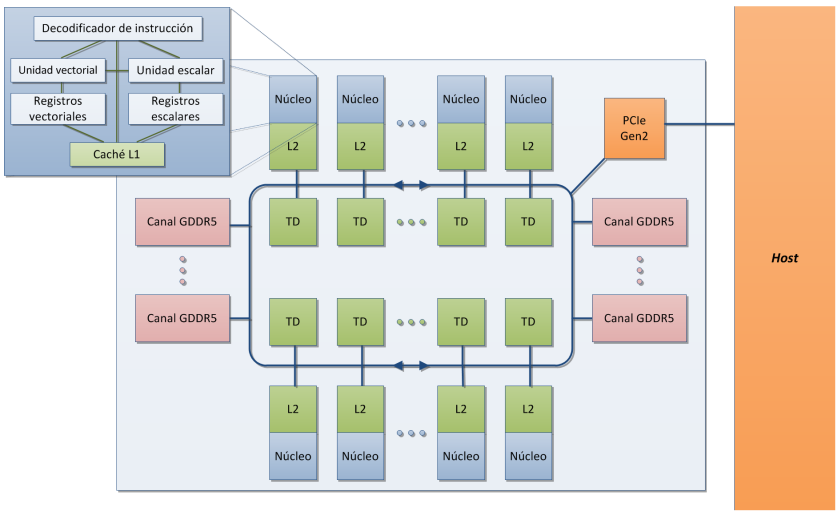

Figura 2.14: Arquitectura del coprocesador Xeon Phi

E/S o con muchas secciones de código secuencial. En esos casos, es recomendable usar el Xeon Phi como un coprocesador (modo *offload*), lo que resulta similar al modo de otros aceleradores como son las GPUs compatibles con CUDA. El *host* ejecuta el código secuencial de la aplicación e invoca la ejecución de *kernels* en el Xeon Phi.

Más allá del modo de ejecución utilizado, los principales aspectos que se deben considerar para obtener alto rendimiento son:

- cómo diseñar el cómputo para mapearlo eficientemente a las unidades vectoriales del Xeon Phi;
- cómo explotar la jerarquía de memoria, especialmente cuando se procesan grandes conjunto de datos.

Idealmente, los programadores sólo requerirían introducir algunas directivas para notificar al compilador sobre desambiguación de punteros y alineamiento y dependencias de datos para posibilitar la vectorización automática. Sin embargo, en esencia, la vectorización guiada no es capaz de lograr el mejor rendimiento posible, por lo que los programadores usualmente necesitan esforzarse y optimizar su código manualmente mediante el uso de las intrínsecas del lenguaje. Si bien el uso de estas primitivas de bajo nivel suelen inhibir otras optimizaciones a nivel de bucle, los códigos altamente optimizados en forma manual suelen superar a sus contrapartes guiadas. De hecho, el uso de intrínsecas es la única opción para aplicaciones complejas que sufren de dependencias de datos o patrones de accesos irregulares que pueden ser ocultadas mediante transformaciones específicas de código. Desafortunadamente, la mejora de rendimiento se logra a costo de perder portabilidad entre plataformas. La mayoría de las familias de procesadores, aun del mismo vendedor, tienen intrínsecas incompatibles que soportan diferentes conjuntos de instrucciones SIMD. Como consecuencia, los desarrolladores necesitan escribir diferentes versiones del código, lo que incrementa los requerimientos de mantenimiento.

Con respecto al futuro de los coprocesadores Xeon Phi, Intel ha anunciado su siguiente generación, la cual se llamará Knights Landing y estará disponible durante 2016 (aunque

originalmente fue anunciada para el segundo semestre del año 2015). Entre las principales diferencias anunciadas, el dispositivo será construido con tecnología de 14nm y será capaz de operar como una CPU autónoma en lugar de un coprocesador. También incorporará procesadores Intel Atom con capacidades vectoriales AVX-512, lo que le permitirá unificar conjunto de instrucciones SIMD con los procesadores de propósito general Xeon Skylake. Por último, la memoria principal tendrá una organización 3D en forma similar a la propuesta de HBM [81].

2.2.2.3. FPGA

La primera FPGA comercial fue desarrollada por Ross Freeman, co-fundador de Xilinx, en la década del '80. Desde entonces, las FPGAs han evolucionado significativamente incorporando características como la adopción de estándares de E/S de alta velocidad, mejoras en la compatibilidad con las CPUs y un continuo aumento en la cantidad de recursos de las placas. Aunque inicialmente fueron utilizadas para el procesamiento digital de señales, en la actualidad se han vuelto atractivas para otras áreas de aplicación como centros de cómputo, sistemas aeroespaciales y de defensa, sistema automotrices, criptografía, entre otros.

Las FPGAs son circuitos integrados reconfigurables compuestos por interconexiones programables que unen bloques lógicos programables, bloques de memoria embebidos y bloques de DSPs, como se ilustra en la Figura 2.15. La comunicación con el exterior se realiza a través de los bloques de E/S, los cuales se organizan en forma de anillo alrededor de la circunferencia del dispositivo. En la actualidad, algunas FPGAs incluyen componentes adicionales como, por ejemplo, núcleos de procesadores [82].

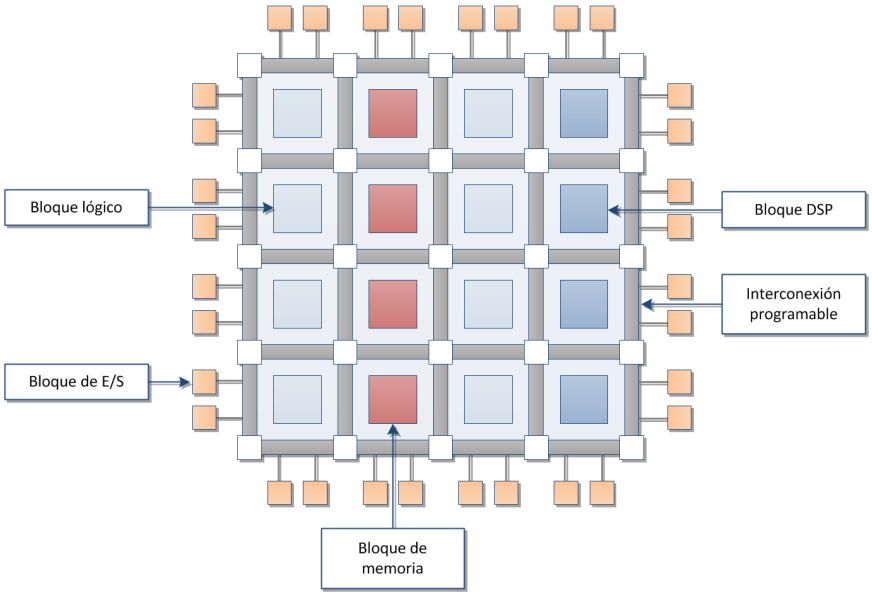

Figura 2.15: Arquitectura de una FPGA

A diferencia de las topologías y las rutas estáticas de datos que procesan las instrucciones de los programas en CPUs y GPUs, los recursos de las FPGAs puede ser configurados e interconectados para crear *pipelines* de instrucciones a medida en los cuales procesar los datos. Si bien tanto la frecuencia del reloj como el pico de rendimiento suelen ser más bajos que los correspondientes a las CPUs y a las GPUs, la capacidad de configurar el hardware

para que se adapte al problema específico a resolver le da la posibilidad de obtener mejores rendimientos. Además, como no hay desperdicio de recursos de silicio, en general son más eficientes desde el punto de vista energético [21, 83].

Xilinx y Altera son las dos empresas líderes en la industria de las FPGAs. Xilinx tuvo tradicionalmente dos series de FPGAs: Virtex y Spartan. Las placas de la serie Virtex ofrecen el mayor rendimiento y el menor consumo, mientras que las de Spartan son de bajo costo y de alto volumen. En el año 2012, Xilinx decidió discontinuar el desarrollo de la serie Spartan e introducir otras dos series adicionales que unifican su arquitectura con la serie Virtex: Kintex es la primera serie de gama media ofreciendo un balance entre rendimiento, consumo y precio; Artix ofrece el menor consumo y costo a expensas de un menor rendimiento. Por su parte, Altera divide a sus FPGAs en tres series con características similares a las de Xilinx: Stratix (Virtex), Arria (Kintex) y Cyclone (Artix).

Más allá de la empresa fabricante, la programación de estas placas se realiza a través de HDLs, como pueden ser Verilog o VHDL. Desafortunadamente, los HDLs son engorrosos, propensos a errores y requieren tener que mantener una explícita noción del tiempo, lo que implica una complejidad adicional [22]. En la actualidad, tanto Altera como Xilinx trabajan en el desarrollo de herramientas que permitan disminuir el esfuerzo de programación de estos dispositivos; en particular, a través del estándar OpenCL (ver Sección 2.2.3.4).

FPGAs de Altera De las tres series de FPGAs ofrecidas por Altera, Stratix presenta las de mayor rendimiento, ancho de banda y cantidad de recursos. La última familia de esta serie se llama Stratix V y se introdujo en 2010. Estas placas se fabrican con tecnología de 28nm y presentan una arquitectura bidimensional: un conjunto de interconexiones organizadas en filas y columnas interconecta bloques lógicos, bloques de memoria y bloques DSP. La Figura 2.16 ilustra la arquitectura de una FPGA Stratix V. Los Módulos de Lógica Adaptativa (ALM, por sus siglas en inglés) constituyen el bloque básico de construcción de estos dispositivos. Por otro lado, los bloques de memoria M20K pueden ser utilizados para almacenar el código del procesador o en la implementación de esquemas de búsqueda o de aplicaciones con grandes requerimientos de memoria. Por último, los bloques DSP se emplean para el procesamiento de señales, como lo indica su nombre, y permiten configurar su precisión.

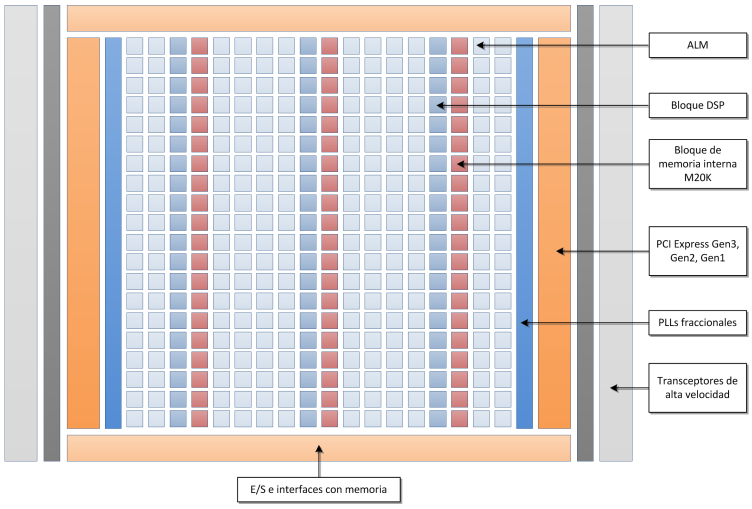

Figura 2.16: Arquitectura de una FPGA Stratix V de Altera

2.2.3. Modelos de programación y herramientas para programación paralela

La programación paralela es una tarea difícil que resulta más compleja que la programación secuencial. Para lograr alto rendimiento en una aplicación, el programador debe enfrentar múltiples dificultades que no están presentes en el paradigma secuencial, como la descomposición del problema en tareas, el mapeo de las tareas a las unidades de procesamiento y la sincronización y/o comunicación entre las tareas. A su vez, el no determinismo asociado a las ejecuciones concurrentes y paralelas hacen aun más difícil la prueba y la depuración de las aplicaciones.

Una decisión que se debe tomar al momento de desarrollar una aplicación paralela es qué modelo de programación usar. Esta decisión reviste importancia ya que afectará la elección del lenguaje y la librería a utilizar para implementar la aplicación. Las dos opciones principales son las siguientes:

- En el modelo de *memoria compartida*, todas las tareas comparten un espacio de direcciones común, en el cual pueden leer y escribir asincrónicamente. Mecanismos como *locks* y semáforos se suelen utilizar para controlar el acceso a la memoria compartida.
- En el modelo de *pasaje de mensajes*, cada tarea tiene su propio espacio de direcciones. Para intercambiar datos, las tareas envían y reciben mensajes.

En la última década, un tercer modelo ha surgido motivado por la incorporación de los procesadores *multicore* a las arquitecturas de *clusters*. Este modelo combina características de los dos anteriores y por eso se lo denomina *híbrido*. Existe un conjunto de tareas principales que poseen su propio espacio de direcciones y se comunican mediante mensajes. Cada una de estas tareas se divide en subtareas que comparten el espacio de direcciones de la tarea principal y que se comunican y se sincronizan accediendo a esta memoria común.

Existe una amplia variedad de lenguajes y librerías para desarrollar programas paralelos, los cuales se pueden clasificar de acuerdo al grado de asistencia que le ofrecen al programador:

- Completamente automático: el compilador analiza el código fuente y genera el código paralelo correspondiente.
- Guiado por el programador: el programador indica explícitamente (en general a través de directivas) qué secciones del código fuente paralelizar y también cómo hacerlo, pero es el compilador el que genera el código paralelo.
- Completamente manual: el programador es responsable de todos los aspectos de la paralelización de la aplicación.

En la actualidad, los compiladores que generan código paralelo no son una opción real. Esto se debe a los pobres resultados que producen y al lento progreso en sus capacidades.

El modelo guiado por el programador suele ser más productivo, ya que permite generar código paralelo a partir de uno secuencial con moderado esfuerzo por parte del programador. OpenMP es el principal exponente de este enfoque.

El modelo completamente manual resulta ser el más costoso en términos de esfuerzo de programación. Sin embargo, es el que permite obtener generalmente los mejores resultados en términos de rendimiento. Todo el paralelismo se expone ante el programador y él es el responsable de manejar todos sus aspectos. Este enfoque se utiliza en la programación de sistemas de memoria distribuida, siendo MPI la herramienta más utilizada, y también en la de aceleradores con CUDA y OpenCL como los ejemplos más representativos.

2.2.3.1. OpenMP

Open Multi-Processing (OpenMP) es un estándar de programación para sistemas de memoria compartida. OpenMP no es un lenguaje de programación sino una interfaz de programación que puede ser usada para extender programas secuenciales en C, C++ o Fortran y así obtener programas paralelos de memoria compartida equivalentes. La interfaz se compone de tres elementos principales: directivas al compilador, funciones de librería y variables de entornos. Las directivas al compilador proveen soporte para concurrencia, sincronización y manejo de datos, mientras que las funciones de librería y las variables de entorno permiten controlar el sistema de ejecución [84].

El modelo de programación de OpenMP sigue un enfoque de bloque estructurado y se basa en el modelo *fork-join*. La ejecución de un programa en OpenMP comienza con un único hilo de control al cual se lo llama *hilo maestro*. Al alcanzar un bloque paralelo, el hilo maestro divide en múltiples hilos independientes (*fork*). Al final del bloque existe una barrera de sincronización implícita y sólo el hilo maestro continúa con la ejecución del programa (*join*), como se ilustra en la Figura 2.17. Al código encerrado por el bloque paralelo se lo denomina *región paralela* y puede explotar tanto paralelismo de datos como de tareas [85].

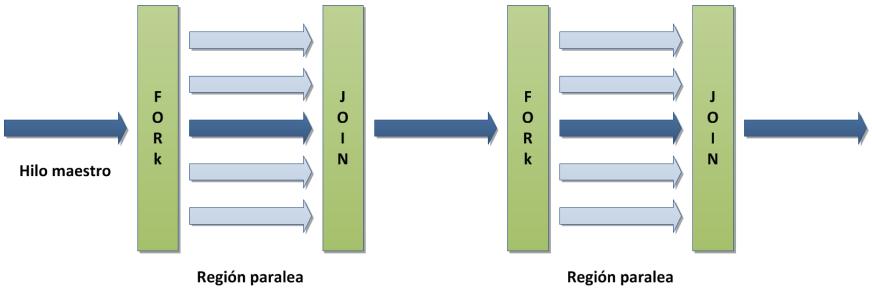

Figura 2.17: Modelo de ejecución *fork-join* en OpenMP

El modelo de memoria de OpenMP distingue entre *memoria compartida* y *memoria privada*. Todos los hilos tienen acceso a una memoria compartida. OpenMP provee directivas y funciones de librería para evitar conflictos en el acceso concurrente a los datos que se encuentran en esa memoria. Además, cada hilo cuenta con un espacio de memoria propio, el cual sólo puede ser accedido por él mismo.

OpenMP fue diseñado en 1997 por un grupo de empresas que conformaron el *OpenMP Architecture Review Board* (ARB). En la actualidad, OpenMP es un estándar ampliamente adoptado para la programación de memoria compartida y su éxito se basa en el enfoque incremental para desarrollar programas paralelos, su simplicidad y facilidad de aprendizaje, sumado a la capacidad de la ARB de mantenerlo relevante más allá de la evolución de la tecnología. La última versión (4.0) fue publicada en 2013 e incorpora la posibilidad de explotar instrucciones vectoriales y de trabajar con aceleradores, entre otras características [86].

2.2.3.2. MPI

La Interfaz de Pasaje de Mensajes (MPI, por sus siglas en inglés) es un estándar de una especificación de librería para pasaje de mensajes. MPI define la sintaxis y la semántica de un conjunto de operaciones de comunicación y puede ser utilizada con los lenguajes C, C++

y Fortran. Al ser una especificación de una interfaz, no da detalles de implementación. Por lo tanto, diferentes librerías MPI podrían emplear distintas implementaciones, posiblemente utilizando optimizaciones específicas para cada plataforma de hardware. De todas maneras, como la interfaz es estándar, la portabilidad de los programas está asegurada [84].

MPI es una interfaz amplia y compleja con más de 500 funciones que dan soporte tanto a comunicaciones punto a punto como a colectivas. Aun así, es posible resolver una gran cantidad de problemas con sólo seis funciones y la mayoría de las aplicaciones sólo emplean entre 10 y 20 funciones [85].

Existen tres versiones del estándar MPI: MPI-1 define operaciones de comunicación estándar y se basa en un modelo estático de procesos. MPI-2 extiende a MPI-1 y provee soporte adicional para manejo dinámico de procesos, comunicación *one-sided* y E/S paralela. La tercera y última versión de MPI (MPI-3) fue publicada en 2012 y revisada en 2015. MPI-3 agrega más operaciones para comunicaciones *one-sided*. Asimismo, extiende las operaciones colectivas existentes y provee versiones no bloqueantes de las mismas, entre otras características.

A lo largo de los años, MPI se ha convertido en el estándar *de facto* para programas paralelos sobre sistemas de memoria distribuida y se lo suele utilizar en conjunto con otras herramientas para el desarrollo de aplicaciones que se ejecutan sobre clusters de nodos que integran aceleradores.

2.2.3.3. CUDA

La Arquitectura Unificada de Dispositivos de Cómputo (CUDA, por sus siglas en inglés) hace referencia tanto a un compilador como a un conjunto de herramientas desarrollados por NVIDIA para la programación de sus GPUs de propósito general. El primer Kit de Desarrollo de Software (SDK, por sus siglas en inglés) fue publicado en 2007 y, a pesar de ser una extensión propietaria, se ha convertido en el estándar *de facto* para programación de GPUs en HPC.

La Figura 2.18 ilustra un diagrama del modelo de ejecución y de la arquitectura de memoria CUDA. La ejecución de los programas acelerados con CUDA se basan en el modelo *host*-dispositivo, siendo el dispositivo en este caso la GPU. El *host* es el responsable de administrar la memoria del dispositivo y sus transferencias de datos, además de invocar la ejecución de los *kernels*.

Un *kernel* es un trozo de código que ejecutan miles de hilos primitivos en paralelo en la GPU. A este conjunto de hilos se los denomina *grid*. A su vez, los *grids* se organizan en *bloques*, donde todos tienen la misma dimensión y tamaño, los cuales son especificados por el programador.

Existen varios niveles de memoria en la GPU, cada uno con diferentes características de lectura y escritura. Cada hilo tiene acceso a registros propios así como también a una memoria privada llamada *memoria local*. Este nombre es poco apropiado ya que se ubica fuera del chip. También existe una *memoria compartida* unificada, la cual es accesible por todos los hilos de un mismo bloque durante su tiempo de vida. Por último, todos los hilos tienen acceso de lectura y escritura a una *memoria global*, la cual se ubica fuera del chip, en los módulo de memoria GDDR de la placa. También existen otros dos niveles de memoria de sólo lectura, llamados *constante* y *de textura*, en la misma ubicación que la memoria global [87].

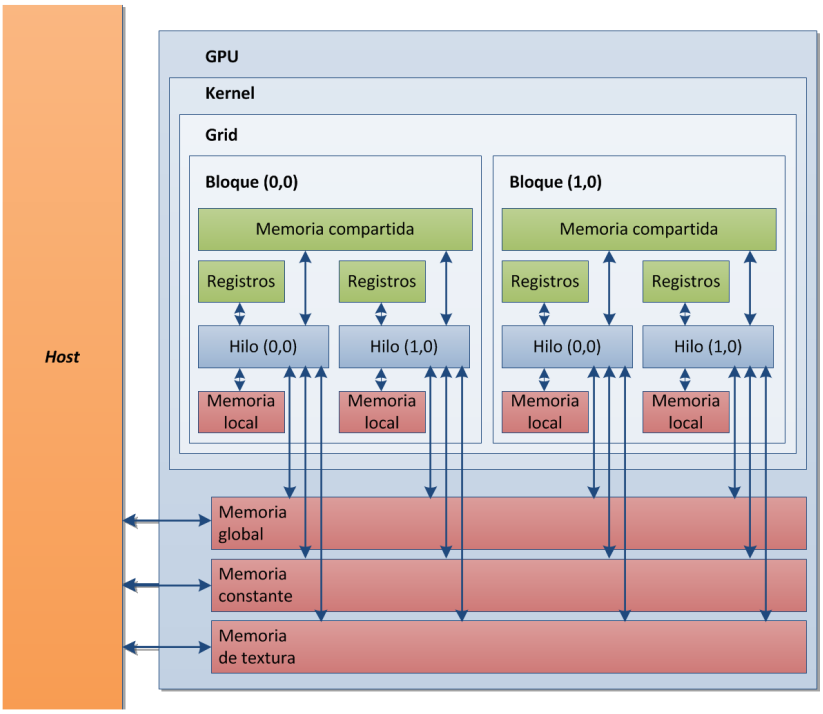

Figura 2.18: Diagrama del modelo de ejecución y arquitectura de memoria CUDA

2.2.3.4. OpenCL

OpenCL es un estándar de una interfaz para programación paralela multi-plataforma basada en el lenguaje C. Su diseño fue iniciado por Apple y estuvo motivado por la posibilidad de desarrollar aplicaciones paralelas portables en plataformas heterogéneas.

La Figura 2.19 ilustra un diagrama del modelo de ejecución y de la arquitectura de memoria OpenCL. Al igual que CUDA, OpenCL se basa en el modelo *host*-dispositivo. OpenCL modela un sistema de cómputo paralelo heterogéneo que integra un *host* y uno o más dispositivos OpenCL.

El modelo de programación de OpenCL divide la carga de trabajo del programa en un espacio multi-dimensional denominado *NDRange*, el cual se divide en *work-groups* y *work-items*. Cada *work-group* se compone de múltiples *work-items* y se ejecuta en forma independiente de los demás.

OpenCL soporta tanto paralelismo de datos como de tareas. El paralelismo de datos se suele explotar de manera SIMD, donde varios *work-items* son agrupados de acuerdo al ancho de la unidad vectorial del dispositivo subyacente. Por otra parte, para explotar paralelismo de tareas se emplea un *kernel* compuesto por un único *work-group* con un sólo *work-item*.

El modelo de memoria de OpenCL distingue diferentes regiones de memoria que se caracterizan por su tipo de acceso, alcance y rendimiento. La *memoria global* es una memoria de lectura y escritura accesible por todos los *work-items* de todos los *work-groups* y usualmente corresponde a la memoria DRAM del dispositivo, la cual involucra una alta latencia de acceso. La *memoria local* es una memoria de lectura y escritura accesible por todos los *work-items* de un mismo *work-group* y habitualmente tiene una baja latencia de acceso. La *memoria constante* es una memoria de sólo lectura visible para todos los *work-items* de todos

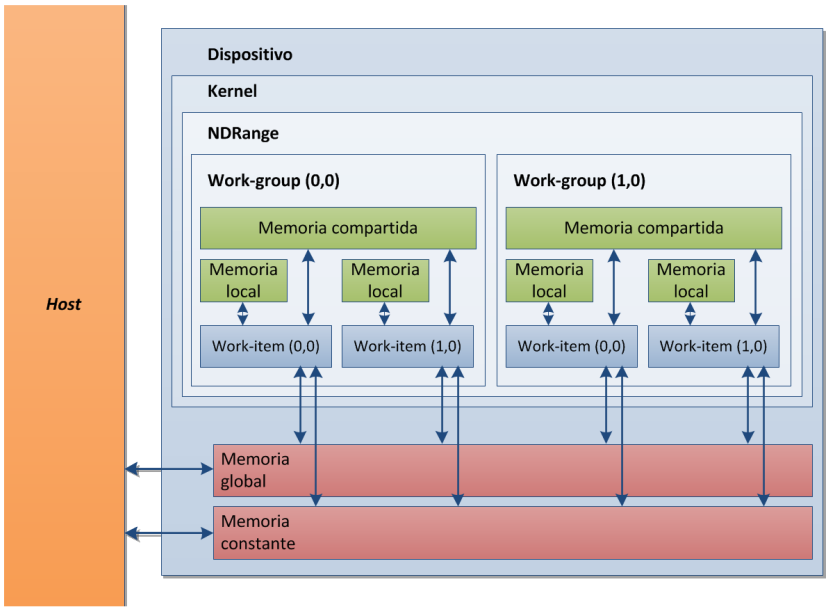

Figura 2.19: Diagrama del modelo de ejecución y arquitectura de memoria OpenCL

los *work-groups*. Por último, la *memoria privada*, como el nombre sugiere, es una memoria sólo accesible por cada *work-item*.

La especificación de OpenCL define un modelo de plataforma, de memoria y de programación, lo que permite a los vendedores proveer extensiones específicas de cada implementación. Existe una amplia libertad en términos de implementar la plataforma siempre y cuando la versión final satisfaga las especificaciones de OpenCL [88].

En la actualidad, OpenCL es mantenido por el Grupo Khronos y cuenta con soporte para múltiples dispositivos de hardware como CPUs, GPUs, DSPs, FPGAs, entre otros dispositivos. La última versión (2.0) fue publicada en 2014 e incluye actualizaciones para trabajar con memoria compartida virtual, paralelismo dinámico, *pipes*, entre otras características [89].

SDK de Altera para OpenCL En el año 2013, Altera presentó un SDK para OpenCL que respeta la especificación 1.0 para perfiles empotrados y que incluye varias de las características de las especificaciones de OpenCL 1.1 y 1.2 [90, 91, 92]. Altera sostiene que la utilización de OpenCL como herramienta de programación para sus FPGAs permite reducir el tiempo y el costo de desarrollo de las aplicaciones sobre estos dispositivos sin perder sus ventajas, como las tasas de rendimiento y bajo consumo [93].

En el modelo de plataforma de OpenCL, las FPGAs de Altera son aceleradores dedicados que se conectan al *host* mediante una interconexión periférica como puede ser el bus PCIe. En la Tabla 2.1 se muestra la asociación entre la compleja jerarquía de memoria de estos dispositivos y el modelo de memoria de OpenCL.

Cada FPGA de Altera puede tener asociada múltiples colas de comandos ordenadas capaces de ejecutar comandos en forma concurrente. Los *kernels* son compilados previamente por el compilador OpenCL de Altera y su contenido es pasado en ejecución para crear el programa objeto de OpenCL.

Con respecto al modelo de ejecución, la especificación de OpenCL no define en qué orden

Memoria OpenCL	Memoria FPGA
global	externa
constante	cache
local	embebida
privada	registros

Tabla 2.1: Modelo de memoria OpenCL para FPGAs

se ejecutan los *work-groups* de un *NDRange*, ni tampoco el de los *work-items* que componen los *work-groups*. En la implementación de Altera, tanto los *work-groups* como los *work-items* son ejecutados en forma secuencial y creciente de acuerdo a su identificación.

La implementación de Altera también proporciona una extensión específica para hacer uso de canales de E/S que se asemejan a los *pipes* de la especificación 2.0 de OpenCL [89]. Los canales de Altera representan un mecanismo eficiente de comunicación para *work-items* del mismo *kernel* o de diferentes *kernels* sin requerir intervención del *host*.

2.2.4. Rendimiento, consumo de potencia y costo de programación

La Tabla 2.2 presenta algunas características relacionadas con el rendimiento y el consumo de potencia de algunos modelos de las diferentes arquitecturas estudiadas. Para medir el rendimiento se ha usado la métrica FLOPS mientras que para la eficiencia energética se ha empleado la métrica FLOPS/Watt (considerando la potencia de diseño térmico como un valor representativo del consumo de potencia[3]). Resulta difícil realizar una comparación justa entre diferentes arquitecturas debido que a no existe un método estándar que lo permita. Sin embargo, la comparación sirve para mostrar tendencias y características distintivas de cada arquitectura, aun cuando existen diferentes modelos de cada una que puedan presentar variaciones en sus especificaciones.

		Intel Xeon E5-4669 v3	NVIDIA Tesla K40	Intel Xeon Phi 7120P	Xilinx Virtex7 XCTV2000T
Número de procesadores		18 núcleos (2 hilos hardware cada uno)	2880 núcleos CUDA	61 núcleos (4 hilos hardware cada uno)	2160 bloques DSP
Frecuencia del reloj (GHz)		2.1 - 2.9	0.745	1.238 - 1.333	< 0.741
Ancho SIMD		8	32	16	Configurable
Pico de FLOPS (GFLOPS)	**Precisión simple**	604.8	4290	2416	1636
	Precisión doble	302.4	1430	1208	671
TDP (Watt)		135	245	300	< 40
GFLOPS/Watt		4.48	17.51	8.05	> 40.9

Tabla 2.2: Características de rendimiento y consumo de potencia de algunos modelos de las diferentes arquitecturas estudiadas

El procesador Intel Xeon E5-4669 v3 posee 18 núcleos con tecnología Hyper-Threading

[3]La potencia de diseño térmico (TDP, por sus siglas en inglés) corresponde a la máxima cantidad de potencia que el sistema de refrigeración necesita disipar de manera de mantener al dispositivo debajo de su máxima temperatura. Aunque este valor no define la máxima cantidad de potencia que se podría disipar, es usualmente aceptado como un valor representativo del consumo de potencia en ciertos contextos [94].

(cada núcleo cuenta con dos hilos hardware). Como las unidades vectoriales de este procesador son capaces de ejecutar hasta 8 operaciones de punto flotante en precisión simple, el Xeon E5-4669 puede alcanzar 604.8 GFLOPS operando a una frecuencia de reloj de 2.1 GHz. En este caso, como el TDP es de 135 Watt, el Xeon E5-4669 reporta 4.48 GFLOPS/Watt.

La GPU Tesla K40 de NVIDIA cuenta con 2880 núcleos CUDA divididos en 15 SMs de 192 SPs cada uno. A una frecuencia de 0.745Ghz, la K40 alcanza 4290 GFLOPS en precisión simple y 1430 GFLOPS en precisión doble. Dado que el TDP es de 245 Watt, el cociente entre GFLOPS y Watt es de 17.51.

El coprocesador Intel Xeon Phi 7120P tiene 61 núcleos y, al igual que el Xeon E5-4669 v3, cuenta con tecnología Hyper-Threading (cada núcleo posee cuatro hilos hardware). Los núcleos del Xeon Phi poseen capacidades vectoriales extendidas, lo que le permiten ejecutar hasta 16 operaciones de punto flotante en precisión simple y la mitad en precisión doble. Este coprocesador tiene un rendimiento máximo de 2416 GFLOPS y como su TDP es de 300 Watt, logra un cociente de 8.05 GFLOPS/Watt.

La FPGA Virtex7 XCTV2000T de la empresa Xilinx puede configurar sus recursos para lograr un pico de rendimiento de 1636 GFLOPS en precisión simple y 671 en precisión doble [21]. Con un valor máximo de TDP de 40 Watt, su cociente supera los 40 FLOPS/Watt.

Como se puede apreciar a partir de la tabla, los aceleradores no sólo son capaces de lograr picos de rendimiento muy superiores a las CPUs sino también mejores cocientes FLOPS/Watt. Las GPUs suelen tener picos de rendimiento y tasas de eficiencia energética mayores a los coprocesadores Xeon Phi. Sin embargo, requieren del aprendizaje de lenguajes de programación específicos mientras que los coprocesadores Xeon Phi soportan modelos de programación tradicionales que favorecen el desarrollo y la portabilidad de los programas. Aunque las FPGAs se utilizaron tradicionalmente para problemas de punto fijo, la evolución en sus capacidades computacionales las convierten en una alternativa a tener en cuenta para problemas de cómputo intensivo en punto flotante. El bajo consumo de potencia hace a las FPGAs muy eficientes desde el punto de vista energético, superando ampliamente al resto de los aceleradores. Desafortunadamente su costo de programación también es mayor al resto de los dispositivos. Es por ello que en la actualidad tanto Altera como Xilinx trabajan en el desarrollo de herramientas que permitan disminuir el esfuerzo de programación de estos dispositivos.

Por último, resulta importante aclarar que el pico de rendimiento sólo permite una comparación teórica entre unidades de procesamiento. El rendimiento sostenible y la eficiencia energética varían de acuerdo a cada problema particular y a su correspondiente implementación en software debido a que cada unidad de procesamiento cuenta con diferentes características de hardware (incluyendo la arquitectura de memoria y la red de comunicación), las cuales pueden ser más apropiadas para algunas aplicaciones que para otras [21].

2.3. Aceleración del algoritmo SW

Como se explicó en la Sección 2.1.3, los algoritmos basados en programación dinámica resultan costosos computacionalmente para ser utilizados en búsquedas de similitud en bases de datos. De hecho, debido al crecimiento exponencial de la información biológica, aun los algoritmos heurísticos no resultan suficiente en ciertas ocasiones. Es aquí donde el empleo de paralelismo se vuelve fundamental para acelerar estas búsquedas. Esta sección se enfoca en el estado del arte de la aceleración del algoritmo SW. Primero se estudian las dependencias de datos del algoritmo SW y las formas posibles de paralelizarlo (Sección 2.3.1) para luego

describir el estado del arte de las implementaciones SW sobre diferentes plataformas de procesamiento paralelo: CPU, GPU, FPGA y Xeon Phi (Sección 2.3.2).

2.3.1. Dependencias de datos y paralelismo

La parte más costosa computacionalmente del algoritmo SW es el cálculo de la matriz de similitud. Resulta importante notar que las celdas de la matriz H no se pueden computar en cualquier orden debido a las dependencias de datos inherentes al problema. Para poder computar el valor de una celda es necesario haber calculado previamente tres valores: el de la celda vecina superior, el de la celda vecina izquierda y el de la celda vecina superior-izquierda, como se muestra en la Figura 2.20.

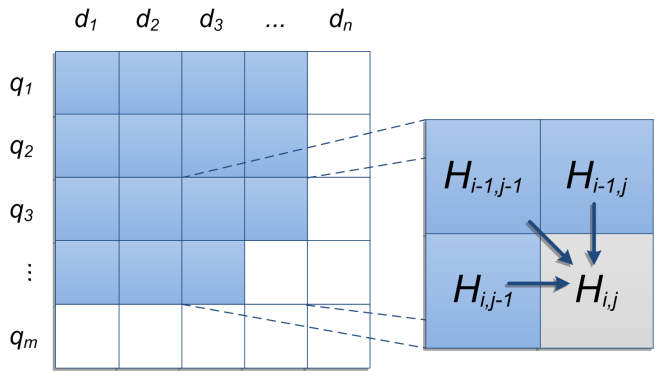

Figura 2.20: Dependencias de datos en la matriz H

Aunque estas dependencias restringen las formas en que se puede computar la matriz de similitud, el algoritmo SW posee cierto paralelismo inherente que puede ser explotado para reducir su costo computacional. En general, las implementaciones suelen emplear alguno de los dos siguientes esquemas:

- El esquema intra-tarea se basa en acelerar el cómputo de un único alineamiento por vez. Las implementaciones que adoptan este esquema suelen computar las celdas de cada antidiagonal de la matriz en paralelo, aprovechando que estos cálculos son independientes entre sí. También es posible computar varias celdas de una fila o de una columna simultáneamente; sin embargo, como este enfoque ignora las dependencias de datos antes mencionadas, se requiere de un ajuste posterior de los valores de las celdas para mantener la corrección del algoritmo.

- El esquema inter-tarea se basa en acelerar el cómputo de múltiples alineamientos al mismo tiempo. La mayor ventaja de este enfoque es la independencia en el cálculo de cada alineamiento.

Ambos enfoques han sido explorados por la comunidad científica en una amplia variedad de trabajos y dispositivos que se describen en las siguientes secciones.

2.3.2. Implementaciones existentes

En esta sección se aborda el estado del arte relativo a la aceleración del algoritmo SW sobre diferentes arquitecturas de hardware. Sin embargo, para poder evaluar el desempeño de una implementación en forma independiente de las secuencias de consulta y de las bases

de datos utilizadas para las diferentes pruebas al igual que de la arquitectura de soporte, se emplea la métrica de rendimiento *Cell Updates Per Second* (CUPS). Un CUPS representa el tiempo requerido para computar completamente una celda de la matriz de similitud H e incluye los cálculos correspondientes a los valores en los arreglos E y F. Dada una secuencia de consulta Q y una base de datos D, el valor de GCUPS (mil millones de CUPS) se calcula como:

$$\frac{|Q| \times |D|}{t \times 10^9} \tag{2.4}$$

donde $|Q|$ es el número de total de símbolos en la secuencia de consulta, $|D|$ es el número total de símbolos en la base de datos y t es el tiempo de ejecución en segundos [95].

A continuación, se describen las implementaciones más destacadas de acuerdo a la arquitectura de soporte empleada.

2.3.2.1. Implementaciones en CPU

Los primeros esfuerzos para acelerar el algoritmo SW se remontan a la década del '90. Alpern *et al.* [8] propusieron varias técnicas incluyendo una implementación paralela que utiliza microparalelismo para dividir en cuatro partes a los registros Z-buffer de 64 bits de los procesadores Intel Paragon i860. Mediante este esquema se pudo comparar la secuencia de consulta con cuatro secuencias de la base de datos al mismo tiempo. Como resultado, obtuvieron una aceleración de 5× comparado a una implementación convencional.

El paralelismo intra-registro de la propuesta anterior sería luego más simple y fácil de utilizar con la introducción de pequeñas capacidades vectoriales por parte de las empresas fabricantes de procesadores. Con el auge de las aplicaciones multimedia, los procesadores de propósito general incorporaron tecnología SIMD, entre las que destacan las extensiones MMX, SSE, AVX o AltiVec por mencionar a algunas de ellas [4]. En general, se pueden identificar diferentes enfoques para la utilización de vectorización en el cómputo de las matrices de similitud, los cuales se ilustran en la Figura 2.21.

La mayoría de los esfuerzos se han concentrado en el enfoque intra-tarea. En el año 1997, Wozniak [5] presentó una implementación paralela para un procesador Sun Ultra Sparc que aprovechaba las instrucciones SIMD de video para computar varias celdas de cada antidiagonal de la matriz en paralelo, ya que éstas son independientes entre sí. Desafortunadamente, la obtención de los puntajes de sustitución para las celdas de la diagonal resulta complicado y difícil de resolver en forma eficiente, ya que cada puntaje de sustitución requiere indexación con diferentes pares de residuos de ambas secuencias. Aún así, Wozniak logró una aceleración de 2× sobre la mejor implementación secuencial de esa época.

Rognes y Seeberg [6] presentaron en el año 2000 una versión SIMD del algoritmo SW utilizando las extensiones MMX/SSE, siendo los primeros en aprovechar estos conjunto de instrucciones. Ellos encontraron que vectorizar a lo largo de la secuencia de consulta resulta más rápido en comparación a hacerlo por las antidiagonales, a pesar de tener que realizar más cálculos por ignorar algunas de las dependencias de datos indicadas en la Sección 2.3.1. Otra diferencia con enfoques anteriores radica en que Rognes y Seeberg usaron enteros de 8 bits. De esta manera es posible procesar un mayor número de alineamientos al mismo tiempo; en particular, 8 y 16 al emplear las instrucciones MMX y SSE, respectivamente. Aunque el rango de representación de puntajes se reduce a 0-255, éste resulta suficiente para la mayoría de los alineamientos ya que el *overflow* sólo ocurre cuando se alinean secuencias muy similares y/o muy largas (en esos casos se emplea otra versión del algoritmo con enteros de mayor rango para garantizar un resultado correcto). Además, Rognes y Seeberg fueron

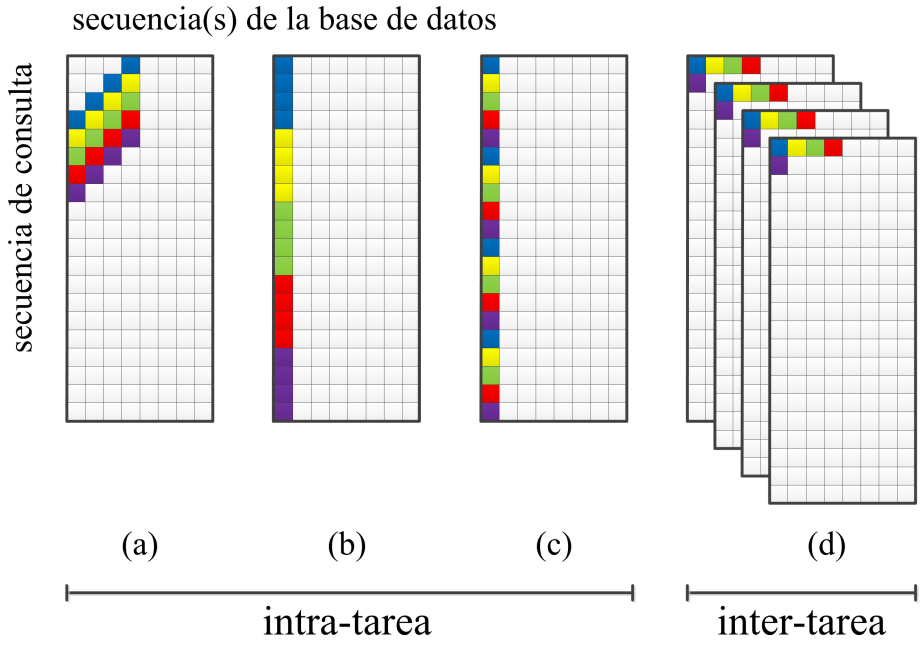

Figura 2.21: Enfoques para paralelización SIMD del algoritmo Smith-Waterman (adaptado de [4]). (a) Vectorización en las anti-diagonales, propuesta por Wozniak *et al* [5]. (b) Vectorización en la secuencia de consulta, propuesto por Rognes y Seeberg [6]. (c) Vectorización por franjas en la secuencia de consulta, propuesto por Farrar [7]. (d) Vectorización en múltiples secuencias de la base de datos, propuesto por Alpern *et al* [8] y posteriormente por Rognes [4].

los primeros en introducir la técnica *Query Profile* (QP), la cual consiste en construir una matriz auxiliar de tamaño $|q| \times |\sum|$, donde q en la secuencia de consulta y $\sum$ es el alfabeto. Cada fila de esta matriz contiene los puntajes del residuo correspondiente de la secuencia de consulta para todos los posibles residuos del alfabeto. La Figura 2.22 ilustra un ejemplo de construcción de la matriz QP y su aplicación al alineamiento entre las secuencias RNNCRA y ANRACD junto a una versión reducida de la matriz de sustitución BLOSUM62. Usando QP se mejora la localidad de los datos ya que se reemplaza un acceso aleatorio por un acceso lineal al momento de obtener los puntajes de sustitución en el bucle más interno del cómputo de la matriz de similitud. El enfoque de vectorización a lo largo de la secuencia de consulta empleando enteros de bajo rango en conjunto con la técnica QP y un uso eficiente de la caché, le permitió a la implementación de Rognes y Seeberg lograr una aceleración de 6× sobre una implementación secuencial altamente optimizada.

Posteriormente, en el año 2007, Farrar [7] utilizó las instrucciones SSE2 para desarrollar una solución mejorada del enfoque de Rognes y Seeberg. Al igual que la propuesta anterior, se utiliza vectorización a lo largo de la secuencia de consulta. Sin embargo, el cómputo de la matriz de similitud se reorganiza para hacerlo en franjas. Este patrón de acceso permite evitar accesos desalineados a los vectores y minimizar el impacto de las dependencias de datos. Además, Farrar también propuso la utilización de la técnica *lazy F*, la cual ayuda a minimizar el número de saltos condicionales dentro del bucle más interno del código. Como resultado de todas estas mejoras, Farrar reportó rendimientos de más de 11 y 20 GCUPS utilizando cuatro y ocho núcleos, respectivamente [96].

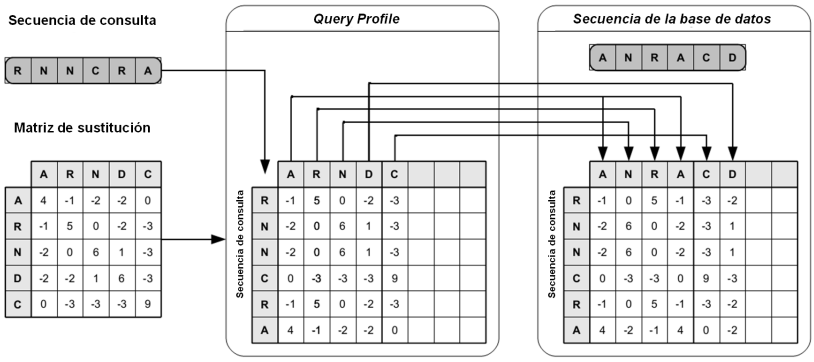

Figura 2.22: Ejemplo de construcción del *Query Profile* y su aplicación al alineamiento entre las secuencias RNNCRA y ANRACD junto a una versión reducida de la matriz de sustitución BLOSUM62 (adaptado de [9]).

Un año más tarde, Szalkowski *et al.* [97] propusieron algunas mejores al enfoque de Farrar presentando una implementación multihilada tanto para arquitecturas x86 con conjunto de instrucciones SSE2 como para la arquitectura Cell/BE de IBM. A esta implementación se la denominó SWPS3 y logró un pico de rendimiento de 15.7 GCUPS utilizando un procesador de cuatro núcleos.

Más adelante en el tiempo, en el año 2011, Rognes [4] presentó una implementación para procesadores Intel con conjunto de instrucciones SSSE3 empleando el enfoque inter-tarea propuesto anteriormente por Alpern *et al.* [8], a la cual llamó SWIPE. Para acelerar la obtención de los puntajes de sustitución, Rognes propuso la técnica *Score Profile* (SP). La técnica SP se basa en construir un arreglo auxiliar de puntajes de tamaño $s \times n \times |\sum|$, donde s es el número de secuencias de la base de datos que se procesan a la vez, n representa la longitud máxima de las secuencias de la base de datos y $\sum$ es el alfabeto. Como cada fila del SP forma un vector de puntajes de s elementos, sus valores pueden ser recuperados utilizando una única operación vectorial *load*. La Figura 2.23 ilustra un ejemplo de construcción de SP dadas 16 secuencias de cuatro residuos cada una. La desventaja de SP radica en que debe ser construido para cada conjunto de secuencias de la base de datos que se procesan en paralelo. Utilizando dos procesadores de seis núcleos, SWIPE alcanzó un pico de rendimiento de 106 GCUPS, siendo hasta seis veces más rápido que SWPS3 y que una implementación propia de Rognes del enfoque de Farrar. Hasta el momento, se considera a SWIPE como la implementación SW más rápida para el conjunto de instrucciones SSE.

Por último, en el año 2013, Zhao *et al.* presentaron SSW [98], una librería desarrollada en C/C++ para facilitar la integración del algoritmo SW en otras aplicaciones genómicas. SSW implementa el enfoque de Farrar tanto para retornar el alineamiento como el puntaje óptimo. Como SSW también se encuentra disponible en forma de herramienta autónoma, los autores evaluaron su rendimiento al realizar búsquedas en la base de datos Swiss-Prot; SSW reportó hasta 2.53 GCUPS sobre un procesador AMD x86 64 2.0Ghz.

La Tabla 2.3 resume el rendimiento de las implementaciones descritas.

2.3.2.2. Implementaciones en GPU

Las primeras implementaciones en GPU fueron las de Weiguo Liu *et al.* [99] y Yang Liu *et al.* [100] en el año 2006. Ambas propuestas comparten muchas características; entre ellas

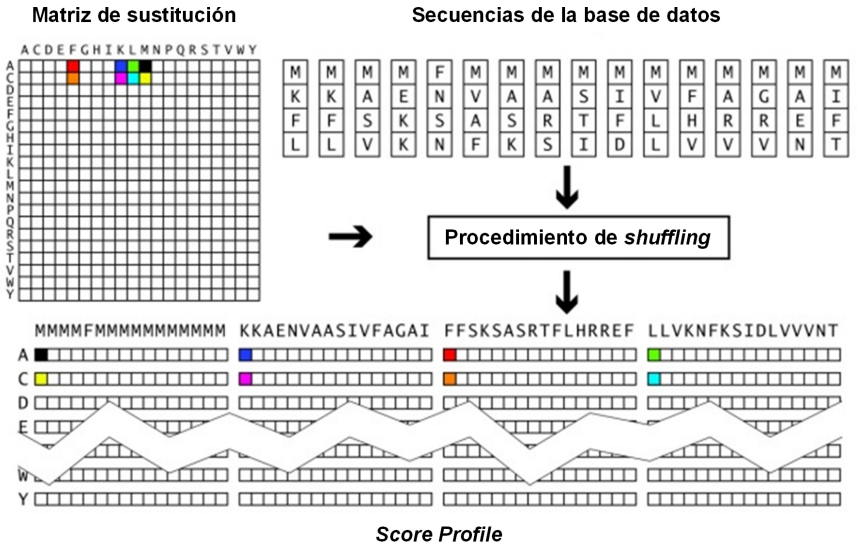

Figura 2.23: Ejemplo de construcción del *Score Profile* dadas 16 secuencias de cuatro residuos cada una (adaptado de [4]).

que fueron programadas con OpenGL, que las matrices de alineamiento se computan por las antidiagonales y que tanto las secuencias como los *buffers* auxiliares son almacenados en memoria de textura. La implementación de Weiguo Liu *et al.* sólo es capaz de procesar secuencias de longitud inferior a 4096 residuos debido a limitaciones impuestas por la capacidad de la memoria de textura de las GPUs de ese momento y, por ese motivo, los experimentos se realizaron con una versión reducida de la base de datos Swiss-Prot (99,8 % de las secuencias originales). Utilizando una placa NVIDIA GeForce 6800 GT, esta implementación alcanzó 0.65 GCUPS, lo que representó una aceleración de aproximadamente 16× sobre una implementación en CPU. Por su parte, la propuesta de Yang Liu *et al.* no impone restricciones en la longitud de las secuencias y ofrece dos modos de ejecución: (1) computando el alineamiento óptimo y (2) computando sólo el puntaje óptimo. Empleando una placa NVIDIA GeForce 7800 GTX, esta implementación reportó 0.18 y 0.24 GCUPS en los modos (1) y (2), respectivamente.

En el año 2008, Manavski y Valle presentaron la primera implementación SW que empleó CUDA [101], a la cual denominaron SW-CUDA. A diferencia de las propuestas anteriores, SW-CUDA emplea el enfoque inter-tarea, ya que cada hilo CUDA computa un alineamiento completo entre la secuencia de consulta y una secuencia de la base de datos. Otra característica distintiva de SW-CUDA es el empleo de la técnica QP, la cual permite reemplazar accesos aleatorios a la matriz de sustitución por accesos secuenciales a la matriz QP. SW-CUDA reportó 1.85 GCUPS sobre una placa NVIDIA GeForce 8800 GTX al realizar búsquedas en la base de datos Swiss-Prot. Además, demostró buena escalabilidad con respecto al número de aceleradores gracias a una técnica de balance de carga dinámica que considera las capacidades de cómputo de cada dispositivo para distribuir el trabajo.

Yongchao Liu *et al.* presentaron CUDASW++ en el año 2009 [95], una implementación para GPUs compatibles con CUDA que combina los enfoques intra-tarea e inter-tarea. El esquema inter-tarea suele reportar mejores rendimientos que el intra-tarea; sin embargo, requiere de un mayor consumo de recursos de memoria. Por ese motivo, CUDASW++

Implementación	Hardware de soporte	Número de hilos	GCUPS	
			Total	Por hilo
Wozniak [5] (1997)	Sun Ultra Sparc Enterprise 6000 167 MHz	12	0.2	0.017
Rognes y Seeberg [6] (2000)	Intel Pentium III 500 MHz	1	0.15	0.15
Farrar [7] (2007)	Intel Xeon Core 2 Duo 2.0 GHz	1	2.9	2.9
SWPS3, Szalkowski *et al.* [97] (2008)	Intel Core 2 Quad Q6600 2.4 GHz	4	15.07	3.77
SWIPE, Rognes [4] (2011)	2 × Intel Xeon X5650 2.67 GHz	24	106	4.42
SSW, Zhao *et al.* [98] (2013)	AMD x86 64 2.0Ghz	1	2.53	2.53

Tabla 2.3: Resumen de rendimiento de las implementaciones en CPU descritas

establece un umbral de longitud configurable para el procesamiento de los alineamientos. Las secuencias de la base de datos de longitud menor al umbral son computadas según el esquema inter-tarea mientras que para el resto se emplea el intra-tarea. CUDASW++ también se caracteriza por una cuidadosa organización de los accesos de memoria para aprovechar acceso coalescente y por la explotación de memoria constante para las secuencias de consulta y de memoria compartida para la matriz de sustitución. Este conjunto de optimizaciones le permitió a CUDASW++ alcanzar 9.63 y 16.09 GCUPS al procesar las base de datos Swiss-Prot sobre una GPU NVIDIA GeForce GTX 280 y una GPU dual NVIDIA GeForce GTX 295, respectivamente.

Un año mas tarde, los mismos autores de CUDASW++ presentaron una versión mejorada de este software, conocida como CUDASW++ 2.0 [102]. Esta versión ofrece dos modos de ejecución: el primero emplea el modelo original pero incorporando la técnica QP para la obtención de los puntajes de sustitución y el uso de datos empaquetados en lugar de escalares; el segundo implementa el enfoque de Farrar [7] a través de la virtualización de instrucciones SIMD en las placas gráficas. Mediante búsquedas en la base de datos Swiss-Prot, CUDASW++ 2.0 reportó rendimientos muy similares entre ambos modos de ejecución, alcanzando 16.9 y 29.6 GCUPS sobre una GPU NVIDIA GeForce GTX 280 y una GPU dual NVIDIA GeForce GTX 295, respectivamente.

GASW es otra herramienta para búsquedas SW en GPUs que soportan CUDA [42]. Este software fue presentado en el año 2010 y propone varias optimizaciones, entre las que se destacan la eliminación de cuellos de botella en la memoria y la conversión de la base de datos a un formato más conveniente para el procesamiento de las placas gráficas. Utilizando la base de datos Swiss-Prot y una GPU NVIDIA GeForce GTX 275, GASW es capaz de lograr hasta 21.36 GCUPS.

Zou *et al.* [103] presentaron en el año 2011 una implementación basada en CUDA que

combina diferentes optimizaciones: accesos a memoria global en forma coalescente, explotación de todos los niveles de memoria y desenrollado de bucles. Empleando una placa NVIDIA GeForce GTX 470 y una base de datos sintética de 107520 secuencias (cada secuencia se conforma de 1024 residuos aleatorios), Zou *et al.* reportaron 28.35 GCUPS.

Existen pocas implementaciones conocidas para búsquedas de similitud en GPUs utilizando OpenCL; entre ellas se encuentran la de Khalafallah *et al.* [104] y la de Borovska y Lazarova [105]. La propuesta de Khalafallah *et al.* emplea el esquema inter-tarea y reutiliza varias optimizaciones de otras propuestas como la organización de los accesos a memoria para aprovechar los beneficios del acceso coalescente y el uso de memoria de textura para la matriz QP. Este trabajo reportó 12.29 y 65.99 GCUPS al realizar búsquedas en una versión limitada de la base de datos Swiss-Prot sobre las unidades gráficas NVIDIA GeForce 9800 GT y ATI HD 5850, respectivamente. Por su parte, la propuesta de Borovska y Lazarova también emplea el enfoque inter-tarea aunque especifica muy pocos detalles de implementación. Esta propuesta alcanza aceleraciones de 1.6 y 7.8 GCUPS al procesar la base de datos Swiss-Prot utilizando las placas de NVIDIA Quadro FX3600M y GeForce GTX 295, respectivamente.

Por último, en el año 2013, Yongchao Liu *et al.* presentaron una tercera versión de CUDASW++, a la que llamaron CUDASW++ 3.0 [106]. Esta implementación combina cómputo concurrente en la CPU y en la GPU empleando el esquema inter-tarea en ambos casos. La carga de trabajo se distribuye dinámicamente utilizando una heurística basada en las características del hardware de soporte y en constantes derivadas de evaluaciones empíricas. Para el cómputo en la GPU se utilizan las instrucciones de video CUDA PTX mientras que para el cómputo en la CPU se usan las instrucciones SSE; en particular el código en la CPU se basa en el de SWIPE. Tanto en la CPU como en la GPU los alineamientos son computados en primer lugar utilizando enteros de 8 bits. Una vez que todos los alineamientos fueron procesados, la CPU verifica los casos en que pudo haber ocurrido *overflow* y recomputa esos alineamientos usando enteros de 16 bits. Es importante mencionar que, como la GPU emplea el esquema inter-tarea, sólo puede procesar aquellas secuencias de la base de datos que están por debajo del umbral de longitud, por lo que las secuencias más largas son procesadas obligatoriamente en la CPU. Con Swiss-Prot como base de datos de prueba, CUDASW++ 3.0 es capaz de alcanzar 119 GCUPS utilizando una computadora personal basada en procesador *quad-core* Intel i7 2700k 3.5Ghz y una placa gráfica NVIDIA GeForce GTX 680. Al día de hoy, se considera a CUDASW++ 3.0 la implementación SW más rápida para sistemas basados en GPUs compatibles con CUDA.

La Tabla 2.4 resume el rendimiento de las implementaciones descritas.

2.3.2.3. Implementaciones en FPGA

El empleo de FPGAs para acelerar el alineamiento de secuencias biológicas es un tema ampliamente estudiado en la comunidad HPC. La mayoría de las propuestas se enfocan en el alineamiento de secuencias de ADN ya que este caso resulta más sencillo que el alineamiento de secuencias de proteínas desde el punto de vista algorítmico, por su alfabeto reducido y su esquema de puntuación simple.

Más allá del tipo de secuencia que procesen, las implementaciones en FPGA se basan en crear bloques básicos que sean capaces de computar el valor de una celda de una matriz en cada ciclo de reloj. Luego, múltiples instancias de estos bloques son combinados a la vez para formar arreglos sistólicos que puedan procesar grandes cantidades de datos en paralelo. Un arreglo sistólico en una organización de unidades de procesamiento en forma de arreglo,

Implementación	Modelo de programación	Hardware de soporte	Base de datos	GCUPS
Weiguo Liu *et al.* [99](2006)	OpenGL	NVIDIA GeForce 6800 GT	Swiss-Prot (99.8 %)	0.65
Yang Liu *et al.* [100] (2006)	OpenGL	NVIDIA GeForce 7800 GTX	462862 aminoácidos en 983 secuencias	0.24
SW-CUDA, Manavski y Valle [101] (2008)	CUDA	NVIDIA GeForce 8800 GTX, 2×NVIDIA GeForce 8800 GTX	Swiss-Prot	1.85, 3.61
CUDASW++, Yongchao Liu *et al.* [95] (2009)	CUDA	NVIDIA GeForce GTX 280, NVIDIA GeForce GTX 295	Swiss-Prot	9.63, 16.09
CUDASW++ 2.0, Yongchao Liu *et al.* [102] (2010)	CUDA	NVIDIA GeForce GTX 280, NVIDIA GeForce GTX 295	Swiss-Prot	16.9, 29.6
GASW, Kentie [42] (2010)	CUDA	NVIDIA GeForce GTX 275	Swiss-Prot	21.36
Zou *et al.* [103] (2011)	CUDA	NVIDIA GeForce GTX 280, NVIDIA GeForce GTX 470	107520 secuencias de 1024 residuos	13.71, 28.35
Khalafallah *et al.* [104] (2010)	OpenCL	NVIDIA GeForce 9800 GT, ATI HD 5850	Swiss-Prot (45 %)	12.29, 65.99
Borovska y Lazarova [105] (2011)	OpenCL	NVIDIA Quadro FX3600M, NVIDIA GeForce GTX 275	Swiss-Prot	1.6, 7.8
CUDASW++ 3.0, Yongchao Liu *et al.* [106] (2013)	Pthreads + CUDA	Intel i7 2700k 3.5Ghz + NVIDIA GeForce GTX 680, Intel i7 2700k 3.5Ghz + NVIDIA GeForce GTX 690	Swiss-Prot	119, 185.6

Tabla 2.4: Resumen de rendimiento de las implementaciones en GPU descritas

donde los datos fluyen sincrónicamente entre las unidades, usualmente en una dirección específica. Este tipo de arreglos opera como las unidades vectoriales de las CPU modernas (por ejemplo las SSE) pero, a diferencia de ellas, pueden configurar la cantidad de elementos que procesan a la vez [45].

Desafortunadamente se hace difícil realizar una comparación analítica que resulte justa debido a diferentes causas [107, 108]:

- Existe una amplia variedad en tipos de FPGAs y cada una implementa sus circuitos de forma diferente, lo que dificulta una comparación directa.
- Existen muy pocas implementaciones que sean completas desde el punto de vista funcional. Las implementaciones que reportan los mejores resultados suelen contemplar pruebas sintéticas con el objetivo de mostrar la viabilidad del empleo de esta clase de aceleradores aunque su uso potencial en el mundo real es limitado. Algunas implementaciones sí utilizan datos reales pero presentan una o más limitaciones: la secuencia de consulta está embebida en el diseño, la secuencia de consulta tiene un tamaño fijo o un

tamaño máximo, los parámetros de la búsqueda (penalizaciones por *gaps* o matriz de sustitución) no se pueden cambiar o requieren reconfigurar el dispositivo para hacerlo, entre otros.

- Falta de documentación en los detalles de implementación. Por ejemplo, el rendimiento alcanzado por cada implementación depende en gran parte del ancho del tipo de dato utilizado. En general, el rendimiento mejora a medida que el tamaño se reduce, aunque un tamaño muy pequeño puede ser insuficiente para procesar todos los alineamientos. Desafortunadamente, no todas las implementaciones reportan el tamaño de datos utilizado.

De todas maneras, se resumen características de implementaciones conocidas para el alineamiento de secuencias de proteínas basadas en FPGAs, las cuales se muestran en la Tabla 2.5.

Implementación	**Lenguaje de programación**	**Hardware de soporte**	**GCUPS**	**Observaciones**
Dydel y Bala [108] (2004)	VHDL	Xilinx Virtex II XC2VP70-5	11.18	Pruebas sintéticas. Tamaños fijos de secuencias.
Oliver *et al.* [109] (2005)	Verilog	Xilinx Virtex II XC2V6000	10.6	Secuencia de consulta de tamaño limitado. Requiere reconfiguración para cambiar parámetros.
Zhang *et al.* [110] (2007)	No especifica	Altera Stratix II EP25180	25.6	Requiere reconfiguración para cambiar parámetros.
Van Court y Herbordt [111] (2007)	VHDL	Xilinx Virtex II Pro XC2VP70–5	5.41	Secuencias de tamaño limitado. Requiere reconfiguración para cambiar parámetros.
Benkrid *et al.* [112] (2009)	Handel-C	Xilinx Virtex II XC2V6000-4	7.66*	Requiere reconfiguración para cambiar parámetros. *El rendimiento indicado es teórico. Aunque realiza pruebas reales no brinda datos suficientes para calcular GCUPS correspondientes.
Isa *et al.* [113] (2011)	No especifica	Xilinx Virtex-5 XC5VLX110	28	
Zou *et al.* [103] (2011)	No especifica	Xilinx Virtex-5 XC5VLX330	47	Pruebas sintéticas.
Benkrid *et al.* [114] (2012)	Handel-C	Xilinx Virtex-4 LX160-11	19.4	

Table 2.5: Resumen de implementaciones en FPGA.

2.3.2.4. Implementaciones en Xeon Phi

Los coprocesadores Xeon Phi también pueden ser empleados para acelerar el alineamiento de secuencias biológicas. En el año 2014, Liu y Schmidt presentaron la herramienta SWAPHI [39] para búsquedas de similitud basada en OpenMP que emplea el modelo *offload* y que es capaz de aprovechar múltiples coprocesadores a la vez. En ese trabajo, los autores exploraron los esquemas de paralelismo intra-tarea e inter-tarea para el cálculo de las matrices de

alineamiento al igual que las técnicas QP y SP para la obtención de los puntajes de sustitución. En particular, SWAPHI es capaz de procesar 16 celdas de una matriz (o de diferentes matrices) al mismo tiempo gracias al empleo de las instrucciones KNC. Utilizando la base de datos TrEMBL y un coprocesador Xeon Phi 5110P, SWAPHI reportó un rendimiento máximo de 45.6 y 58.8 GCUPS con los esquemas intra-tarea e inter-tarea, respectivamente. Al emplear cuatro coprocesadores, el rendimiento aumentó a 164.9 (intra-tarea) y 228.4 GCUPS (inter-tarea) para la misma base de datos.

XSW es otra herramienta para búsquedas de similitud sobre coprocesadores Xeon Phi publicada en el año 2014 [40]. Al igual que SWAPHI, XSW emplea el esquema inter-tarea, la técnica SP y las instrucciones KNC para acelerar el cómputo de los alineamientos. A diferencia de SWAPHI, XSW se ejecuta en modo nativo con Pthreads como modelo de programación. XSW reportó hasta 70 GCUPS al buscar en la base de datos Environmental NR sobre un Xeon Phi 7110P.

Una versión extendida de XSW, conocida como XSW 2.0, fue desarrollada posteriormente por los mismos autores [115]. Esta implementación adopta el modelo *offload* para combinar cómputo concurrente en la CPU y la posibilidad de explotar más de un coprocesador al mismo tiempo. En un sistema compuesto por un procesador Xeon E5-2620 y un coprocesador Xeon Phi 7110P, XSW 2.0 alcanzó hasta 100 GCUPS al procesar la base de datos Environmental NR.

La Tabla 2.6 resume el rendimiento de las implementaciones descritas.

Implementación	**Modelo de programación**	**Hardware de soporte**	**Base de datos**	**GCUPS**
SWAPHI, Liu & Schmidt [39](2014)	OpenMP (*offload*)	Xeon Phi 5110P, 4×Xeon Phi 5110P	TrEMBL	58.8, 228.4
XSW, Wang *et al.* [40] (2014)	Pthreads (nativo)	Xeon Phi 7110P	Environmental NR	70
XSW 2.0, Wang *et al.* [115] (2014)	Pthreads (*offload*)	Xeon E5-2620 + Xeon Phi 7110P	Environmental NR	100

Tabla 2.6: Resumen de rendimiento de las implementaciones en Xeon Phi descritas

2.4. Resumen

En la actualidad, la bioinformática es un área de investigación interdisciplinaria que actúa como puente entre diferentes campos de las ciencias, como la biología, la bioquímica, la informática, la matemática, la física y la estadística. Su objetivo final consiste en descubrir la información importante que se encuentra oculta dentro de enormes volúmenes de datos y obtener una visión más clara de la biología de los organismos.

Una operación fundamental en cualquier investigación biológica es el alineamiento de secuencias, la cual consiste en comparar dos o más secuencias biológicas. El propósito de esta operación es determinar las regiones de similitud entre las secuencias de forma de identificar las relaciones funcionales, estructurales y evolutivas que existen entre ellas. El alineamiento de secuencias resulta esencial en el análisis filogenético, el perfilado de enfermedades genéticas, la identificación y cuantificación de regiones conservadas o unidades funcionales, y el perfilado y predicción de secuencias ancestrales. También resulta importante en el desarrollo de nuevas drogas y en la investigación forense criminal, y es por ello que actualmente la

comunidad científica realiza un gran esfuerzo en este campo de la bioinformática.

Las secuencias se representan simbólicamente mediante una sucesión de letras mayúsculas. Por su parte, los algoritmos de alineamiento operan directamente sobre las letras buscando emparejarlas de forma tal que el grado de similitud sea maximizado. Como múltiples alineamientos son posibles entre dos secuencias, se requiere de alguna técnica de clasificación que permita determinar objetivamente el mejor alineamiento para cualquier par de secuencias. El esquema de penalización de *gap* por afinidad suele ser el más utilizado por su mayor sentido biológico. Aunque este esquema es el mismo para los alineamientos de secuencias de proteínas como los de secuencias de ADN, se considera al caso de las proteínas más complejo por su alfabeto más grande y el empleo de matrices de sustitución en lugar de valores fijos.

Los algoritmos para alineamientos de secuencias se pueden clasificar de acuerdo a dos propiedades. La primera propiedad considera si el algoritmo produce un alineamiento global o uno local. Los alineamientos globales calculan la similitud considerando toda la longitud de las secuencias mientras que los alineamientos locales consideran la similitud de pequeñas regiones. Como los segundos permiten exponer de mejor manera las similitudes entre las secuencias, suelen conducir a resultados con más sentido biológico. La segunda propiedad clasifica a los algoritmos de acuerdo a la metodología de resolución que emplee. Los algoritmos basados en programación dinámica siempre encuentran la solución óptima. Desafortunadamente, deben examinar todas las formas posibles de resolver un problema, lo que suele ser computacionalmente costoso. En sentido contrario, los algoritmos basados en heurísticas encuentran la solución de forma más rápida pero a expensas de una reducción en la precisión de los resultados.

El algoritmo SW es un método para alineamientos locales de secuencias propuesto por Temple Smith y Michael Waterman en 1981. Este algoritmo es un método popular ya que ha sido utilizado como base para otros algoritmos posteriores y como patrón con el cual comparar otras técnicas de alineamiento. Al estar basado en programación dinámica, resulta costoso computacionalmente (complejidad temporal cuadrática) ya que garantiza encontrar el alineamiento óptimo entre las secuencias. Este alineamiento es justamente lo que le interesa a los investigadores. Cuando un investigador tiene una porción de secuencia biológica, usualmente la alinea contra una base de datos de secuencias conocidas, lo que se conoce como búsqueda biológica, búsqueda de similitud o también búsqueda SW. Aprovechando que el alineamiento óptimo sólo tiene valor para las secuencias de mayor puntaje, en primer lugar se suele calcular sólo los puntajes de alineamiento entre la secuencia de consulta y todas las secuencias de la base de datos. Luego, cuando resulta necesario, se puede calcular los alineamientos óptimos entre la secuencia de consulta y las secuencias de la base de datos más parecidas. De esta forma, se reducen los requerimientos de memoria de las búsquedas que, a su vez, impactan también en el tiempo de ejecución requerido.

La utilidad de las búsquedas de similitud depende en gran parte de la calidad y la cantidad de secuencias que integran las bases de datos biológicas. Es por eso que los tres proveedores de datos biológicas más grandes del mundo conformaron un proyecto colaborativo para que todos tengan disponible la mayor información posible. Entre las bases de datos de proteínas más populares se encuentran UniprotKB y NR. UniprotKB se divide en dos secciones: Swiss-Prot y TrEMBL. En la primera, las secuencias son revisadas y anotadas manualmente, mientras que en la segunda las secuencias son anotadas en forma automática mediante métodos computacionales. A pesar de contar con un número mucho menor de secuencias, se considera a Swiss-Prot la base de datos de referencia por su calidad. Por otro lado, NR compila las proteínas de varias bases de datos y se caracteriza por no tener secuencias duplicadas. Environmental NR es una variante de NR compuesta por secuen-

cias de proteínas traducidas que provienen directamente del ambiente, por lo que todos los organismos se encuentran mezclados.

La bioinformática no hubiera sido posible sin el progreso de las tecnologías informáticas, las cuales permiten almacenar, manipular y analizar enormes cantidades de información de manera cada vez más eficiente. Uno de los logros más importantes de la informática en los últimos años ha sido el desarrollo de los procesadores *multicore*. Durante casi 50 años, la mejora de rendimiento de los procesadores estuvo guiada por el aumento tanto en el número de transistores en el chip como en la frecuencia de reloj, lo que permitió la implementación de técnicas avanzadas de procesamiento como *pipelining*, unidades vectoriales y ejecución superescalar, entre otras. Sin embargo, a principios de la década del 2000, este proceso de mejora llegó a su límite debido a la dificultad de extraer más ILP de los programas pero principalmente por los niveles insostenibles de consumo de potencia y generación de calor que alcanzaban los procesadores. En consecuencia, fue necesario buscar otras alternativas en el diseño de los procesadores que permitieran continuar incrementando su rendimiento. Así fue cómo surgieron los procesadores *multicore*, los cuales integran varios núcleos computacionales más limitados y menos veloces pero que al combinarlos permiten mejorar tanto el rendimiento global como la eficiencia energética. Desde sus inicios los *multicore* han sofisticado su diseño con las sucesivas familias de procesadores integrando más núcleos computacionales e incorporando más niveles de caché, entre otras tecnologías.

En los últimos años, la tendencia para incrementar la velocidad de procesamiento sin transgredir los límites termo-energéticos ha sido integrar tecnología de aceleradores y coprocesadores a los nodos convencionales con procesadores *multicore*. Estos sistemas híbridos que emplean diferentes recursos de procesamiento se denominan arquitecturas heterogéneas y son capaces de obtener mejores cocientes FLOPS/Watt, lo que los hace más eficientes energéticamente. Existe una amplia variedad de aceleradores y coprocesadores, siendo los más populares las GPUs (mayoritariamente de NVIDIA sobre AMD) seguidos por los coprocesadores Xeon Phi de Intel. Por otro lado, las FPGAs (principalmente las de Altera y las de Xilinx) aparecen como una opción promisoria para debido a su capacidad de cómputo creciente, su bajo consumo energético y al desarrollo de nuevas herramientas que facilitan su programación.

Sin importar qué acelerador se elija, su programación siempre representa un desafío. La programación paralela resulta más compleja que la programación secuencial debido a que el programador enfrenta múltiples dificultades adicionales. Existe una amplia variedad de lenguajes y librerías para desarrollar programas paralelos, los cuales se pueden clasificar de acuerdo al grado de asistencia que le ofrecen al programador. OpenMP es el principal exponente del modelo guiado, en el cual el programador indica qué secciones de código se paralelizan y cómo hacerlo pero es el compilador el responsable de generar el código paralelo. El modelo completamente manual resulta ser el más costoso en términos de esfuerzo de programación, ya que el programador es el responsable de manejar todos los aspectos del paralelismo. Sin embargo, es el que permite obtener generalmente los mejores resultados en términos de rendimiento. Este enfoque se utiliza en la programación de sistemas de memoria distribuida, siendo MPI la herramienta más utilizada, y también en la de aceleradores con CUDA y OpenCL como los ejemplos más representativos.

Aunque las GPUs suelen tener picos de rendimiento y tasas de eficiencia energética mayores a los coprocesadores Xeon Phi, requieren del aprendizaje de lenguajes de programación específicos. En sentido opuesto, los coprocesadores Xeon Phi soportan modelos de programación tradicionales que favorecen el desarrollo y la portabilidad de los programas. Aunque las FPGAs se utilizaron tradicionalmente para problemas de punto fijo, la evolución

en sus capacidades computacionales las convierten en una alternativa a tener en cuenta para problemas de cómputo intensivo en punto flotante. El bajo consumo de potencia hace a las FPGAs muy eficientes desde el punto de vista energético, superando ampliamente al resto de los aceleradores. Desafortunadamente su costo de programación también es mayor al resto de los dispositivos. Es por ello que en la actualidad tanto Altera como Xilinx trabajan en el desarrollo de herramientas que permitan disminuir el esfuerzo de programación de estos dispositivos. Más allá de las capacidades teóricas, el rendimiento sostenible y la eficiencia energética de un sistema varían de acuerdo a cada problema particular y a su correspondiente implementación en software debido a que cada unidad de procesamiento cuenta con diferentes características de hardware, las cuales pueden ser más apropiadas para algunas aplicaciones que para otras.

Una amplia variedad de arquitecturas de hardware han sido utilizadas por la comunidad científica para acelerar el algoritmo SW. La parte más costosa computacionalmente de este algoritmo es obviamente el cálculo de la matriz de similitud. Aunque las dependencias de datos inherentes al problema restringen las formas en que se puede computar esta matriz, es posible aplicar dos esquemas de paralelismo diferentes:

- El esquema intra-tarea se basa en acelerar el cómputo de un único alineamiento por vez. Las implementaciones que adoptan este esquema suelen computar las celdas de cada antidiagonal de la matriz en paralelo, aprovechando que estos cálculos son independientes entre sí. También es posible computar varias celdas de una fila o de una columna simultáneamente; sin embargo, como este enfoque ignora las dependencias de datos antes mencionadas, se requiere de un ajuste posterior de los valores de las celdas para mantener la corrección del algoritmo.

- El esquema inter-tarea se basa en acelerar el cómputo de múltiples alineamientos al mismo tiempo. La mayor ventaja de este enfoque es la independencia en el cálculo de cada alineamiento.

Ambos enfoques han sido explorados por la comunidad científica en una amplia variedad de dispositivos. Para poder evaluar el desempeño de una implementación en forma independiente de las secuencias de consulta y de las bases de datos utilizadas para las diferentes pruebas al igual que de la arquitectura de soporte, se emplea la métrica de rendimiento CUPS.

Las implementaciones en CPU se han valido de instrucciones vectoriales para acelerar el cómputo de las matrices de similitud. La mayoría de los esfuerzos se han concentrado en el enfoque intra-tarea. En el año 1997, Wozniak logró una aceleración de 2× sobre el mejor algoritmo secuencial de esa época al presentar una implementación que aprovechaba las instrucciones SIMD de video para computar varias celdas de las antidiagonales. Posteriormente, en el año 2000, Rognes y Seeberg se convirtieron en los primeros en explotar las extensiones MMX/SSE para el algoritmo SW. Ellos encontraron que vectorizar a lo largo de la secuencia de consulta resulta más rápido en comparación a hacerlo por las antidiagonales, a pesar de tener que hacer más cálculos por ignorar algunas de las dependencias de datos inherentes al problema. A su vez, fueron los primeros en introducir la técnica QP, la cual mejora la localidad de los datos en la obtención de los puntajes de sustitución. Siete años más tarde, Farrar mejoró la propuesta de Rognes y Seeberg al reorganizar en franjas el cómputo de cada matriz. Este patrón de acceso permitió evitar accesos desalineados a los vectores y minimizar el impacto de las dependencias de datos. Además, al utilizar enteros de menor rango, logró duplicar la cantidad de alineamientos que se computaban simultáneamente. Como resulta-

do, Farrar reportó aceleraciones superiores a las propuestas anteriores. Por último, en el año 2011, Rognes presentó una implementación paralela que explota las instrucciones SSSE3 bajo el enfoque inter-tarea propuesto anteriormente por Alpern *et al.*, a la cual llamó SWIPE. Para acelerar la obtención de los puntajes de sustitución, Rognes propuso la técnica SP. La aceleración proviene de calcular de antemano todos los posibles vectores de sustitución de las secuencias de la base de datos, lo que reduce los cálculos durante el cómputo de la matriz y mejora la localidad de datos. SWIPE mostró rendimientos superiores a sus implementaciones contemporáneas por lo que se la considera la implementación SW más rápida para el conjunto de instrucciones SSE.

Las dos primeras implementaciones en GPU se remontan al año 2006 y ambas comparten muchas características; entre ellas que fueron programadas con OpenGL, que las matrices de alineamiento se computan por las antidiagonales y que tanto las secuencias como los *buffers* auxiliares son almacenados en memoria de textura. En el año 2008, Manavski y Valle presentaron SW-CUDA, la primera implementación SW para GPUs compatibles con CUDA. A diferencia de las propuestas anteriores, SW-CUDA emplea el enfoque inter-tarea y la técnica QP. Posteriormente, Yongchao Liu *et al.* presentaron CUDASW++, una implementación basada en CUDA que combina los enfoques intra-tarea e inter-tarea en forma adaptativa de manera de maximizar el rendimiento. CUDASW++ también se caracteriza por una cuidadosa organización de los accesos de memoria para aprovechar acceso coalescente y por la explotación de memoria constante para las secuencias de consulta y de memoria compartida para la matriz de sustitución. Más adelante, los mismos autores presentaron una versión mejorada de este software, conocida como CUDASW++ 2.0. Esta versión incorpora mejoras como la técnica QP para la obtención de los puntajes de sustitución, el uso de datos empaquetados en lugar de escalares y la virtualización de instrucciones SIMD en las placas gráficas. GASW es otra herramienta para búsquedas SW en GPUs que soportan CUDA, la cual propone varias optimizaciones entre las que se destacan la eliminación de cuellos de botella en la memoria y la conversión de la base de datos a un formato más conveniente para el procesamiento de las placas gráficas. En cuanto a OpenCL, las pocas implementaciones conocidas apuntan a la portabilidad más que al rendimiento. Por último, una tercera versión de CUDASW++ fue presentada en el 2013. Esta implementación combina cómputo concurrente en la CPU y en la GPU empleando el esquema inter-tarea en ambos casos y se la considera la implementación SW más rápida para sistemas basados en GPUs compatibles con CUDA.

El empleo de FPGAs para acelerar el alineamiento de secuencias biológicas es un tema ampliamente estudiado en la comunidad HPC, aunque la mayoría de las propuestas se enfocan en el caso de ADN por ser más sencillo de resolver. En general, las implementaciones se basan en crear bloques básicos que sean capaces de computar el valor de una celda de una matriz en cada ciclo de reloj. Luego, múltiples instancias de estos bloques son combinados a la vez para formar arreglos sistólicos que puedan procesar grandes cantidades de datos en paralelo. De todas formas, existen muy pocas implementaciones que sean completas desde el punto de vista funcional. Las implementaciones que reportan los mejores resultados suelen contemplar pruebas sintéticas con el objetivo de mostrar la viabilidad del empleo de esta clase de aceleradores aunque su uso potencial en el mundo real es limitado.

Los coprocesadores Xeon Phi también pueden ser empleados para acelerar el alineamiento de secuencias biológicas. En el año 2014, Liu y Schmidt presentaron la herramienta SWAPHI basada en el modelo *offload* de OpenMP. SWAPHI soporta los esquemas de paralelismo intra-tarea e inter-tarea al igual que las técnicas QP y SP para la obtención de los puntajes de sustitución. XSW es una herramienta alternativa publicada en el año 2014. Al igual que

SWAPHI, XSW emplea el esquema inter-tarea y la técnica SP para acelerar el cómputo de los alineamientos. A diferencia de SWAPHI, XSW se ejecuta en modo nativo con Pthreads como modelo de programación. Una versión extendida de XSW, conocida como XSW 2.0, fue desarrollada posteriormente. Esta implementación adopta el modelo *offload* para combinar cómputo concurrente en la CPU.

Capítulo 3

Optimización de SW para sistemas heterogéneos basados en aceleradores Xeon Phi: SWIMM

Intel presentó recientemente el coprocesador Xeon Phi, el cual fue diseñado para aplicaciones HPC. Entre sus principales ventajas se encuentran su facilidad de programación y su gran compatibilidad con códigos x86. Este capítulo presenta la herramienta SWIMM para búsquedas de similitud en sistemas heterogéneos basados en aceleradores Xeon Phi, la cual trabaja con bases de datos biológicas reales y permite configurar los parámetros de ejecución. Además, admite tres modos de ejecución: (1) cómputo en el *host*, (2) cómputo en los coprocesadores y (3) cómputo concurrente en el *host* y los coprocesadores. Mediante esta herramienta, se estudia y se evalúa la viabilidad de esta clase de sistemas heterogéneos para el alineamiento de secuencias de proteínas desde la perspectiva del rendimiento. Primero se describen las diferentes implementaciones propuestas junto a las técnicas de programación y optimización empleadas (Sección 3.1). A continuación, se detallan los distintos experimentos llevados a cabo y se analizan los resultados obtenidos (Sección 3.2). Para concluir, la Sección 3.3 presenta un resumen del capítulo.

3.1. Implementación

En esta Sección se describen las técnicas de programación y de optimización realizadas para las dispositivos Xeon y Xeon Phi. En primer lugar, se proponen implementaciones individuales para cada uno de los procesadores y luego se combinan ambas para obtener una implementación heterogénea híbrida. Los algoritmos propuestos comparten el mismo flujo de control, el cual se puede resumir en los siguientes pasos:

1. Etapa de preprocesamiento: las secuencias de la base de datos son preprocesadas de forma de adaptarlas para su posterior procesamiento paralelo.

2. Etapa Smith-Waterman: una vez preprocesada la base de datos, se computan los alineamientos mediante el método SW.

3. Etapa de ordenación: finalmente, los puntajes de similitud son ordenados en forma descendente.

A continuación se describen las características de las implementaciones propuestas.

3.1.1. Esquema de paralelismo

Los alineamientos se computan siguiendo el esquema de paralelismo inter-tarea. Como se analizó en la Sección 2.3.2.1, este esquema de paralelización es el que logra mayor rendimiento a partir de la explotación de las pequeñas capacidades vectoriales disponibles en la mayoría de los microprocesadores modernos. En particular, las secuencias de la base de datos se agrupan de acuerdo al ancho de la unidad de procesamiento vectorial (VPU, por sus siglas en inglés). De forma de equilibrar la carga entre las diferentes VPUs, las secuencias de la base de datos son ordenadas antes de ser agrupadas.

3.1.2. Preprocesamiento de la base de datos

En la etapa de preprocesamiento, se ordenan las secuencias de la base de datos según su longitud antes de agruparlas de acuerdo al ancho de la VPU del procesador destino. Además, se completan con valores nulos (*dummy*) las secuencias más cortas de cada grupo para que todas posean la misma longitud. De esta forma, se minimizan los desbalances por el procesamiento de los grupos de secuencia. Por otra parte, para la implementación en el Xeon Phi y la implementación heterogénea híbrida, la base de datos se divide en *chunks*. La implementación en el Xeon explota una granularidad más fina en la distribución del trabajo para minimizar el desbalance en la carga de los hilos. Como este proceso debe ser repetido para cada búsqueda, las bases de datos son preprocesadas en forma separada para evitar trabajo repetido. Las bases de datos se leen en formato FASTA[1] y luego son transformadas a un formato binario interno, el cual facilita una lectura posterior más rápida.

3.1.3. Implementación para Xeon y Xeon Phi

Las implementaciones explotan múltiples niveles de paralelismo:

- Paralelismo a nivel de datos: Como se mencionó en la Sección 2.2.2.2, tanto los Xeon como los Xeon Phi soportan instrucciones SIMD a través del uso de vectorización guiada o codificación manual mediante instrucciones intrínsecas. Ambos enfoques de explotación SIMD son explorados:
  - Vectorización guiada: mediante el uso de directivas al preprocesador. A partir de la versión 4.0, OpenMP ofrece una directiva específica para expresar paralelismo de datos: `#pragma omp simd` fuerza la vectorización de un bucle. Una de las ventajas más importantes en este tipo de vectorización es la portabilidad. Si bien la cantidad de canales de las VPUs en el Xeon y en el Xeon Phi pueden diferir, el compilador es capaz de generar dos versiones diferentes de acuerdo al procesador destino.
  - Vectorización con intrínsecas: utilizando las extensiones SSE y AVX2 para los Xeon y las KNC para los Xeon Phi.
- Paralelismo a nivel de hilo: Para explotar el paralelismo entre los múltiples núcleos, se emplea una versión multihilada del algoritmo SW basada en el modelo de programación OpenMP, el cual se encuentra disponible tanto en los Xeon como en los Xeon Phi.

[1] Descripción del formato FASTA: http://blast.ncbi.nlm.nih.gov/blastcgihelp.shtml

Algoritmo 3.1 Pseudo-código de la implementación SW usando vectorización guiada para el coprocesador Xeon Phi

```
▷ Q son las secuencias de consulta
▷ vD es la base de datos preprocesada
▷ SM es la matriz de sustitución
▷ G_o es el puntaje por insertar un gap
▷ G_e es el puntaje por extender un gap
▷ S son los puntajes de alineamiento
▷ n_mic es el número de Xeon Phi
▷ t_mic es el número de hilos de los Xeon Phi

#pragma omp parallel num_threads(n_mic)
{
    #pragma offload target(mic:mic_no) in (Q, SM, G_o, G_e)
    { alocar buffers }
    #pragma omp for schedule(dynamic)
    for c < obtener_número_de_chunks(vD) do
        #pragma offload target(mic:mic_no) in(vD_c) out(S_c)
        {S_c = SW_SEARCH(Q, vD_c, SM, G_o, G_e, t_mic) }
    end for
    #pragma offload target(mic:mic_no)
    { liberar buffers }
}
S = ordenar(S) ▷ en orden descendente

function SW_SEARCH(Q, vD_c, SM, G_o, G_e, t) {
    if QP then
        QPs = construir_QPs(Q, SM)
    end if
   numAlineamientos = obtener_número_de_secuencias(Q)×obtener_número_de_secuencias(vD_c)
    #pragma omp parallel num_threads(t) schedule(dynamic)
    for a < numAlineamientos do
        q = obtener_secuencia(Q, a)
        vd = obtener_secuencia(vD_c, a)
        if SP then
            P = construir_SP(vd, SM)
        else
            P = obtener_QP(QPs, q)
        end if
        for i < |q| do
            #pragma unroll
            for j < |vd| do
                [H_i,j, E_i,j, F_i,j] = SW_CORE(H, P, E, F, G_o, G_e)
                S_a = max(S_a,H_i,j)
            end for
        end for
    end for
    return S
end function
```

En el Algoritmo 3.1 se muestra el pseudo-código de la implementación SW usando vectorización guiada para el coprocesador Xeon Phi. El bucle principal que itera sobre los *chunks* de la base de datos se distribuye entre los hilos con la directiva `#pragma omp for` utilizando la política de planificación `dynamic`. Cada hilo de la CPU delega el *chunk* de la base de datos asignado al correspondiente coprocesador para computar los alineamientos y luego espera a su finalización. Dentro del Xeon Phi, los alineamientos se resuelven en paralelo usando multihilado otra vez. Las iteraciones del bucle paralelo son distribuidas usando una política de planificación `dynamic`. Más allá de haber ordenado la base de datos de referencia, la política de planificación `static` no provee buenos resultados, como se indicó en una investigación similar [39]. Al utilizar la planificación `guided`, el rendimiento tiende a ser menor. Para aliviar el *overhead* asociado con la reserva y liberación de memoria, todos los *buffers* de los hilos son alocados previo a la etapa SW y luego reutilizados en los siguientes *offloads*. El *offload* inicial se aprovecha al mismo tiempo para transferir datos comunes a todos los hilos, como las secuencias de consulta y la matriz de sustitución. Por último, una vez que todos los *chunks* fueron procesados, se ordenan todos los puntajes de alineamiento en sentido descendente para completar la etapa de ordenamiento.

Resulta importante remarcar que el código de la versión vectorizada en forma guiada para el procesador Xeon es esencialmente el mismo que para el Xeon Phi, a excepción de las directivas `#pragma offload` que no son incluidas. Como se mencionó en la Sección 3.1.2, la base de datos para el Xeon se conforma de un sólo *chunk*, por lo que el código para este procesador está representado por una única invocación a la función *SW_SEARCH*.

Con respecto a las versiones vectorizadas mediante intrínsecas, los algoritmos para el Xeon y el Xeon Phi son estructuralmente iguales a sus equivalentes guiadas. La diferencia se encuentra en la forma en que implementan la función *SW_CORE*; las versiones guiadas emplean las directivas `#pragma omp simd` mientras que las versiones manuales se valen de instrucciones intrínsecas. En la Figura 3.1 se muestran las diferentes implementaciones intrínsecas de *SW_CORE*. En todos los casos, la matriz de alineamiento se procesa fila a fila. *vCur* representa la celda de la matriz que se está computando mientras que *vPrev* se corresponde con la anterior. *vH* mantiene la fila procesada previamente y sus valores son reemplazados por la fila actual a medida que no se los necesita más. *vSub* es el puntaje de sustitución para los residuos de la secuencia de la base de datos contra el residuo de la secuencia de consulta. *vE* y *vF* son los vectores de puntajes para los alineamientos terminados en un *gap* en la secuencia de consulta y en la de la base de datos, respectivamente. *vGoe* representa el vector para la suma de las penalizaciones por inserción y por extensión de *gaps* mientras que *vGe* es el vector de penalización por extensión de *gap*. Por último, *vS* contiene el puntaje óptimo de alineamiento actual.

SSE	AVX2	KNC
vCur = _mm_adds_epi8(vH[j-1], vSub);	vCur = _mm256_adds_epi8(vH[j-1], vSub);	vCur = _mm512_add_epi32(vH[j-1], vSub);
vCur = _mm_max_epi8(vCur, vF[i]);	vCur = _mm256_max_epi8(vCur, vF[i]);	vCur = _mm512_max_epi32(vCur, vF[i]);
vCur = _mm_max_epi8(vCur, vE[j]);	vCur = _mm256_max_epi8(vCur, vE[j]);	vCur = _mm512_max_epi32(vCur, vE[j]);
vCur = _mm_max_epi8(vCur, vZero);	vCur = _mm256_max_epi8(vCur, vZero);	vCur = _mm512_max_epi32(vCur, vZero);
vS = _mm_max_epi8(vS, vCur);	vS = _mm256_max_epi8(vS, vCur);	vS = _mm512_max_epi32(vS, vCur);
vF[i] = _mm_sub_epi8(vF[i], vGe);	vF[i] = _mm256_sub_epi8(vF[i], vGe);	vF[i] = _mm512_sub_epi32(vF[i], vGe);
vE[j] = _mm_sub_epi8(vE[j], vGe);	vE[j] = _mm256_sub_epi8(vE[j], vGe);	vE[j] = _mm512_sub_epi32(vE[j], vGe);
vAux = _mm_sub_epi8(vCur, vGoe);	vAux = _mm256_sub_epi8(vCur, vGoe);	vAux = _mm512_sub_epi32(vCur, vGoe);
vF[i] = _mm_max_epi8(vF[i], vAux);	vF[i] = _mm256_max_epi8(vF[i], vAux);	vF[i] = _mm512_max_epi32(vF[i], vAux);
vE[j] = _mm_max_epi8(vE[j], vAux);	vE[j] = _mm256_max_epi8(vE[j], vAux);	vE[j] = _mm512_max_epi32(vE[j], vAux);
vH[j-1] = vPrev;	vH[j-1] = vPrev;	vH[j-1] = vPrev;
vPrev = vCur;	vPrev = vCur;	vPrev = vCur;

Figura 3.1: Implementaciones intrínsecas de la función *SW_CORE*

3.1.3.1. Selección de puntajes de sustitución

Para obtener los puntajes de la matriz de sustitución, el código implementa en forma separada las optimizaciones QP y SP descritas en la Sección 2.3.2.1.

Al utilizar el esquema inter-tarea, la estrategia QP se basa en crear una matriz auxiliar de tamaño $|q| \times |\sum|$, donde q en la secuencia de consulta y $\sum$ es el alfabeto. Cada fila de esta matriz contendrá los puntajes del residuo correspondiente de la secuencia de consulta para todos los posibles residuos del alfabeto. Aunque esta técnica incrementa los requerimientos de memoria caché, suele producir una mejora en la localidad de datos ya que se reemplaza un acceso aleatorio a la matriz de sustitución por un acceso lineal a la matriz QP.

Como se describió en la Sección 2.3.2.1, la técnica SP se basa en construir un arreglo auxiliar de puntajes de tamaño $n \times l \times |\sum|$, donde n representa la longitud de la secuencia de la base de datos, l es el número de canales en el vector y $\sum$ es el alfabeto. Como cada fila del SP forma un vector de puntajes de l canales, sus valores pueden ser recuperados utilizando una única operación vectorial *load*. Sin embargo, el SP debe ser re-construido para cada secuencia de la base de datos, por lo que su utilidad debe ser evaluada, especialmente para secuencias de consulta pequeñas.

3.1.3.2. Selección de rango del tipo de dato entero

La mayoría de los puntajes de alineamiento puede ser representados utilizando enteros de bajo rango. En el caso del procesador Xeon, los alineamientos son computados inicialmente usando operaciones con enteros de 8 bits. Mientras que el conjunto de instrucciones SSE permite empaquetar 16 enteros de 8 bits en un sólo registro SIMD, las extensiones AVX2 lo duplican permitiendo 32 elementos. Las sumas se realizan utilizando operaciones aritméticas con saturación para posibilitar la detección de *overflow*. Cuando se detecta *overflow* (el puntaje de alineamiento es igual al máximo valor de la representación entera empleada), el alineamiento es recalculado usando el siguiente rango de enteros más amplio. Desafortunadamente, el Xeon Phi sólo soporta enteros de 32 bits, por lo que no puede procesar más de 16 alineamientos al mismo tiempo.

3.1.3.3. Desenrollado de bucle

Se utilizó la técnica de desenrollado de bucles para acelerar el cómputo de las matrices de alineamiento. En particular, se desenrollan las iteraciones del bucle más interno que computa las celdas de las filas de la matriz (que se corresponde con la secuencia de la base de datos). Para ello, se empleó la directiva `#pragma unroll` proporcionada por el compilador de Intel. Utilizando esta directiva, el programador le indica al compilador que debe desenrollar el bucle que se encuentra debajo de ella.

3.1.3.4. Localidad de datos

La localidad de los datos es un elemento clave para obtener buen rendimiento, especialmente en el Xeon Phi, donde el procesamiento por bloques es también necesario para reducir el número de fallos de caché [116]. Además, las estructuras de datos se alinean al momento de la reserva de memoria para evitar el *overhead* de accesos a memoria desalineados (alineación a 32 bytes para el Xeon y a 64 bytes para el Xeon Phi).

3.1.4. Implementación heterogénea híbrida

Utilizando los códigos para ambos procesadores, se desarrollaron dos versiones heterogéneas híbridas que aprovechan tanto el Xeon como el Xeon Phi en forma simultánea. Las versiones se diferencian en la forma en que distribuyen el trabajo entre las unidades de procesamiento; mientras que una emplea una técnica estática, la otra utiliza una técnica dinámica. Las técnicas estáticas pueden llevar a una distribución con mejores resultados que las dinámicas. Sin embargo, si no se dispone de antemano de algún tipo de información relacionada a las capacidades computacionales de cada dispositivo, pueden ser poco útiles en el *mundo real*. En sentido contrario, las técnicas dinámicas no necesitan de ninguna información previa pero pueden sufrir penalizaciones de rendimiento debido a desbalances u ocio.

3.1.4.1. Distribución estática

En el Algoritmo 3.2 se muestra el pseudo-código de la implementación heterogénea híbrida con distribución estática del trabajo. Antes de computar los alineamientos, la base de datos se divide en dos partes: una para la CPU y otra para el coprocesador. Para ello, se emplea un parámetro configurable (p) que indica el porcentaje de residuos que son asignados a la CPU; el resto se destina al coprocesador. Una vez dividida la base de datos, se invoca a la función *SW_SEARCH* dos veces. La primera invocación se realiza delegando el cómputo al Xeon Phi en forma asíncrona. La segunda invocación corresponde a la CPU. Esta implementación se basa en un esquema de paralelismo anidado, ya que la rutina *SW_SEARCH* crea una nueva región paralela (se genera un hilo OpenMP por cada hilo hardware del dispositivo destino). Antes de ordenar los puntajes de similitud, se espera a que ambas instancias de la función *SW_SEARCH* se hayan completado.

Algoritmo 3.2 Pseudo-código de la implementación heterogénea híbrida con distribución estática de la carga de trabajo

▷ p es el porcentaje de residuos de la base de datos asignados a la CPU
▷ vD_{cpu} es la parte de la base de datos correspondiente a la CPU
▷ vD_{mic} es la parte de la base de datos correspondiente a los coprocesadores
▷ t_{cpu} es el número de hilos de la CPU en el segundo nivel

dividir (vD, p, vD_{cpu}, vD_{mic})

#pragma offload target(mic) in(Q, vD_{mic}, SM, G_o, G_e) out(S_{mic}) signal(sem)
{S_{mic} = *SW_SEARCH*(Q, vD_{mic}, SM, G_{o}, G_{e}, t_{mic}) } ▷ Computar alineamientos en el Xeon Phi

S_{cpu} = *SW_SEARCH*(Q, vD_{cpu}, SM, G_{o}, G_{e}, t_{cpu}) } ▷ Computar alineamientos en la CPU

#pragma offload target(mic) wait(sem)

S = *ordenar*(S, t_{scpu}) ▷ Ordenar puntajes en sentido descendente

La estrategia de distribución de esta implementación es muy sencilla. Resulta evidente que esta implementación tiene una usabilidad limitada en el mundo real, ya que se deben

conocer las potencias de cómputo relativas de los dispositivos en forma previa para lograr una carga balanceada y, en consecuencia, un buen rendimiento. De todas formas, se considera esta implementación para fines analíticos.

3.1.4.2. Distribución dinámica

En el Algoritmo 3.3 se muestra el pseudo-código de la implementación heterogénea híbrida con distribución dinámica del trabajo. Se puede apreciar que esta implementación es muy similar a la del Algoritmo 3.1, con la salvedad de que se utiliza una sentencia condicional adicional para seleccionar en qué dispositivo se procesará el *chunk*. Al igual que la anterior, esta implementación se basa en un esquema de paralelismo anidado: inicialmente se generan n_{mic}+1 hilos (n_{mic} corresponde al número de coprocesadores) que invocan a la rutina *SW_SEARCH*, la cual crea una región paralela anidada (se genera un hilo OpenMP por cada hilo hardware del dispositivo destino). Los primeros n_{mic} hilos son responsables de delegar sus *chunks* a los correspondientes coprocesadores mientras que el último hilo computa los alineamientos de sus *chunks* empleando los recursos de la CPU.

Algoritmo 3.3 Pseudo-código de la implementación heterogénea híbrida con distribución dinámica de la carga de trabajo

```
▷ t_cpu es el número de hilos del Xeon en el segundo nivel

#pragma omp parallel num_threads(n_mic+1)
{
   if mic_thread then
      #pragma offload target(mic:mic_no) in (Q, SM, G_o, G_e)
      { alocar buffers }
   end if
   #pragma omp for schedule(dynamic)
   for c < obtener_número_de_chunks(vD) do
      if mic_thread then
         #pragma offload target(mic:mic_no) in(vD_c) out(S_c)
         {S_c = SW_SEARCH(Q, vD_c, SM, G_o, G_e, t_mic) } ▷ Computar alineamientos en el Xeon Phi
      else
         {S_c = SW_SEARCH(Q, vD_c, SM, G_o, G_e, t_cpu) } ▷ Computar alineamientos en el Xeon
      end if
   end for
   if mic_thread then
      #pragma offload target(mic:mic_no)
      { liberar buffers }
   end if
}
S = ordenar(S, t_cpu) ▷ Ordenar puntajes en sentido descendente
```

Es importante notar que, al igual que en la versión anterior, la distribución del trabajo requiere de un enfoque de paralelismo anidado en la implementación del Xeon Phi y en la implementación híbrida. En estos dos últimos casos, la base de datos se divide en *chunks* de

aproximadamente el mismo tamaño de residuos. Estos *chunks* son distribuidos siguiendo una política de planificación bajo demanda: una vez que un dispositivo finalizó el procesamiento de un *chunk* (primer nivel), inmediatamente se le asigna otro hasta que no haya más *chunks* que procesar. Una vez que un *chunk* fue asignado, el dispositivo (Xeon/Xeon Phi) lo procesa distribuyendo los grupos de secuencias que lo conforman entre sus núcleos (segundo nivel).

Por último, resulta importante destacar que, a diferencia de la versión anterior, esta implementación soporta múltiples coprocesadores y no requiere de información por parte del usuario para distribuir el trabajo entre los dispositivos.

3.2. Resultados experimentales

3.2.1. Entorno y diseño experimental

Todas las pruebas fueron realizadas en dos arquitecturas heterogéneas con sistema operativo CentOS 6.5:

- La primera está equipada con:
  - Dos procesadores Intel Xeon E5-2670 2.60GHz de ocho núcleos (dos hilos hardware por núcleo) y 32 GB de memoria RAM.
  - Un coprocesador Intel Xeon Phi 3120P de 57 núcleos (cuatro hilos hardware por núcleo) y 6 GB de memoria dedicada.
- La segunda está equipada con:
  - Dos procesadores Intel Xeon E5-2695 v3 2.30GHz de 14 núcleos (dos hilos hardware por núcleo) y 64 GB de memoria RAM.
  - Un coprocesador Intel Xeon Phi 3120P de 57 núcleos (cuatro hilos hardware por núcleo) y 6 GB de memoria dedicada.
  - Una GPU NVIDIA Tesla K20c (2496 núcleos CUDA) con 5 GB de memoria dedicada y Capacidad Computacional 3.5.

El compilador utilizado es Intel ICC (versión 14.0.2.144) con nivel de optimización O3 habilitado por defecto. Los hilos OpenMP fueron mapeados a los núcleos del procesador mediante la afinidad `scatter`. La versión del SDK de CUDA es 7.0.

Los experimentos utilizados para evaluar rendimiento son similares a los realizados en investigaciones anteriores [4, 95, 39]. Se evaluó el rendimiento de las diferentes implementaciones buscando 20 secuencias de proteínas en dos bases de datos reconocidas: Swiss-Prot (versión 2013_11)[2] y Environmental NR (versión 2014_11)[3]. La base de datos Swiss-Prot consiste de 192480382 aminoácidos en 541561 secuencias, siendo la más larga de 35213 residuos. La base de datos Environmental NR se compone de 1291019045 residuos en 6552667 secuencias, siendo la longitud máxima igual a 7557. Las secuencias de consulta fueron extraídas de la base de datos Swiss-Prot (números de acceso: P02232, P05013, P14942, P07327, P01008, P03435, P42357, P21177, Q38941, P27895, P07756, P04775, P19096, P28167, P0C6B8,

[2]La base de datos Swiss-Prot se encuentra disponible en http://web.expasy.org/docs/swiss-prot_guideline.html

[3]La base de datos Environmental NR se encuentra disponible en ftp://ftp.ncbi.nih.gov/blast/db/FASTA/env_nr.gz

P20930, P08519, Q7TMA5, P33450 y Q9UKN1) y sus longitudes varían entre 144 y 5478 residuos. La matriz de sustitución seleccionada es BLOSUM62 y los puntajes de inserción y extensión de *gaps* fueron establecidos en 10 y 2, respectivamente.

En forma adicional, se seleccionaron las herramientas SWIPE (versión 2.0.5), SWAPHI (versión 1.0.4) y XSW 2.0 (versión 2.0) para realizar una comparación de rendimiento con las implementaciones Xeon, Xeon Phi e híbrida, respectivamente. Además, también se seleccionó CUDASW++ 3.0 (versión 3.1) para una comparación con GPUs. Como se detalló en la Sección 2.3.2, estas implementaciones son las más eficientes de su clase.

Resulta importante destacar que los experimentos de las Secciones 3.2.2 y 3.2.3 fueron realizados empleando la base de datos Swiss-Prot. Sin embargo, debido a su tamaño limitado, los experimentos con el sistema heterogéneo completo fueron llevados a cabo con la base de datos Environmental NR (de mayor tamaño que Swiss-Prot), de forma de poder realizar análisis relacionados con la comparación de rendimiento con GPUs (Sección 3.2.5) y el efecto de variar la distribución de carga en el rendimiento (Sección 3.2.4). Con respecto a los sistemas utilizados, los experimentos mencionados en las Secciones 3.2.2, 3.2.3 y 3.2.4 fueron realizados en la primera arquitectura descrita (excepto el análisis de rendimiento de las instrucciones AVX2). Las pruebas de las Sección 3.2.5 se llevaron a cabo en el segundo sistema.

Por último, las implementaciones presentadas en este capítulo han sido integradas en una única herramienta, a la cual se ha llamado SWIMM y que se encuentra disponible en un repositorio web público[4].

3.2.2. Resultados de la implementación en el Xeon

Como se mencionó en la Sección 2.3.2, se utiliza la métrica GCUPS para evaluar el rendimiento de las implementaciones SW. Para el cálculo de los GCUPS de las implementaciones descritas en este capítulo se consideró dentro del tiempo de ejecución: la creación de *buffers*, las transferencias del Xeon al Xeon Phi, el cómputo de los alineamientos y las transferencias del Xeon Phi al Xeon.

En la Figura 3.2 se muestra el rendimiento del sistema basado en procesadores Xeon E5-2670 para los diferentes enfoques bajo evaluación utilizando un número creciente de hilos OpenMP. Las implementaciones sin vectorización apenas mejoran su rendimiento de 0.5 a 3 GCUPS (denotadas como *no-vec* en la figura). La vectorización automática no sólo mejora el rendimiento en forma significativa sino que también posibilita que escale con el número de hilos (etiqueta *simd* en la figura). Los códigos basados en intrínsecas SSE (denotados como *intrinsic*) superan a sus equivalentes guiadas. La diferencia de rendimiento entre los dos enfoques es notable (aproximadamente 2×). En general, la técnica SP da mejores resultados que QP. En particular, SP logra una aceleración casi lineal, alcanzando 113.7 GCUPS con 32 hilos OpenMP.

En la Figura 3.3 se ilustra el rendimiento de las diferentes implementaciones al variar la longitud de la secuencia de consulta usando 32 hilos OpenMP. La mayoría de las implementaciones no exhiben grandes variaciones en su comportamiento dado que emplean el esquema de paralelismo inter-tarea. Sin embargo, se observa una pequeña caída en el rendimiento para los tamaños de secuencia más pequeños, el cual es más notable con las versiones vectorizadas manualmente. Si bien SP requiere construir una matriz de puntajes para cada secuencia de la base de datos, resulta más eficiente con respecto al acceso a la memoria que su variante QP.

[4]SWIMM se encuentra disponible *online* en https://github.com/enzorucci/SWIMM

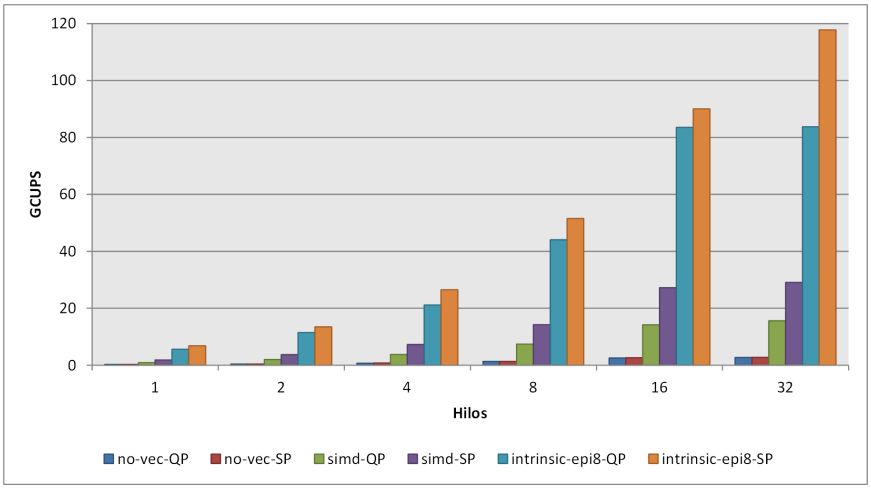

Figura 3.2: Rendimiento de las diferentes implementaciones en el sistema basado en Xeon E5-2670 al variar el número de hilos.

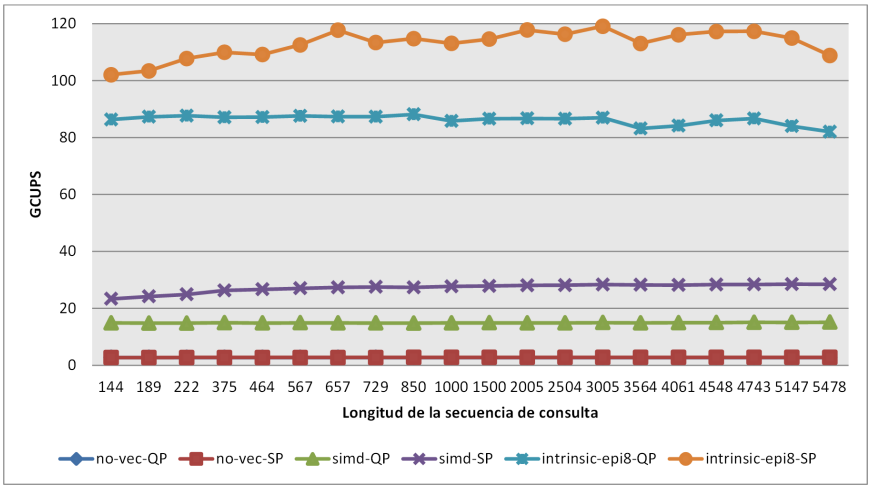

Figura 3.3: Rendimiento de las diferentes implementaciones en el sistema basado en Xeon E5-2670 al variar el tamaño de la secuencia de consulta

Los puntajes de alineamiento no requieren de un rango de representación muy amplio. Por lo tanto es posible utilizar tipos de datos pequeños que permitan una mayor explotación de paralelismo a nivel de datos. En particular, las instrucciones SSE permiten empaquetar 16 enteros de 8 bits, 8 enteros de 16 bits o 4 enteros de 32 bits en un sólo vector. En la Figura 3.4, se exhibe el rendimiento de las implementaciones intrínsecas utilizando enteros de diferente rango al variar el número de hilos. Como se espera, el uso de un tipo de dato más pequeño mejora el rendimiento en forma significativa, desde 29.9 y 57.5 GCUPS para las versiones de 32 bits y 16 bits (denotadas como *epi32* y *epi16* respectivamente) a 113.7 GCUPS en la implementación con 8 bits (etiquetada como *epi8*).

Las versiones que emplean 8 y 16 bits también consideran la detección de *overflow*, como se mencionó en la Sección 3.1.3.2. Para verificar cuando ocurre *overflow* se emplean operaciones aritméticas con saturación. Cuando se detecta el valor de saturación en el puntaje de alineamiento, el alineamiento se recomputa usando el siguiente rango de enteros. Se puede

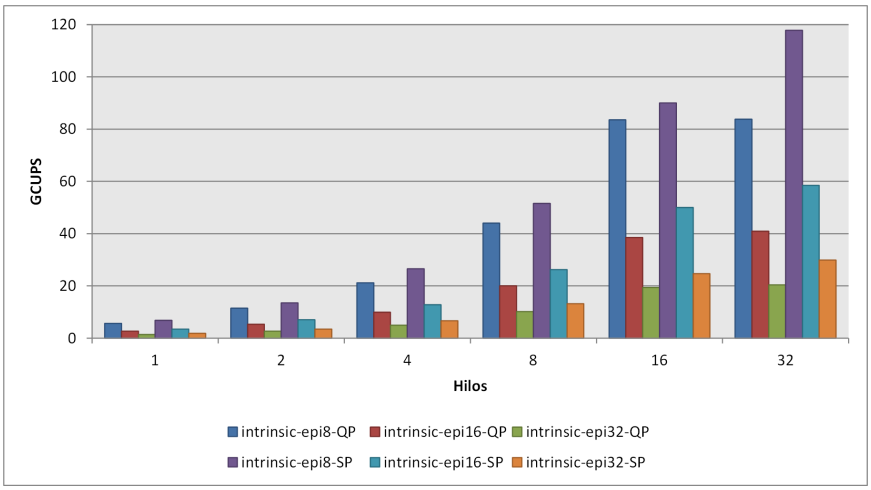

Figura 3.4: Rendimiento de las implementaciones intrínsecas en el sistema basado en Xeon E5-2670 utilizando enteros de diferente rango al variar el número de hilos

observar que el *overhead* adicional relacionado con recomputar el alineamiento usando un tipo de dato más ancho es prácticamente despreciable. El rendimiento sólo se ve afectado por el ancho del vector. De hecho, en los experimentos realizados con la base de datos Swiss-Prot sólo ocurre *overflow* en 0.5 % de los alineamientos en la representación de 8 bits mientras que no hay casos de *overflow* al utilizar 16 bits. Más aun, tasas de *overflow* similares se observan con la base de datos Environmental NR (0.75 % para 8 bits y 0 % para 16 bits), por lo que vale la pena usar enteros de 8 bits con el fin de explotar tanto paralelismo de datos como sea posible.

Como la explotación de paralelismo a nivel de datos resulta ser un factor determinante para obtener mejores rendimientos, interesa estudiar el desempeño alcanzable al emplear las instrucciones AVX2 diseñadas para anchos vectoriales de 256 bits. Las extensiones AVX2 se encuentran disponibles a partir de la familia de microprocesadores Haswell de Intel. En la Figura 3.5 se compara el rendimiento alcanzado por SWIMM al usar tanto las extensiones SSE como las AVX2 sobre el sistema basado en los procesadores Xeon E5-2695 v3. A su vez, también se incluye el rendimiento de la implementación SWIPE. En el caso de las SSE, se puede observar que los rendimientos de SWIPE y de SWIMM son muy parejos, siendo el primero el que saca una pequeña ventaja con a lo sumo 28 hilos. Sin embargo, se puede apreciar que SWIPE no se beneficia del uso de Hyper-Threading ya que su rendimiento decae levemente cuando se aumenta de 28 a 56 hilos, lo que deja a SWIMM como la implementación de mayor aceleración en ese caso. En el caso de las AVX2, se puede notar que SWIMM supera a SWIPE con ambos esquemas de puntaje de sustitución, pero es con SP con la cual le saca mayor ventaja logrando incluso aceleraciones de hasta 1.4×. También con el esquema SP se alcanza el pico máximo de rendimiento (342.3 GCUPS) al utilizar todos los núcleos disponibles del sistema. Por último, de acuerdo a la aceleración observada con el uso de las instrucciones AVX2 en comparación a las SSE, se espera obtener mejores rendimientos en sistemas con capacidades vectoriales más amplias, como por ejemplo la siguiente generación de procesadores Xeon que contarán con las anunciadas AVX-512.

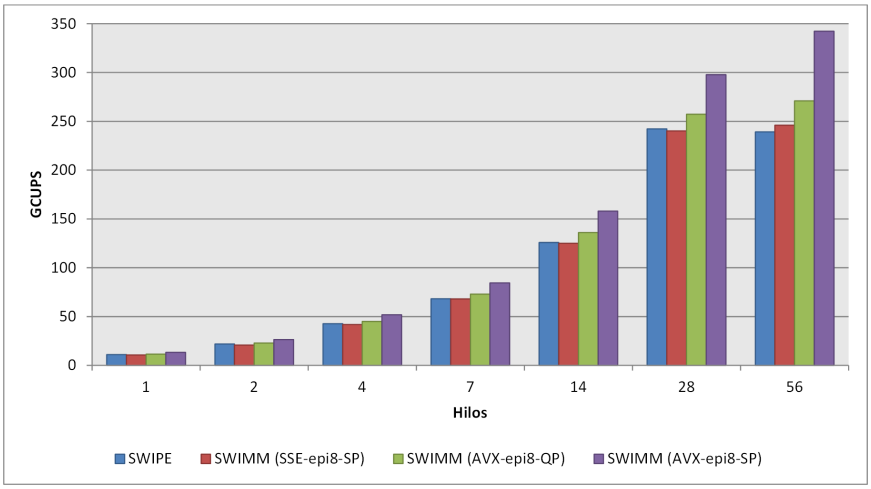

Figura 3.5: Rendimiento de SWIMM en comparación con SWIPE en el sistema basado en Xeon E5-2695 v3

3.2.3. Resultados de la implementación en el Xeon Phi

En la Figura 3.6 se muestra el rendimiento del coprocesador Xeon Phi para las diferentes implementaciones desarrolladas al variar el número de hilos (de 57 a 228 hilos hardware). Es importante recordar que, a diferencia de los procesadores Xeon, este acelerador sólo provee operaciones con enteros de 32 bits. Resulta evidente que esta característica limita radicalmente el rendimiento alcanzable teniendo en cuenta que la explotación de paralelismo de datos es primordial en esta aplicación, como se mostró en la sección previa.

Al igual que con los procesadores Xeon, las implementaciones que no utilizan vectorización prácticamente no mejoran sus prestaciones al incrementar la cantidad de hilos. Las implementaciones que emplean vectorización guiada presentan desempeños similares, alcanzando un máximo de 12.3 y 14.3 GCUPS para los esquemas QP y SP, respectivamente. Una vez más, los códigos basados en las intrínsecas KNC superan a sus equivalentes guiadas: 38.8 y 43.9 para los enfoques QP y SP. Sin embargo, en este caso, la diferencia de rendimiento entre ambos enfoques es levemente menor (un promedio de 1.1×). El mejor rendimiento es logrado por la implementación con intrínsecas y esquema SP (43.9 GCUPS con 228 hilos), la cual escala casi linealmente con el número de hilos.

En la Figura 3.7 se ilustra el rendimiento de las diferentes implementaciones en el Xeon Phi usando 228 hilos hardware al variar el tamaño de la secuencia de consulta. Se puede observar una notable caída en el rendimiento con secuencias de menos de 567 residuos. Un comportamiento similar se ha presentado en una investigación previa en el Xeon Phi [39]. Este fenómeno se explica por el *overhead* que provoca la construcción del SP, el cual no compensa los beneficios de indexación posteriores en el caso de secuencias pequeñas. De hecho, el esquema QP es más eficiente en el Xeon Phi con secuencias cortas ya que el repertorio de instrucciones incorpora una operación de permutación única, mientras que en los procesadores Xeon se deben ejecutar varios operaciones de *shuffling* para lograr el mismo resultado. Por este motivo, y para evitar las divergencias observadas de acuerdo al tamaño de la consulta, se propone una implementación adaptable que combina los enfoques QP y SP y los selecciona de acuerdo a la longitud de la secuencia de consulta (denotado en la figura como AP, del inglés *Adaptive Profile*).

Como se mencionó anteriormente, la explotación de capacidad SIMD es una de las claves

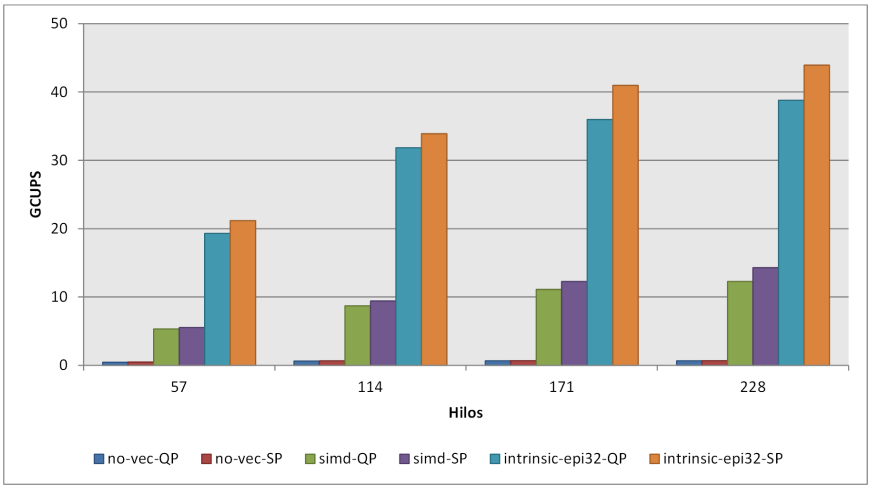

Figura 3.6: Rendimiento de las diferentes implementaciones en el Xeon Phi al variar el número de hilos.

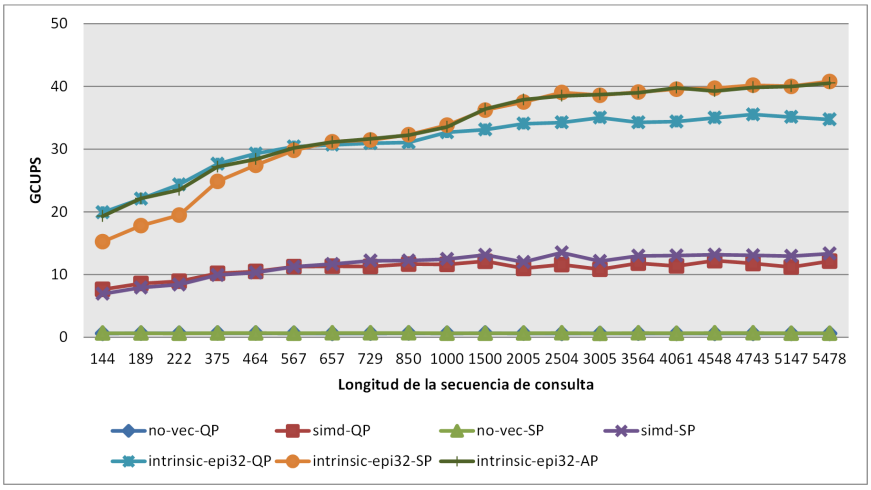

Figura 3.7: Rendimiento de las diferentes implementaciones en el Xeon Phi al variar el tamaño de la secuencia de consulta

para obtener buenos desempeños. Si bien el empaquetamiento de enteros de 8 bits no está disponible aun en el Xeon Phi, su incorporación se espera en la próxima generación de estos coprocesadores por medio de las extensiones AVX-512. Es posible emular esta capacidad mediante software empaquetando dos enteros de 16 bits en uno sólo de 32 bits. Sin embargo, las operaciones de enmascaramiento necesarias para mantener la coherencia del algoritmo y la detección de la condición de *overflow* por software anula las mejoras obtenidas.

En la Figura 3.8 se muestra una comparación de rendimiento entre las implementaciones SWIMM (versión Xeon Phi) y SWAPHI. En particular, se configuró SWAPHI para que emplee el esquema inter-tarea con las estrategias QP y SP en forma individual. Se puede observar que la propuesta AP de SWIMM supera a SWAPHI para secuencias de consulta mediana y largas que precisamente son las más costosas en cuanto al tiempo de búsqueda requerido.

Finalmente, en la Figura 3.9 se analiza el impacto de la técnica de procesamiento por

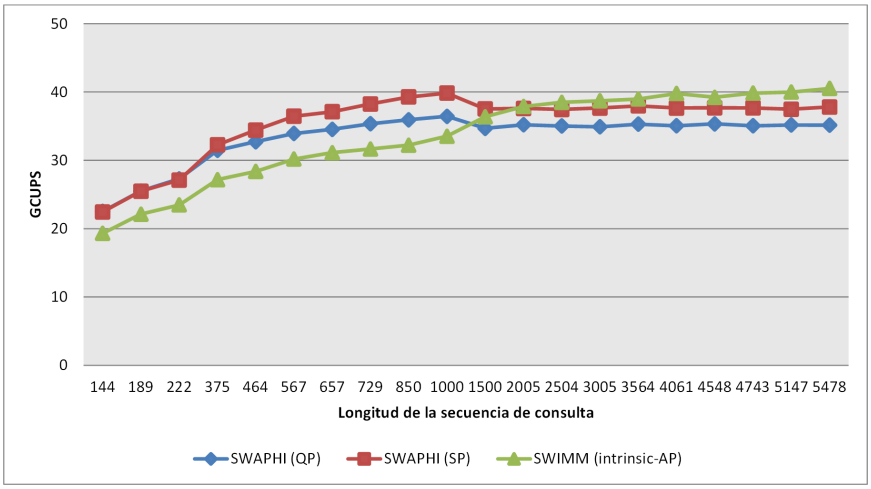

Figura 3.8: Comparación de rendimiento entre SWIMM y SWAPHI en el Xeon Phi

bloques para las configuraciones más exitosas: 32 hilos hardware y esquema SP en el Xeon y 228 hilos hardware con el esquema AP en el Xeon Phi. La técnica por bloques logra una mejora casi constante de 1.7× en la tasa de rendimiento, aun para las secuencias más cortas en el Xeon Phi. Sin embargo, los beneficios obtenidos en el Xeon por este modo de procesamiento son menores (un promedio de 1.1×), dado que este procesador tiene una memoria caché más grande que la correspondiente al Xeon Phi.

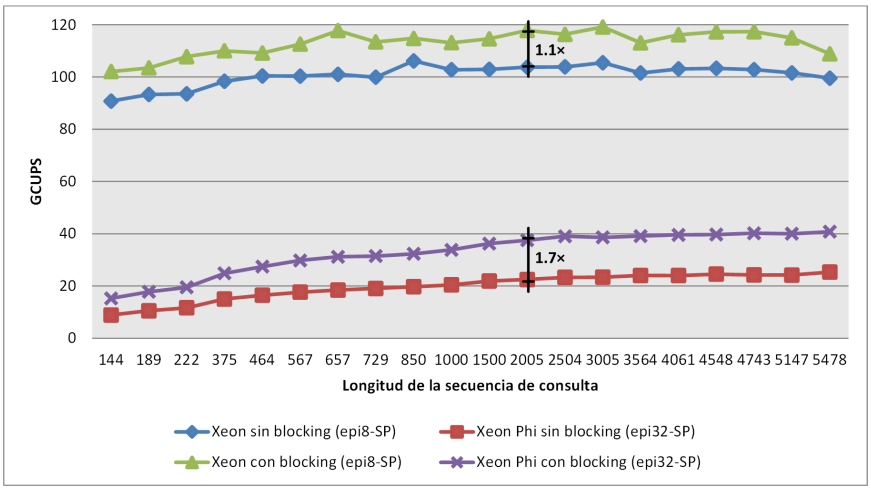

Figura 3.9: Impacto del procesamiento por bloques al variar el tamaño de la secuencia de consulta

3.2.4. Resultados de la implementación heterogénea híbrida

En esta sección se analiza el desempeño de la implementación híbrida que emplea el Xeon y el Xeon Phi en forma simultánea.

3.2.4.1. Distribución estática de la carga de trabajo

Una de las claves para mejorar el rendimiento en los sistemas heterogéneos consiste en lograr una carga de trabajo equilibrada entre la CPU y el acelerador. En la Figura 3.10 se muestra el rendimiento de la implementación híbrida con distribución estática al variar la carga de trabajo en ambos dispositivos utilizando las versiones intrínsecas con el esquema que dio mejor resultado (SP en el Xeon y AP en el Xeon Phi). La abscisa indica el porcentaje de residuos de la base de datos que son alineadas en el Xeon, mientras que el porcentaje restante es alineado en el Xeon Phi. El máximo rendimiento se logra asignando un 75 % de la carga de trabajo al Xeon y un 25 % en el Xeon Phi. Más allá de emplear un esquema simple para distribuir el trabajo, el rendimiento alcanza los 160 GCUPS. Este valor se corresponde casi con la suma de los rendimientos individuales en el Xeon y en el Xeon Phi (117.8 y 41.9, respectivamente), por lo que se puede afirmar que el *overhead* causado por esta distribución de la carga de trabajo no incide significativamente en el rendimiento obtenido.

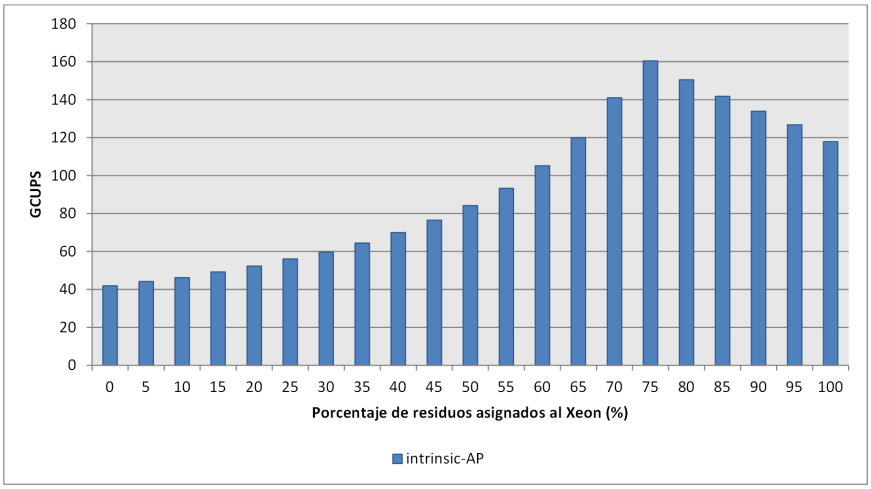

Figura 3.10: Rendimiento de la implementación híbrida al variar el porcentaje de secuencias de la base de datos que es procesado por el Xeon

3.2.4.2. Distribución dinámica de la carga de trabajo

En la Figura 3.11 se exhibe una comparación de rendimiento entre las implementaciones SWIMM (versión híbrida) y XSW 2.0. Al igual que en el experimento anterior, se emplearon los esquemas SP y AP en el Xeon y en el Xeon Phi respectivamente, aunque este caso se utilizó la estrategia de distribución dinámica. Ambas implementaciones se probaron utilizando todos los recursos disponibles: 32 hilos en el Xeon E5-2670 y 228 en el Xeon Phi. Como se puede ver, SWIMM supera el desempeño de XSW 2.0 ampliamente para todos los tamaños de consulta considerados, logrando aceleraciones de hasta 3.9×.

3.2.5. Comparación de rendimiento con otras implementaciones SW

En esta sección se realiza una comparación de rendimiento entre SWIMM y otras implementaciones SW disponibles. En la Figura 3.12 se muestra el rendimiento logrado por las diferentes implementaciones SW en el sistema equipado con procesadores Intel Xeon E5-2695 v3. Como se esperaba, SWIMM obtuvo las mejores tasas de rendimiento (300-380 GCUPS),

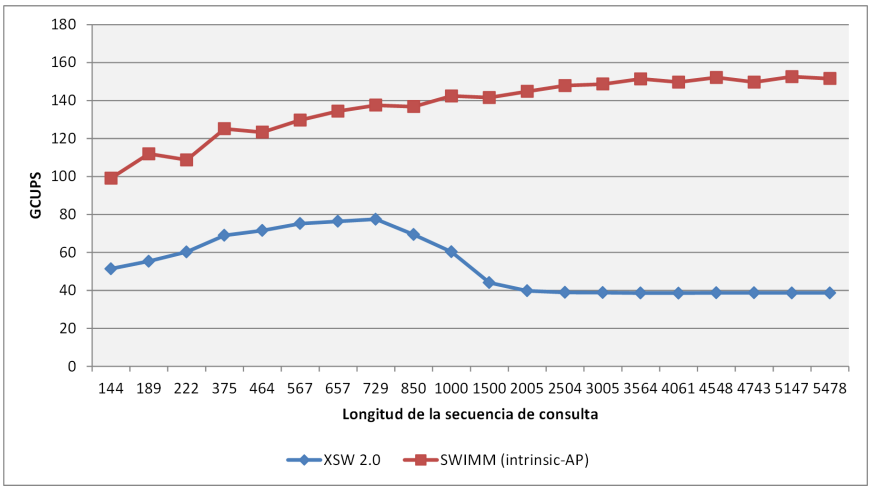

Figura 3.11: Comparación de rendimiento entre SWIMM y XSW al variar el tamaño de la secuencia de consulta

aunque la mayor parte de la aceleración proviene de la explotación de paralelismo a nivel de datos mediante el uso de las AVX2 (315-360 GCUPS). Resulta importante mencionar que, si bien el uso de computación heterogénea prevee mejorar las tasas de prestaciones, la gran diferencia entre los rendimientos individuales del Xeon y del Xeon Phi implica un serio desbalance de carga (los hilos de la CPU se encuentran ociosos) que se traduce en una disminución significativa del desempeño global. Al comparar el rendimiento obtenido con las implementaciones de referencia seleccionadas, se observa que CUDASW++ 3.0 alcanza hasta 300 GCUPS, SWIPE logra un pico de 260 GCUPS y XSW 2.0 exhibe un desempeño pobre. Por último, también se puede notar un rendimiento limitado del coprocesador Xeon Phi al alcanzar un pico de 50 GCUPS.

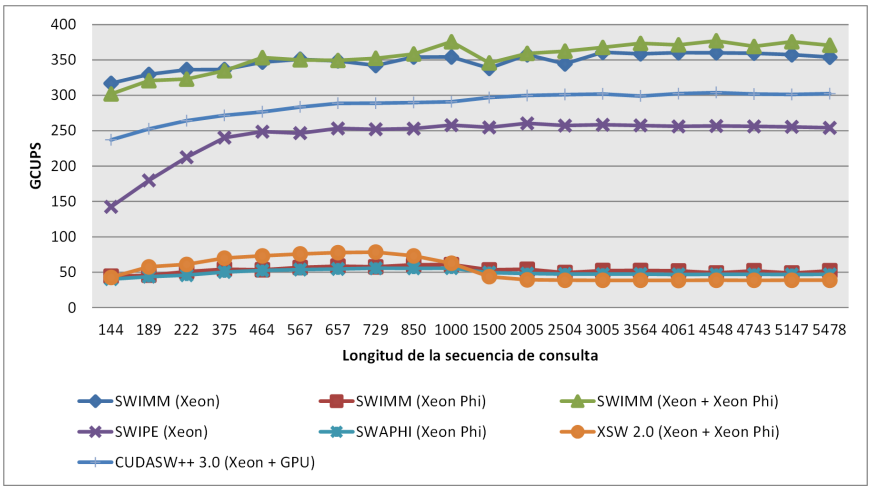

Figura 3.12: Comparación de rendimiento entre diferentes implementaciones SW al variar el tamaño de la secuencia de consulta

3.3. Resumen y conclusiones

En este capítulo, se presentó la herramienta SWIMM para búsquedas de similitud en sistemas heterogéneos basados en aceleradores Xeon Phi, la cual trabaja con bases de datos biológicas reales y permite configurar los parámetros de ejecución. Además, admite tres modos de ejecución: (1) cómputo en el *host*, (2) cómputo en los coprocesadores y (3) cómputo concurrente en el *host* y los coprocesadores. Mediante esta herramienta, se estudió y evaluó la viabilidad de esta clase de sistemas heterogéneos para el alineamiento de secuencias de proteínas mediante el algoritmo SW.

En primer lugar, considerando la compatibilidad de código entre los procesadores Xeon y los coprocesadores Xeon Phi, se desarrolló una implementación común para ambos dispositivos siguiendo el enfoque inter-tarea. La propuesta explota dos niveles de paralelismo: a nivel de hilo mediante OpenMP y a nivel de datos usando vectorización guiada. Luego, se desarrollaron versiones alternativas para cada dispositivo que reemplazan la vectorización guiada por vectorización manual a través de funciones intrínsecas. En todos los casos se estudiaron y aplicaron diferentes técnicas de programación y optimización a las distintas implementaciones; entre ellas, la selección de puntajes de sustitución mediante las estrategias QP y SP, la explotación de enteros de bajo rango, el desenrollado de bucles, el procesamiento por bloques y la alineación de accesos a memoria. Finalmente, utilizando los códigos desarrollados para ambos dispositivos se implementó una versión híbrida capaz de computar alineamientos en el Xeon y en el Xeon Phi de forma simultánea. Esta implementación admite dos modos de distribución de la carga de trabajo (estática y dinámica).

Posteriormente, se seleccionaron secuencias de consulta y bases de datos de proteínas representativas para la comunidad científica que sirvieran como *benchmarks*. En particular, se eligieron 20 secuencias de consulta de variada longitud y las bases de datos de datos Swiss-Prot y Environmental NR, las cuales fueron utilizadas conjuntamente en investigaciones similares anteriormente. Además, se seleccionaron otras implementaciones de referencia con las cuales poder realizar comparaciones: SWIPE para CPU, SWAPHI para Xeon Phi, XSW 2.0 para CPU-Xeon Phi y CUDASW++ para CPU-GPU.

Se probaron las distintas implementaciones sobre dos arquitecturas heterogéneas diferentes. A continuación se detallan las principales conclusiones del análisis experimental:

- Para lograr implementaciones SW de alto rendimiento tanto en los Xeon como en los Xeon Phi, se adaptó el código con el fin de:
  - Explotar dos niveles de paralelismo: usar más hilos no resulta suficiente; se debe explotar paralelismo de datos también. En ese sentido, la vectorización guiada ofrece portabilidad de código y un menor costo de programación. Sin embargo, la mejora de rendimiento que se puede lograr es pequeña comparada a la de la vectorización manual.
  - Aprovechar el hecho de que los puntajes de similitud no requieren de un amplio rango de representación. Mediante la explotación de tipos de dato enteros de bajo rango es posible aumentar el paralelismo a nivel de datos (más elementos por vector). En particular, se mostró que resulta conveniente computar los alineamientos usando enteros de 8 bits en primera instancia y utilizar progresivamente representaciones más amplias cuando sea necesario.
  - Utilizar la estrategia SP en el Xeon para obtener los puntajes de sustitución. En el Xeon Phi, la estrategia más eficiente depende de la longitud de la secuen-

cia consulta, por lo que el esquema adaptativo propuesto (AP) soluciona este inconveniente.

- Emplear técnicas de procesamientos por bloques que mejoren la localidad de datos, especialmente en el Xeon Phi considerando que su memoria caché es más pequeña.
- Lograr una carga de trabajo balanceada entre el Xeon y el Xeon Phi en versiones híbridas. Una diferencia de potencia computacional (en GCUPS) muy grande entre ambos dispositivos puede llevar a un serio desbalance de carga, lo que se traslada a una reducción del rendimiento.

- Con respecto a los GCUPS, la implementación con intrínsecas SSE logró 118 GCUPS en el Xeon E5-2670 con 32 hilos mientras que su equivalente con AVX2 alcanzó 355 GCUPS en el Xeon E5-2695 v3 con 56 hilos. Aun así, la implementación con KNC sólo obtuvo un máximo de 50 GCUPS para 228 hilos. El pobre desempeño del Xeon Phi se debe a la ausencia de capacidades vectoriales para explotar enteros de menos de 32 bit.

- En cuanto a la comparación con otras implementaciones SW, la versión SSE de SWIMM se mostró equiparable con SWIPE mientras que la versión AVX2 logró superarlo ampliamente alcanzado diferencias de hasta 1.4×. Cabe destacar que, hasta donde llega el conocimiento del tesista, esta es la primera implementación SW para el conjunto de instrucciones AVX2. Por otra parte, la versión KNC de SWIMM se mostró competitiva con SWAPHI, siendo capaz de superarlo para secuencias medianas y largas. Complementariamente, la versión híbrida de SWIMM con distribución dinámica (SSE+KNC) superó notablemente a su alternativa XSW 2.0, con mayores diferencias para secuencias de consulta largas. Por último, en el sistema basado en Xeon E5-2695 v3, SWIMM (versión AVX2+KNC) logró mejores rendimientos que CUDASW++ 3.0 utilizando los recursos del *host* y una GPU NVIDIA K20c.

Para concluir, resulta importante recordar que la próxima generación de coprocesadores Xeon Phi (Knights Landing) unificará su conjunto de instrucciones con la generación actual de procesadores Xeon (Skylake) adoptando las extensiones AVX-512 de 512 bits. Como este conjunto de instrucciones sí posee capacidades para explotar enteros de bajo rango, se espera un incremento notable en el rendimiento de ambos dispositivos usando la metodología propuesta.

Capítulo 4

Optimización de SW para sistemas heterogéneos basados en aceleradores FPGA: OSWALD

Aunque al día de hoy no existen sistemas HPC basados en FPGAs que aparezcan en el TOP500, el uso de este tipo de aceleradores aumenta en diferentes áreas debido al crecimiento de su capacidad computacional, su bajo consumo energético y al desarrollo de herramientas que disminuyen su costo de programación. Este capítulo introduce la herramienta OSWALD para búsquedas biológicas en sistemas heterogéneos basados en aceleradores FPGA, la cual trabaja con bases de datos biológicas reales y permite configurar los parámetros de ejecución. Mediante esta herramienta se estudia y evalúa la viabilidad de esta clase de sistemas heterogéneos para el alineamiento de secuencias de proteínas desde la perspectiva del rendimiento. En primer lugar, se describen las diferentes implementaciones propuestas junto a las técnicas de programación y optimización empleadas (Sección 4.1). Luego, se detallan los distintos experimentos llevados a cabo y se analizan los resultados obtenidos (Sección 4.2). Por último, la Sección 4.3 presenta un resumen del capítulo.

4.1. Implementación

En esta sección se detallan los aspectos de programación y optimización considerados para las implementaciones sobre plataformas basadas en FPGA. A diferencia de las implementaciones disponibles en la literatura, se decidió explorar los beneficios de utilizar una tecnología innovadora, como lo es OpenCL en el ámbito de las FPGAs, en lugar de HDLs tradicionales como VHDL o Verilog. En primer lugar, se propone una implementación heterogénea donde los alineamientos se llevan a cabo en una única FPGA. Luego, se extiende la implementación anterior para que soporte ejecución en más de una FPGA. La implementación final explota cómputo en el *host* y en las FPGAs de forma concurrente. Al igual que las implementaciones del capítulo anterior, los algoritmos comprenden tres etapas:

1. Etapa de preprocesamiento: las secuencias de la base de datos son preprocesadas de forma de adaptarlas para su posterior procesamiento paralelo en múltiples dispositivos.

2. Etapa Smith-Waterman: una vez preprocesada la base de datos, se computan los alineamientos mediante el método SW.

3. Etapa de ordenación: finalmente, los puntajes de similitud son ordenados en forma descendente.

Es importante destacar que las etapas 1 y 3 se llevan a cabo en el *host* en ambas implementaciones. La etapa 2 se ejecuta en una o más FPGAs en el caso de la implementación heterogénea y concurrentemente en el *host* y en las FPGAs en la versión híbrida.

4.1.1. Esquema de paralelismo

Los alineamientos son computados empleando el esquema de paralelismo inter-tarea descrito en la Sección 2.3.1 y utilizado en las implementaciones del capítulo anterior. En lugar de alinear una secuencia de la base de datos con la secuencia de consulta a la vez, múltiples secuencias de la bases de datos son alineadas con la secuencia de consulta al mismo tiempo mediante la utilización de las capacidades vectoriales del dispositivo donde se procese. Por este motivo, las secuencias de la base de datos son procesadas en grupo y el tamaño de los grupos está determinado por el número de canales del vector. En el caso del *host*, el tamaño de los grupos lo define el ancho de la VPU. En el caso de las FPGAs, es posible configurar sus recursos para determinar el número de elementos que se procesan a la vez.

4.1.2. Preprocesamiento de la base de datos

Al igual que ocurre con SWIMM, resulta necesario preprocesar la base de datos antes de poder utilizarla. Las secuencias de la base de datos son ordenadas de acuerdo a su longitud (en forma ascendente) antes de ser agrupadas y completadas con símbolos nulos (*dummy*) para que todas sean del mismo tamaño. Este paso favorece el patrón de acceso a la memoria y minimiza los desbalances en el procesamiento de los grupos de secuencias. En las implementaciones para FPGAs, se divide la base de datos en *chunks* teniendo en cuenta que la memoria global de estos dispositivos es escasa. Además, el número de *chunks* es múltiplo del número de FPGAs involucradas y se procura que los *chunks* sean aproximadamente del mismo tamaño para lograr una carga de trabajo equilibrada entre los aceleradores involucrados.

Resulta importante mencionar que en la implementación híbrida, la base de datos se divide en dos partes principales para favorecer a una distribución balanceada del trabajo. Esta estrategia se describe más adelante en la Sección 4.1.5.1. La parte correspondiente al *host* se mantiene como una única porción mientras que la correspondientes a las FPGAs se divide en múltiples *chunks*, siguiendo el enfoque de las implementaciones FPGA. Como este proceso debe ser repetido para cada búsqueda, las bases de datos son preprocesadas en forma separada para evitar trabajo repetido. Las bases de datos se leen en formato FASTA y luego son transformadas a un formato binario interno, el cual facilita una lectura ulterior más rápida.

4.1.3. Implementación heterogénea para una FPGA

El Algoritmo 4.1 exhibe el pseudo-código de la implementación en el *host*. La administración de los *buffers* en las FPGAs se realiza en OpenCL mediante las operaciones `clCreateBuffer` (reserva de memoria e inicialización), `clEnqueueWriteBuffer` (transferencias hacia el dispositivo) y `clEnqueueReadBuffer` (transferencias desde el dispositivo). Cada *kernel* computa los alineamientos entre una única secuencia de consulta y las se-

cuencias de un *chunk* de la base de datos. La invocación al *kernel* se realiza mediante `clEnqueueNDRangeKernel`.

Algoritmo 4.1 Pseudo-código del *host* para la implementación que emplea una FPGA

▷ Q son las secuencias de consulta
▷ vD es la base de datos preprocesada
▷ SP son los Score Profiles
▷ SM es la matriz de sustitución
▷ S son los puntajes de alineamiento
▷ n son las longitudes de las secuencias de la base de datos
▷ t_h es el número de hilos en el *host*

clCreateBuffer's(...) ▷ Crear *buffers* e inicializar los que correspondan
SetKernelArg's(...) ▷ Establecer argumentos comunes para todos los alineamientos
for $c <$ *obtener_número_de_chunks(vD)* **do**
  SP_c = *construir_SPs*(vD_c, SM, t_h)
  clEnqueueWriteBuffer(SP_c) ▷ Transferir *SPs* a la FPGA
  clEnqueueWriteBuffer(n_c) ▷ Transferir longitudes a la FPGA
  clSetKernelArg's(...) ▷ Establecer argumentos para el chunk c
  for $q <$ *obtener_número_de_secuencias(Q)* **do**
    clSetKernelArg's(...) ▷ Establecer argumentos para la secuencia de consulta q
      clEnqueueNDRangeKernel(...) ▷ Computar alineamientos entre la secuencia de consulta q y el chunk c
  end for
  clEnqueueReadBuffer(S_c) ▷ Transferir puntajes al *host*
end for
S = *recomputar_por_overflow*(S, Q, vD, SM, t_h) ▷ Recomputar alineamientos en caso de overflow
S = *ordenar*(S, t_h) ▷ Ordenar puntajes en sentido descendente

El *kernel* se implementa siguiendo el modelo de programación paralela de tarea descrito en la especificación 1.0 de OpenCL, donde el *kernel* consiste de un sólo *work-group* que contiene un único *work-item*. Este esquema resulta conveniente ya que un sólo *work-item* no requiere ninguna fase de sincronización. En el Algoritmo 4.2 se muestra el pseudo-código del *kernel* implementado. La matriz de alineamiento se divide en bloques verticales, los cuales son computados fila a fila (ver Figura 4.1). Esta técnica de procesamiento por bloques no sólo mejora la localidad de datos sino que también reduce los requerimientos de memoria para computar un bloque, lo que favorece el uso de memoria privada de baja latencia. El desenrollado del bucle más interno permite aumentar el número de operaciones por ciclo de reloj y lograr acceso coalescente a memoria, lo que repercute en un incremento del rendimiento. Para ello es necesario que las secuencias sean extendidas en la etapa de preprocesamiento hasta tener una longitud múltiplo de una constante BLOCK_WIDTH, ya que el compilador requiere conocer la cantidad de iteraciones en tiempo de compilación.

Adicionalmente, se emplea la extensión de canales provista por Altera para resolver las dependencias de datos entre los bloques (para procesar el bloque i se requiere la última columna de H y de E del bloque *i-1*). La combinación de estas técnicas permite que el compilador de Altera genere un *pipeline* de ejecución paralela en forma exitosa.

Algoritmo 4.2 Pseudo-código del kernel SW

```
#pragma OPENCL_EXTENSION cl_altera_channels: enable

▷ numSecuencias es el número de secuencias de la base datos
▷ q es la secuencia de consulta
▷ m es la longitud de la secuencia de consulta

__attribute__((reqd_work_group_size(1,1,1)))
__attribute__((task))
__kernel void SW_kernel (numSecuencias, n, SP_c, q, m, S)
{
for s < numSecuencias do ▷ Para cada secuencia del chunk
   numBloques ← n[s] / BLOCK_WIDTH
   for k < numBloques do
      for i < m do ▷ Para cada fila
         if k ≠ 0 then
            j ← k×BLOCK_WIDTH
            H_{i-1,j-1} ← read_channel(H_channel)
            E_{i,j} ← read_channel(E_channel)
         end if
         #pragma unroll
         for jj < BLOCK_WIDTH do
            j ← k×BLOCK_WIDTH + jj
            H_{i,j} ← SW_core(H, E, F, SP[s], q)
            score ← max(score, H_{i,j})
         end for
         if k ≠ numBloques-1 then
            j ← (k+1)×BLOCK_WIDTH-1
            write_channel(H_channel, H_{i-1,j})
            write_channel(E_channel, E_{i,j+1})
          end if
       end for
      S[s] ← score ▷ Guardar puntaje de alineamiento
   end for
}
```

4.1.3.1. Selección de puntajes de sustitución

Para obtener los puntajes de la matriz de sustitución al momento de comparar los residuos de las diferentes secuencias se utiliza la técnica SP descrita en la Sección 2.3.2.1. Los SPs son construidos en el *host* empleando uno más o hilos que utilizan un conjunto de funciones intrínsecas SSE (como se realizó en la implementación de la Sección 3.1.3) y son transferidos luego al acelerador correspondiente, lo que permite reducir la utilización de recursos de hardware en las FPGAs.

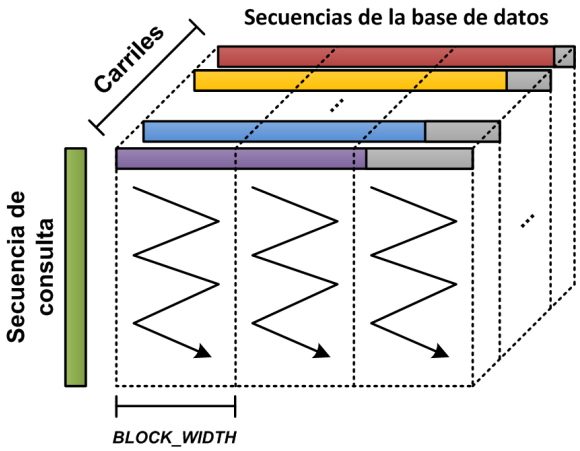

Figura 4.1: Representación esquemática del *kernel* OpenCL.

4.1.3.2. Selección del tipo de dato entero

Como se observó en la Sección 3.2.2, los puntajes de alineamiento no requieren una representación entera amplia. En particular, se mostró que más del 99 % de los puntajes obtenidos pueden ser representados con 8 bits mientras que para el resto de los puntajes resulta suficiente emplear 16 bits. Dado que el tamaño del tipo de dato a emplear en la FPGA está relacionado con el consumo de los recursos disponibles, se exploraron diferentes representaciones de tipos de dato entero (8, 16 y 32 bits). Cuando el tipo de dato resulta insuficiente para el puntaje de alineamiento, es decir que ocurre *overflow*, se recomputa el alineamiento en el *host* empleando el siguiente tipo de dato más amplio. El código del *host* emplea operaciones vectoriales disponibles en los procesadores de Intel como SSE o AVX2 y está basado en la implementación presentada en la Sección 3.1.3. Como los *kernels* emplean suma con saturación (en particular, la función `add_sat`), el *host* es capaz de detectar los casos potenciales de *overflow* al localizar los puntajes de alineamiento computados en la FPGA que son iguales al máximo valor de la representación entera empleada.

4.1.3.3. Buffers en el *host* y transferencias de datos

Los *buffers* en el *host* son alineados a 64 bytes al momento de la reserva de memoria, lo que permite mejorar la eficiencia en las transferencias entre el *host* y las FPGAS ya que se activa el mecanismo Acceso Directo a Memoria. Los datos que son comunes a todos los alineamientos, como las secuencias de consultas, son transferidos al momento de crear los *buffers*.

4.1.4. Implementación heterogénea multi-FPGA

Una estrategia simple para extender la solución de la Sección 4.1.3 y emplear más de una FPGA consiste en explotar paralelismo a nivel de hilo en el *host*. Al igual que en la implementación heterogénea de la Sección 3.1.3, se debe generar un hilo OpenMP por cada acelerador utilizado. Luego, los *chunks* de la base de datos se distribuyen entre las diferentes FPGAs utilizando una directiva `parallel for`. Lamentablemente esta estrategia no es viable ya que la librería para OpenCL de Altera no es *thread-safe* a nivel de *host* [88]. Para reemplazar esta limitación y posibilitar la ejecución simultánea de múltiples FPGAs,

las transferencias entre el *host* y las FPGAs al igual que las invocaciones a los kernels son realizadas en forma no bloqueante. Como algunas de estas operaciones requieren que otras hayan terminado para poder ejecutarse, se utiliza la función `clFinisih` para sincronizar el *host* con los dispositivos. El Algoritmo 4.3 muestra el pseudo-código del *host* en esta implementación. Con respecto al *kernel* OpenCL, el código no se ve afectado por los cambios aplicados en el *host*.

4.1.5. Implementación heterogénea híbrida

Es factible aprovechar la potencia de cómputo del *host* en simultáneo con el trabajo de las FPGAs mediante la explotación de paralelismo a nivel hilo. En el Algoritmo 4.4 se exhibe el pseudo-código del *host* en esta implementación. Inicialmente se generan sólo dos hilos OpenMP, donde un hilo será responsable de computar en el *host* mientras que el otro se encargará de delegar el cómputo a las FPGAs utilizadas. Los alineamiento en la CPU son computados en la función *búsqueda_ SWIMM* de acuerdo a la implementación descrita en el Algoritmo 3.1, la cual es capaz de explotar multihilado y las instrucciones vectoriales SSE y AVX2. La función *búsqueda_ multi-FPGA* computa los alineamientos como se describió en el Algoritmo 4.3.

4.1.5.1. Estrategia de distribución del trabajo

Un factor clave para obtener alto rendimiento es una carga de trabajo balanceada entre el *host* y los aceleradores. Como se explicó en el capítulo anterior, las técnicas estáticas pueden llevar a una distribución *perfecta*. Sin embargo, estas técnicas usualmente requieren conocer algún tipo de información de antemano, como las características de hardware relacionados a las capacidades computacionales, la jerarquía memoria, entre otras. En sentido contrario, las técnicas dinámicas no necesitan de ninguna información previa pero suelen sufrir penalizaciones de rendimiento debido a desbalances u ocio. Para resolver este problema, se emplea una técnica semi-dinámica que toma lo mejor de ambos enfoques: un subconjunto de las secuencias de la base de datos son utilizadas para estimar la capacidad computacional de cada dispositivo.

Las secuencias de consulta y un porcentaje configurable de residuos de la base de datos (p) son utilizados para estimar las potencias de cómputo relativas entre el *host* y los aceleradores. El número de residuos de la base de datos asignados a las FPGAs (R_F) es evaluado de acuerdo a la Ecuación 4.1:

$$R_F = |D| \times \frac{n_F \times GCUPS_F}{n_F \times GCUPS_F + GCUPS_h} \tag{4.1}$$

donde $|D|$ es el número total de residuos en la base de datos, n_F es el número de FPGAs, y $GCUPS_F$ y $GCUPS_h$ corresponden a los GCUPS obtenidos por la FPGA y el *host*, respectivamente. t_h hilos son utilizados en el *host* para estimar su potencia de cómputo mientras que en el caso de las FPGAs sólo se emplea un único acelerador. Se asume que, en un sistema basado en FPGAs, todos los aceleradores poseen las mismas características. En un ambiente con FPGAs diferentes, se debería estimar la potencia de cómputo relativa de cada FPGA en forma individual de manera de distribuir la carga de trabajo de manera más balanceada.

Algoritmo 4.3 Pseudo-código del *host* para la implementación que emplea más de una FPGA

▷ n_F es el número de FPGAs

S = *búsqueda_multi-FPGA* (Q, vD, SM, t_h, n_F) ▷ Computar alineamientos en FPGAs
S = *recomputar_por_overflow*(S, Q, vD, SM, t_h) ▷ Recomputar alineamientos en caso de overflow
S = *ordenar*(S, t_h) ▷ Ordenar puntajes en sentido descendente

function *búsqueda_multi-FPGA* (Q, vD, SM, t_h, n_F) {
  for $d < n_F$ **do**
    clCreateBuffer's(...) ▷ Crear *buffers* en la FPGA d e inicializar los que correspondan
    clSetKernelArg's(...) ▷ Establecer argumentos comunes para todos los alineamientos
  end for
  for (i=0; $i <$ *obtener_número_de_chunks*(vD); i+=n_F) **do**
    for $d < n_F$ **do**
      $c \leftarrow i + d$
      SP_c = *construir_SPs*(vD_c, SM, t_h)
      clEnqueueWriteBuffer(SP_c) ▷ Transferir *SPs* a la FPGA d
      clEnqueueWriteBuffer(n_c) ▷ Transferir longitudes a la FPGA d
      clSetKernelArg's(...) ▷ Establecer argumentos para el chunk c
    end for
    for $d < n_F$ **do** ▷ Sincronizar host con FPGAs
      clFinish(...)
    end for
    for $d < n_F$ **do**
      $c \leftarrow i + d$
      for $q \leq$ *obtener_número_de_secuencias*(Q) **do**
        clSetKernelArg's(...) ▷ Establecer argumentos para la secuencia de consulta q
        clEnqueueNDRangeKernel(...) ▷ Computar alineamientos entre la secuencia de consulta q y el chunk c
      end for
    end for
    for $d < n_F$ **do** ▷ Sincronizar host con FPGAs
      clFinish(...)
    end for
    for $d < n_F$ **do**
      $c \leftarrow i + d$
      clEnqueueReadBuffer(S_c) ▷ Transferir puntajes al *host*
    end for
  end for
  return S
end function

Algoritmo 4.4 Pseudo-código del *host* para la implementación heterogénea híbrida

▷ p es el porcentaje de la base de datos usado para calcular R_F
▷ R_F es el número de residuos asignados a las FPGAs
▷ vD_p es el chunk de la base datos usado para estimar las potencias de cómputo relativas
▷ vD_h es la parte de la base de datos correspondiente al *host*
▷ vD_F es la parte de la base de datos correspondiente a las FPGAs

vD_p = *extraer_chunk* (vD, p) ▷ Extraer *chunk* de la base de datos para estimar potencias de cómputo relativas
[R_F, S] = *estimar_potencias_de_cómputo* (Q, vD_p, SM, t_h) ▷ Calcular R_F invocando a la función *búsqueda_híbrida*
[vD_h, vD_F] = *dividir_trabajo* (vD, p, R_F, n_F) ▷ Dividir base de datos preprocesada
S = *búsqueda_híbrida* (Q, vD_h, vD_F, SM, t_h, n_F) ▷ Computar alineamientos en CPU y FPGAs
S = *ordenar*(S, t_h)

function *búsqueda_híbrida* (Q, vD_h, vD_F, SM, t_h, n_F) {
  #pragma omp parallel num_threads(2)
  {
    #pragma omp single nowait
    { S_F = *búsqueda_multi-FPGA* (Q, vD_F, SM, 1, n_F) }
    #pragma omp single
    { S_h = *búsqueda_SWIMM* (Q, vD_h, SM, t_h, n_F) }
  }
  S_F = *recomputar_por_overflow*(S_F, Q, vD_F, SM, t_h)
  return S
end function

4.2. Resultados experimentales

4.2.1. Entorno y diseño experimental

Todas las pruebas fueron realizadas sobre dos arquitecturas heterogéneas con sistema operativo CentOS 6.5. El primero posee dos procesadores Intel Xeon E5-2670 2.60GHz de ocho núcleos (dos hilos hardware por núcleo) y 32 GB de memoria RAM mientras que el segundo cuenta con dos procesadores Intel Xeon E5-2695 v3 2.60GHz de 14 núcleos (dos hilos hardware por núcleo) y 64 GB de memoria RAM. Los dos sistemas están equipados con:

- Dos placas Altera Stratix V GSD5 Half-Length PCIe con 8 GB de memoria RAM Dual DDR3 (dos bancos de 4 GB DDR3).
- Una GPU NVIDIA Tesla K20c (2496 núcleos CUDA) con 5 GB de memoria dedicada y Capacidad Computacional 3.5.
- Un coprocesador Intel Xeon Phi 3120P de 57 núcleos (cuatro hilos hardware por núcleo) con 6 GB de memoria dedicada.

El compilador utilizado es Intel ICC (versión 15.0.2) con nivel de optimización O3 habilitado por defecto. La herramienta para la síntesis utilizada es Quartus II DKE V12.0.2 con OpenCL

SDK v14.0 Los hilos OpenMP fueron mapeados a los núcleos del procesador mediante la afinidad `scatter`. La versión del SDK de CUDA es 7.0.

Los experimentos utilizados para evaluar rendimiento son los mismos que los descritos en la Sección 3.2.1. Para facilidad del lector se vuelven a detallar. Se evaluó el rendimiento de las implementaciones propuestas buscando 20 secuencias de proteínas en dos bases de datos reconocidas: Swiss-Prot (versión 2013_11) y Environmental NR (versión 2014_11). La base de datos Swiss-Prot consiste de 192480382 aminoácidos en 541561 secuencias, siendo la más larga de 35213 residuos. La base de datos Environmental NR se compone de 1291019045 residuos en 6552667 secuencias, siendo la longitud máxima igual a 7557. Las secuencias de consulta fueron extraídas de la base de datos Swiss-Prot (números de acceso: P02232, P05013, P14942, P07327, P01008, P03435, P42357, P21177, Q38941, P27895, P07756, P04775, P19096, P28167, P0C6B8, P20930, P08519, Q7TMA5, P33450, y Q9UKN1) y sus longitudes varían entre 144 y 5478 residuos. La matriz de sustitución seleccionada es BLOSUM62 y los puntajes de inserción y extensión de *gaps* fueron establecidos en 10 y 2, respectivamente.

En forma adicional, se realizó una comparación de rendimiento con implementaciones para otros tipos de aceleradores. Por ese motivo, se seleccionaron las implementaciones que presentan el mejor rendimiento en sistemas basados en CPU, en coprocesadores Xeon Phi y en GPUs: SWIMM fue elegido para los procesadores Xeon y Xeon Phi mientras que CUDASW++ 3.0 (versión 3.1) fue escogido para las GPUs compatibles con CUDA. Lamentablemente no fue posible realizar una comparación con otras implementaciones FPGA. Aunque sólo tres de las ocho propuestas FPGA analizadas en la Sección 2.3.2.3 son completas desde el punto de vista funcional, ninguna de ellas tiene su código disponible. Si bien un análisis teórico aun es posible, los diferentes dispositivos, tecnologías y enfoques empleados no sólo complican una comparación directa sino que también dificultan hacerla en forma justa.

Es importante mencionar que los experimentos para las implementaciones con una y múltiples FPGAs fueron realizados en el sistema basado en procesadores Xeon E5-2695 v3 utilizando 28 hilos OpenMP. El resto de los experimentos incluyen ambas arquitecturas. Con respecto a las bases de datos, para los experimentos de las implementaciones con una y múltiples FPGAs se utilizó la base de datos Swiss-Prot. Sin embargo, debido a su tamaño limitado, se empleó Environmental NR para complementar los experimentos de la implementación multi-FPGA y para realizar una comparación de rendimiento entre la versión híbrida CPU-FPGA y otras implementaciones SW. En relación a los porcentajes de la base de datos utilizados para estimar las potencias de cómputo relativas entre *host* y FPGAs, se utilizó 1 % para el sistema basado en procesadores Xeon E5-2670 y 2 % para el sistema basado en procesadores Xeon E5-2695 v3.

Por último, las implementaciones desarrolladas han sido integradas en una única herramienta, a la cual se ha llamado OSWALD y que se encuentra disponible en un repositorio web público[1].

4.2.2. Resultados de la implementación para una FPGA

Como se mencionó en la Sección 2.3.2, se utiliza la métrica GCUPS para evaluar el rendimiento de las implementaciones SW. Para el cálculo de los GCUPS de las implementaciones descritas en este capítulo se consideró dentro del tiempo de ejecución: la creación de

[1] OSWALD se encuentra disponible *online* en https://github.com/enzorucci/OSWALD

buffers, las transferencias del *host* a la(s) FPGA(s), el cómputo de los alineamientos y las transferencias de la(s) FPGA(s) al *host*.

De forma de poder evaluar el rendimiento en la FPGA, se consideran diferentes implementaciones según el grado de paralelismo de datos y la explotación de la jerarquía de memoria. A continuación se detallan las diferencias entre las implementaciones:

- La versión *escalar* es la implementación base que no emplea paralelismo de datos y que servirá como punto de partida.
- Las versiones SIMD emplean diferentes representaciones de datos enteros y explotan paralelismo de datos al utilizar vectorización. La nomenclatura vectorial indica el ancho del vector: *char4* emplea vectores de 4 elementos, mientras que *char8* y *char16* usan 8 y 16 elementos respectivamente. Por otro lado, el prefijo en el nombre indica el tipo de datos entero utilizado; es decir *int*, *short* y *char* representan tipos de datos enteros de 32, 16 y 8 bits, respectivamente.
- Con respecto a la explotación de memoria, la versión *Q constante* hace referencia al uso de memoria constante para la secuencia de consulta mientras que la versión *Q privada* copia la secuencia de consulta de memoria global a memoria privada antes de computar los alineamientos. Las dos versiones buscan aprovechar la baja latencia de acceso de ambas memorias.

La Tabla 4.1 presenta el uso de recursos y el rendimiento alcanzado para *kernels* OpenCL que utilizan diferentes tipos de dato entero. Los valores de las columnas ALMs, Regs, RAM y DSPs hacen referencia a los porcentajes de bloques lógicos configurables, de registros, de bloques de memoria y de bloques DSPs utilizados, respectivamente. Para este conjunto de experimentos se empleó el mismo valor de la constante BLOCK_WIDTH (en particular, se usó el valor 8 ya que el consumo de recursos por parte del kernel *int16* no permitió un valor mayor). Como se puede observar, la mejor opción resulta ser *char16* y no sólo desde el punto de vista del rendimiento sino también considerando el uso de recursos. *char16* reporta un incremento de 1.53× en el rendimiento y una reducción de 0-0.46× en el uso de recursos con respecto a *int16*. Aunque *int16* no requiere recalcular alineamientos en el *host*, los puntajes de similitud no necesitan de una representación entera muy amplia. Por lo tanto resulta conveniente computar los alineamientos en la FPGA usando enteros de 8 bits y recomputar en el *host* aquellos donde ocurra *overflow*.

Kernel	**GCUPS**	**Uso de recursos**				**Incremento del rendimiento**	**Decremento del uso de recursos**
		ALMs	**Regs**	**RAM**	**DSPs**		
int16	17.6	76%	39%	75%	1%	-	-
short16	22.7	54%	28%	48%	1%	1.29×	0-0.36×
char16	27.0	41%	24%	41%	1%	1.53×	0-0.46×

Tabla 4.1: Rendimiento y uso de recursos para implementaciones de *kernels* OpenCL con diferentes tipos de dato entero

La Tabla 4.2 presenta el uso de recursos y el rendimiento alcanzado para *kernels* OpenCL con diferente grado de vectorización. A diferencia de los experimentos correspondientes a la Tabla 4.1, se asignó el valor 16 a la constante BLOCK_WIDTH debido a un menor consumo de recursos por parte de estos kernels. Se puede apreciar que sin vectorizar las operaciones, el rendimiento es muy pobre, como lo demuestra la versión *escalar*. La explotación de paralelismo de datos mediante el uso de vectorización permite incrementar el rendimiento en

forma significativa. El mejor rendimiento lo obtiene la versión *char16*, la cual exhibe una aceleración de 11.9× a costo de un aumento de entre 0 y 2.5× en el uso de recursos.

Kernel	GCUPS	Uso de recursos				Incremento del rendimiento	Incremento del uso de recursos
		ALMs	Regs	RAM	DSPs		
escalar	4.0	28%	16%	28%	1%	-	-
char4	14.2	34%	18%	31%	1%	3.56×	0-1.21×
char8	27.7	43%	22%	38%	1%	6.95×	0-1.54×
char16	47.5	60%	30%	70%	1%	11.9×	0-2.5×

Tabla 4.2: Rendimiento y uso de recursos para implementaciones de *kernels* OpenCL con diferente grado de vectorización

Como se mencionó en la Sección 4.1.3, las matrices de alineamiento son procesadas de a bloques verticales. La constante BLOCK_WIDTH determina el número de bloques verticales de cada matriz. La Tabla 4.3 coteja el uso de recursos y el rendimiento alcanzado para diferentes valores de ancho de bloque de la implementación *char16*. Se puede apreciar en la tabla que el ancho de bloque no sólo impacta en el rendimiento sino también en el uso de recursos del dispositivo. Incrementar el ancho del bloque produce una mejora en el rendimiento pero también aumenta el uso de recursos, aunque la mejora en el rendimiento se reduce a medida que BLOCK_WIDTH se incrementa. Como las longitudes de las secuencias deben ser múltiplo de BLOCK_WIDTH, valores más grandes implican secuencias más largas y, como consecuencia, el *overhead* en el cómputo de los alineamientos se incrementa. El mejor rendimiento alcanza 57.3 GCUPS.

BLOCK_WIDTH	GCUPS	Uso de recursos				Incremento del rendimiento	Incremento del uso de recursos
		ALMs	Regs	RAM	DSPs		
4	11.7	40%	21%	36%	1%	-	-
8	27.0	41%	24%	41%	1%	2.31×	0-1.14×
12	37.9	53%	29%	46%	1%	3.24×	0-1.38×
16	47.5	60%	30%	70%	1%	4.06×	0-1.94×
20	52.5	75%	39%	74%	1%	4.49×	0-2.06×
24	55.0	82%	41%	81%	1%	4.7×	0-2.25×
28	57.3	92%	42%	88%	1%	4.9×	0-2.44×

Tabla 4.3: Rendimiento y uso de recursos para diferentes ancho de bloque en la versión *char16*

También se evaluó el impacto del uso de memoria constante y de memoria privada para las secuencias de consultas. La Tabla 4.4 muestra el rendimiento y el uso de recursos para diferente explotación de memoria en el kernel *char16*. Se puede observar que el uso de memoria constante para las secuencias de consulta no tiene el impacto esperado ya que reduce levemente el rendimiento en lugar de mejorarlo (*char16 + Q constante*). La memoria constante está optimizada para un acceso con una tasa alta de *cache hits*. Los residuos de la secuencia de consulta se utilizan para indexar el SP y se accede a un residuo por cada fila de un bloque vertical procesado. Como la memoria global incorpora hardware adicional para reducir el impacto de las latencias largas de memoria, se obtiene mejor rendimiento si las secuencias de consulta se transfieren directamente a esta memoria. Por otra parte, se puede ver que el empleo de memoria privada para las secuencias de consulta incrementa levemente el rendimiento con un aumento en el uso de recursos prácticamente despreciable (*char16 + Q privada*).

Por último, la Figura 4.2 ilustra el rendimiento de las diferentes versiones de la imple-

Kernel	GCUPS	Uso de recursos			
		ALMs	Regs	RAM	DSPs
char16	57.3	92 %	42 %	88 %	1 %
char16 + Q constante	57.2	92 %	41 %	88 %	1 %
char16 + Q privada	58.0	92 %	42 %	89 %	1 %

Tabla 4.4: Rendimiento y uso de recursos para diferente explotación de memoria en la versión *char16*

mentación para una FPGA al variar el tamaño de la secuencia de consulta. Como se puede ver, la versión *escalar* apenas mejora su rendimiento al incrementar el tamaño de la consulta. En cambio, los *kernels* que utilizan vectorización se ven beneficiados de una carga de trabajo más grande. Como se vio en el experimento anterior, la versión *char16 + Q privada* supera a todas las demás, en este caso alcanzando un pico de 52.9 GCUPS.

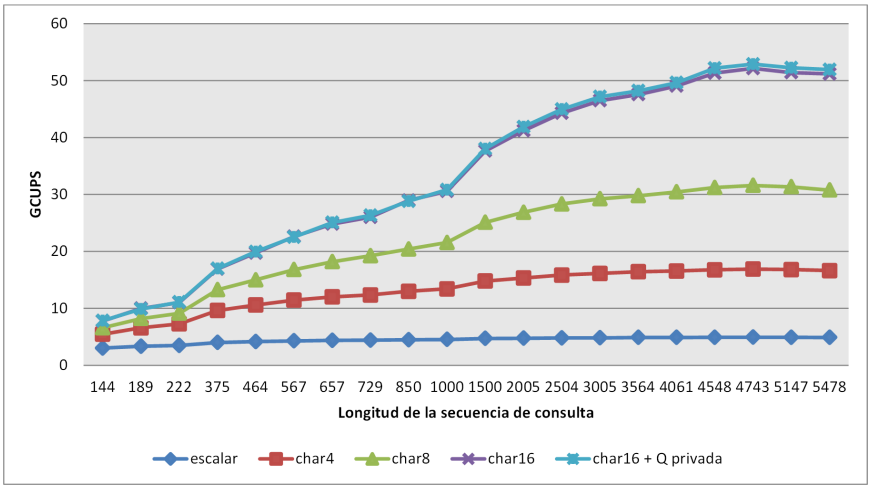

Figura 4.2: Rendimiento de las diferentes versiones de la implementación para una FPGA al variar el tamaño de la secuencia de consulta

4.2.3. Resultados de la implementación multi-FPGA

De los experimentos realizados en la sección anterior, se identificó a la versión *char16 + Q privada* como la de mayor rendimiento. Por ese motivo, se lo seleccionó para ser utilizado en los experimentos de la implementación multi-FPGA y de la implementación heterogénea híbrida. La Tabla 4.5 muestra el rendimiento de la implementación multi-FPGA para las dos bases de datos seleccionadas al variar el número de FPGAs. Como se puede observar, esta implementación se beneficia de cargas de trabajo más grandes. También es capaz de escalar su rendimiento exhibiendo un buen balance de carga al usar más de un acelerador. Debido al tamaño limitado de la base de datos Swiss-Prot, la implementación multi-FPGA alcanza una aceleración de 1.85× al usar dos aceleradores. Sin embargo, la aceleración aumenta a 1.96× con la base de datos más grande, Environmental NR.

Base de datos	FPGAs	
	1	2
Swiss-Prot	58.0	101.4
Environmental NR	58.4	114.0

Tabla 4.5: Rendimiento de la implementación multi-FPGA

4.2.4. Resultados de la implementación heterogénea híbrida

La Tabla 4.6 muestra el rendimiento de SWIMM (versión Xeon), de la implementación mono-FPGA y de la implementación CPU-FPGA híbrida al alinear las 20 secuencias de consulta seleccionadas contra Environmental NR en ambas arquitecturas de soporte. Se puede apreciar que la implementación híbrida logra rendimientos cercanos a las sumas de los rendimientos de las implementaciones individuales, lo que refleja la efectividad de la estrategia de distribución empleada. En particular, el rendimiento de OSWALD (*host*+FPGA) representa el 96.2 % de la suma de los rendimientos de SWIMM (*host*) y de OSWALD (FPGA) en el sistema basado en Xeon E5-2670. En el caso del sistema basado en E5-2695 v3, este porcentaje asciende a 97.1 %. La pequeña pérdida de rendimiento se debe al ocio en el que puede incurrir la implementación híbrida, ya sea durante la etapa de estimación de potencia de cómputo del *host* y la FPGA o también en el final debido a un desbalance por el margen de error de la estimación anterior. Cabe destacar que en el sistema basado en Xeon E5-2670 se empleó sólo un 1 % de los residuos de la base de datos para lograr la distribución de carga obtenida. Como el sistema basado en Xeon E5-2695 v3 es más potente, fue necesario aumentar a este porcentaje a 2 % para obtener resultados similares.

Implementación	Unidades de procesamiento	GCUPS	
SWIMM	Host	127.5	354.8
OSWALD	FPGA*	58.4	58.4
OSWALD	Host + FPGA	178.9	401.1
		Basado en Xeon E5-2670	Basado en Xeon E5-2695 v3
		Sistema	

* El *host* recomputa los alineamientos en donde ocurrió *overflow*

Tabla 4.6: Rendimiento de diferentes implementaciones SW en los dos sistemas heterogéneos

4.2.5. Comparación de rendimiento con otras implementaciones SW

Se comparó el rendimiento de la implementación heterogénea híbrida con otras implementaciones SW de referencia sobre las dos arquitecturas seleccionadas. Se seleccionó la herramienta SWIMM (descrita en el capítulo anterior) para la explotación de procesadores Xeon y de sistemas heterogéneos basados en coprocesadores Xeon Phi. Con respecto a la computación basada en GPUs, se eligió la herramienta CUDASW++ 3.0 (v3.1).

La Figura 4.3 ilustra el rendimiento alcanzado por las diferentes implementaciones SW para las distintas secuencias de consulta en el sistema heterogéneo basado en procesadores Intel Xeon E5-2670. Como se puede observar, la versión CPU de SWIMM presenta una curva casi plana debido al empleo de multihilado y el aprovechamiento de las extensiones SSE. La incorporación del coprocesador Xeon Phi le permite a SWIMM mejorar su rendimiento, con excepción en las secuencias más cortas ya que no es capaz de aprovechar todo el poder

computacional disponible. La ausencia de enteros vectoriales de bajo rango en el conjunto de instrucciones del Xeon Phi es la causa del poco rendimiento adicional que otorga. Los picos de rendimiento son 129.3 y 156.7 para la versión pura y la versión híbrida de SWIMM, respectivamente. Por otra parte, el rendimiento de OSWALD se encuentra siempre por encima de SWIMM (incluso para secuencias cortas) mientras que la diferencia se incrementa a medida que la secuencia de consulta se alarga. Gracias a una distribución balanceada de la carga de trabajo, OSWALD alcanza hasta 168.3 GCUPS. Por último, CUDASW++ 3.0 supera a todas las demás implementaciones logrando hasta 210 GCUPS, principalmente debido al gran poder computacional de la GPU NVIDIA K20c.

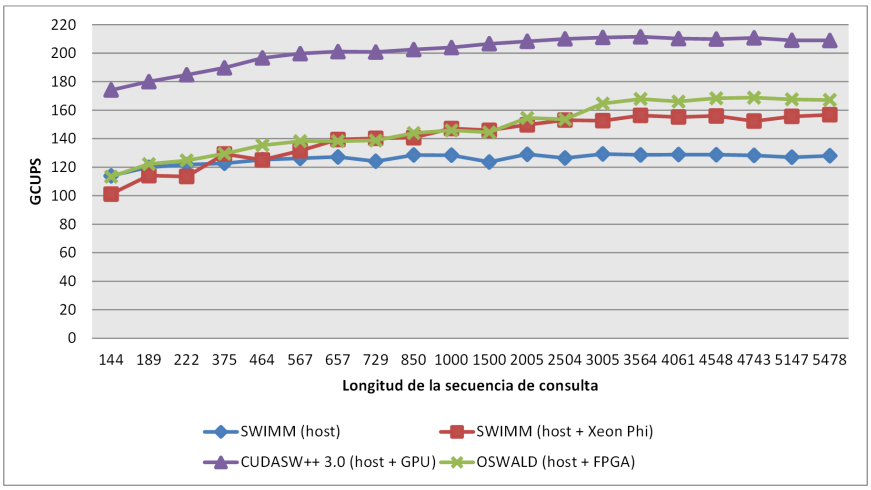

Figura 4.3: Rendimiento de las diferentes implementaciones SW al variar el tamaño de la secuencia de consulta en el sistema heterogéneo basado en procesadores Intel Xeon E5-2670

La Figura 4.4 muestra el rendimiento alcanzado por las diferentes implementaciones SW para las distintas secuencias de consulta en el sistema heterogéneo basado en procesadores Intel Xeon E5-2695 v3. A diferencia del otro sistema heterogéneo, este sistema posee más hilos hardware y dispone del conjunto de instrucciones AVX2 que permiten un mayor grado de paralelismo de datos. El comportamiento de SWIMM es similar al caso anterior. La versión CPU de SWIMM presenta una curva casi plana alcanzando hasta 360 GCUPS. La incorporación del coprocesador Xeon Phi decrementa el rendimiento para las secuencias de consulta más cortas y otorga pocos GCUPS adicionales para el resto. En contraste con los resultados obtenidos en el sistema anterior, CUDASW++ 3.0 presenta el peor rendimiento. Esto se debe a que CUDASW++ 3.0 sólo es capaz de explotar las extensiones SSE en el *host* y no puede aprovechar la disponibilidad de las AVX2. Finalmente, OSWALD logra las mejores tasas de rendimiento con valores cercanos a los 400 GCUPS. La explotación de las instrucciones AVX2 y una carga balanceada de trabajo entre *host* y acelerador son aspectos claves en el rendimiento logrado por OSWALD.

4.3. Resumen y conclusiones

En este capítulo, se introdujo la herramienta OSWALD para búsquedas biológicas en sistemas heterogéneos basados en aceleradores FPGA, la cual trabaja con bases de datos biológicas reales y permite configurar los parámetros de ejecución. Mediante esta herramienta, se estudió y evaluó la viabilidad de esta clase de sistemas heterogéneos para el

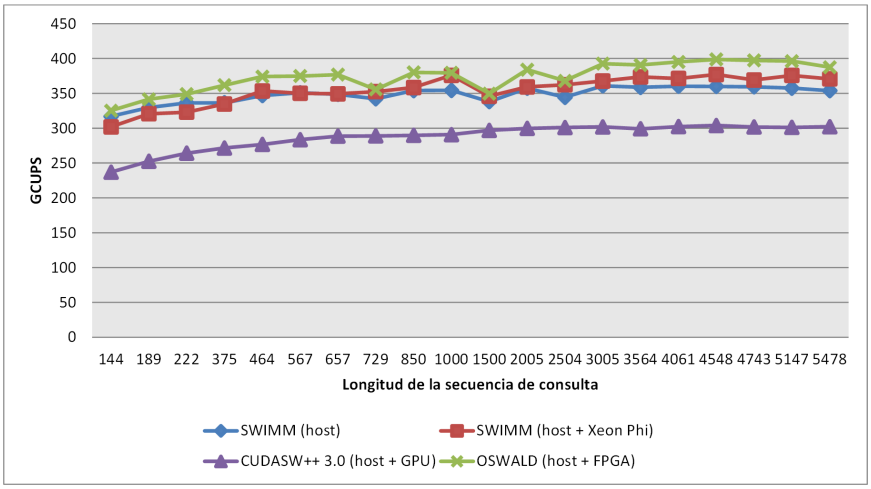

Figura 4.4: Rendimiento de las diferentes implementaciones SW al variar el tamaño de la secuencia de consulta en el sistema heterogéneo basado en procesadores Intel Xeon E5-2695 v3

alineamiento de secuencias de proteínas mediante el algoritmo SW. A diferencia de las implementaciones disponibles en la literatura, se decidió explorar los beneficios de utilizar una tecnología innovadora, como lo es OpenCL en el ámbito de las FPGAs, en lugar de HDLs tradicionales como VHDL o Verilog.

Como primer paso, se desarrolló una implementación para una única FPGA siguiendo el modelo de programación paralela de tarea especificado en OpenCL 1.0. En cuanto a la paralelización de los alineamientos se optó por el esquema inter-tarea. Para el desarrollo del algoritmo, se estudiaron y aplicaron diferentes técnicas de programación y optimización; entre ellas, la selección de puntajes de sustitución mediante la estrategia SP, la explotación de enteros de bajo rango, el desenrollado de bucles, el procesamiento por bloques, el empleo de canales, la alineación de accesos a memoria y la explotación de la jerarquía de memoria. Luego, se evaluó el rendimiento y el consumo de recursos de las diferentes versiones desarrolladas de forma de encontrar la configuración más beneficiosa. A continuación, se extendió la implementación mono-FPGA para que sea capaz de soportar ejecución en múltiples aceleradores al mismo tiempo. Finalmente, utilizando el código Xeon de SWIMM se implementó una versión híbrida capaz de computar alineamientos en la CPU y en las FPGAs de forma simultánea.

Empleando las secuencias de consulta y las bases de datos de proteínas utilizadas en el capítulo anterior, se probó la implementación multi-FPGA y la implementación híbrida sobre dos arquitecturas heterogéneas diferentes. También se probaron otras implementaciones de referencia con las cuales comparar las propias: SWIMM para sistemas heterogéneos basados en Xeon Phi y CUDASW++ 3.0 para los basados en GPUs compatibles con CUDA. Lamentablemente no fue posible realizar una comparación con otras implementaciones FPGA debido a la falta de disponibilidad del código fuente. Si bien un análisis teórico es posible, las significativas diferencias entre las distintas implementaciones impiden una comparación justa. A continuación se detallan las principales conclusiones del análisis experimental:

- De la exploración de los diferentes kernels SW se puede extraer:
  - Al utilizar el modelo de programación paralela de tarea de OpenCL no se requiere

sincronización adicional.

- Al computar las matrices de alineamiento empleando técnicas de procesamientos por bloques no sólo se mejora la localidad de datos sino que también se reduce los requerimientos de memoria del *kernel*, lo que favorece el uso de memoria privada de baja latencia.
- Al desenrollar el bucle más interno se aumenta el número de operaciones por ciclo de reloj y se logra acceso coalescente a la memoria.
- Al emplear los canales de comunicación Altera se resuelve en forma eficiente las dependencias de datos entre los bloques de cada matriz de alineamiento.
- Al explotar un mayor grado de paralelismo de datos a través de la vectorización de operaciones se logra incrementar el rendimiento en forma notable a expensas de un aumento moderado en el consumo de recursos.
- Al explotar tipos de dato enteros de bajo rango no sólo se reduce el consumo de recursos sino que también se mejora el rendimiento. De esta forma se aprovecha el hecho de que los puntajes de similitud no requieren de un amplio rango de representación. En particular, se mostró que resulta conveniente computar los alineamientos usando enteros de 8 bits en la FPGA y recomputar en el *host* cuando sea necesario.
- Al calcular los SPs en el *host* se evita dedicar recursos de los aceleradores para tal función, de manera de poder destinarlos a aumentar el número de celdas que se pueden computar a la vez.
- Con respecto a la explotación de la jerarquía de memoria, el uso de memoria privada reporta grandes beneficios, el uso de memoria global se debe evitar siempre que sea posible y el uso de memoria constante se debe estudiar cuidadosamente.
- Lograr una carga balanceada entre los diferentes dispositivos de procesamientos es un factor clave para obtener alto rendimiento. En la implementación multi-FPGA, ésto se logra en la etapa de preprocesamiento al dividir la base de datos en número de *chunks* que sea múltiplo del número de FPGAs y procurando que los *chunks* contengan aproximadamente la misma cantidad de residuos. Por su parte, la implementación híbrida requiere de un esquema más complejo por las diferentes capacidades computacionales que pueden tener el *host* y los aceleradores. Al estimar la potencia de cómputo de cada dispositivo de procesamiento, no sólo se logra una carga de trabajo balanceada con un *overhead* mínimo sino también una implementación general para cualquier sistema heterogéneo de esta clase.

- Con respecto a los GCUPS, la implementación mono-FPGA logró 58.4 GCUPS mientras que la implementación multi-FPGA con dos aceleradores obtuvo 114 GCUPS, lo que representa una escalabilidad casi lineal. Por su parte, la implementación híbrida alcanzó 178.9 y 401.1 GCUPS en los sistemas basados basados en Xeon E5-2670 y en Xeon E5-2695 v3, respectivamente.

- En cuanto a la comparación con otras implementaciones SW, OSWALD y SWIMM mostraron comportamientos similares en los dos sistemas heterogéneos utilizados. El rendimiento de OSWALD estuvo siempre por encima del de SWIMM (incluso para secuencias cortas), logrando mayores diferencias a medida que la carga de trabajo

aumentaba. Por su parte, CUDASW++ 3.0 superó a todas las demás implementaciones en el sistema basado en Xeon E5-2670, principalmente por el gran poder computacional de la GPU NVIDIA K20c. En contraste, presentó el peor rendimiento en el sistema basado en Xeon E5-2695 v3 debido a su incapacidad de aprovechar las extensiones AVX2.

El costo de programación y la falta de portabilidad en los códigos FPGA han limitado tradicionalmente su aplicación a las búsquedas SW. OSWALD es una implementación portable, completamente funcional y general para acelerar búsquedas de similitud en sistemas heterogéneos basados en FPGA que demostró ser competitivo con otras implementaciones SW de referencia. Como en la actualidad, la comunidad HPC no sólo está interesada en mejorar el rendimiento sino también en reducir el consumo energético, en el capítulo siguiente se evalúa la eficiencia energética de diferentes sistemas heterogéneos en el contexto de SW.

Capítulo 5

Eficiencia energética de SW en sistemas heterogéneos

En la actualidad, evaluar el consumo energético de una arquitectura además de su rendimiento se ha tornado una tarea necesaria en HPC. Este capítulo estudia y evalúa la viabilidad de diferentes sistemas heterogéneos para el alineamiento de secuencias de proteínas desde la perspectiva de la eficiencia energética. La Sección 5.1 detalla la metodología de medición empleada en cada uno de los dispositivos junto a los diferentes experimentos realizados. Luego se analizan los rendimientos energéticos de las implementaciones SWIMM (Sección 5.2) y OSWALD (Sección 5.3). A continuación, se realiza un análisis comparativo del rendimiento, del consumo de potencia y de la eficiencia energética lograda por las diferentes implementaciones utilizadas (Sección 5.4) seguido de una comparación con trabajos relacionados (Sección 5.5). Finalmente, la Sección 5.6 presenta un resumen del contenido del capítulo.

5.1. Entorno y diseño experimental

Para evaluar el rendimiento energético de las diferentes implementaciones analizadas en los dos capítulos anteriores, no se realizaron pruebas especiales sino que se monitorizó el consumo de potencia durante la ejecución de las mismas. En particular, se tuvieron en cuenta todas las ejecuciones que procesaron la base de datos Environmental NR. Las ejecuciones correspondientes a Swiss-Prot no fueron consideradas ya que su tamaño resultaba limitado para el análisis cuando se empleaba todos los recursos computacionales de los diferentes sistemas utilizados.

Con respecto al consumo de potencia, se describe la metodología de medición empleada en el *host* y en los diferentes aceleradores:

- En los procesadores Xeon, se empleó la herramienta Performance Counter Monitor (versión 2.7) [1], la cual fue desarrollada por Intel y permite realizar mediciones de potencia en sus procesadores. Se seleccionó esta herramienta dado que desde su versión 2.0 tiene soporte para los procesadores Xeon E5 utilizados en esta tesis. En particular, los procesadores Intel ya contaban con capacidad de monitorización mediante contadores de hardware, pero no era evidente la forma de determinar el consumo de

[1] Intel PCM se encuentra disponible en http://www.intel.com/software/pcm

potencia utilizándolos. La interfaz de Performance Counter Monitor permite a cualquier programador realizar análisis de consumo de recursos de los procesadores en forma fácil.

- Los coprocesadores Xeon Phi proveen información sobre el consumo de potencia a través de la utilidad Intel System Management Controller (SMC)[117]. Esta utilidad accede a un microcontrolador ubicado en la placa, el cual monitoriza la entrada de corriente continua y distintos sensores térmicos. En este contexto, un analizador de potencia basado en software y desarrollado por Intel facilita la medición de consumo de potencia del coprocesador mediante la utilidad *micsmc*. Aun más, el trabajo [118] concluye que las mediciones realizadas por medio de SMC son completamente confiables, observando menos de 1 % de error en comparación a medir directamente el consumo del Xeon Phi a través del canal correspondiente del PCI-e.

- En forma similar a SMC para los Xeon Phi de Intel, NVIDIA ha presentado la utilidad NVIDIA System Management Interface [2] basada en la librería NVIDIA Management Library y pensada para ayudar en la administración y monitorización de sus placas gráficas. Las últimas GPUs de NVIDIA cuentan con sensores integrados que permiten consultar el consumo de potencia durante la ejecución mediante la utilidad *nvidia-smi*.

- La FPGA es monitorizada mediante un extensor PCI-Express Furaxa (modelo PCIeEXT16HOT), el cual reporta la corriente suministrada en las dos líneas de 12V y 3.3V. Los valores reportados por el extensor son adquiridos por un dispositivo adquisidor de datos USB, el cual está conectado a una computadora externa. Para administrar estos dispositivos se usó el software *pmlib* [119]. Este entorno *ad-hoc* permite monitorizar el consumo de potencia con una frecuencia de muestreo suficiente para los experimentos realizados.

Las mediciones se realizaron en forma concurrente en el *host* mediante un *script* Python. Se tomaron 20 muestras por segundo en todos los casos.

5.2. Resultados energéticos de SWIMM

Esta sección se enfoca en el consumo de potencia de la implementación híbrida SWIMM en el sistema heterogéneo basado en procesadores Xeon E5-2670 empleando todos los hilos hardware disponibles y el coprocesador Xeon Phi. En la Sección 3.2.4.1, se observó que una carga de trabajo equilibrada entre el *host* y el acelerador es una de las claves para obtener buenos desempeños en una arquitectura heterogénea. De acuerdo a esta premisa, en la Figura 5.1 se detalla el consumo de potencia del sistema heterogéneo en cuestión al emplear la implementación híbrida de SWIMM (con distribución dinámica) para alinear la secuencia de consulta más larga contra la base de datos Environmental NR. En esta implementación, los *chunks* de la base de datos se distribuyen entre el *host* y el acelerador siguiendo un esquema bajo demanda. Se puede notar en la figura que el tamaño del *chunk* incide sobre el balance de carga (tamaños de 64MB y 128MB). En este caso, una granularidad más fina (*chunks* más pequeños) en la distribución del trabajo conlleva a un mejor balance de carga, lo que a su vez implica un menor tiempo de ejecución. Este hecho se hace evidente en la

[2] NVIDIA System Management Interface: https://developer.nvidia.com/nvidia-system-management-interface

ejecución con *chunk* de 128B donde el *host* espera a que el coprocesador termine con su última tarea.

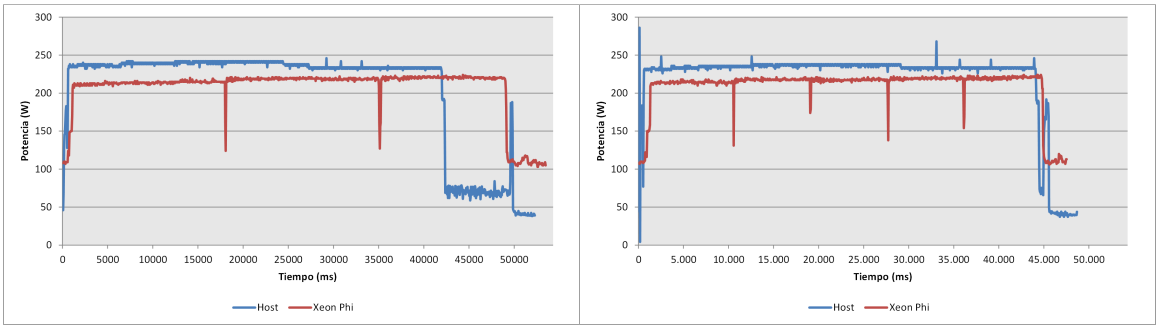

Figura 5.1: Perfil del consumo de potencia de la implementación híbrida de SWIMM (con distribución dinámica) para *chunks* de tamaño 128MB (izquierda) y 64 MB (derecha).

Además, en la Figura 5.1 se pueden apreciar las diferentes fases de ejecución en el Xeon Phi. Las caídas en el consumo del coprocesador están asociadas con las transferencias de entrada y de salida involucradas en el procesamiento de un *chunk*. Este comportamiento no se puede percibir en el *host* porque no es necesario transferir datos para procesar un *chunk* y es por eso que se observa un consumo constante a lo largo del tiempo. Finalmente, el último pico de consumo en el *host* se debe a la etapa final de ordenación de los puntajes una vez que todos los alineamientos fueron procesados.

En la Figura 5.2 se detalla la energía (medida en Joules) requerida por cada dispositivo del sistema heterogéneo utilizando la implementación híbrida de SWIMM (con distribución estática) al variar la carga de trabajo entre ellos. Las barras azules se corresponden con el consumo de potencia del *host* mientras que las rojas hacen referencia al del Xeon Phi. Como se espera, el perfil del consumo varía sustancialmente con respecto al desempeño observado en la Figura 3.10. Las bajas tasas de rendimiento observadas en el Xeon Phi (en términos de GCUPS) penalizan el consumo de forma severa. De hecho, la configuración que computa el 100 % de las secuencias en el *host* (deshabilitando el coprocesador) alcanza un ratio de 0.52 GCUPS/Watts mientras que la configuración que obtiene el mejor rendimiento en el sistema heterogéneo (75 % en el *host* y 25 % en el coprocesador) sólo logra 0.368 GCUPS/Watts.

5.3. Resultados energéticos de OSWALD

Esta sección se centra en el consumo de potencia de la implementación híbrida de OSWALD en el sistema heterogéneo basado en procesadores Xeon E5-2670 empleando todos los hilos hardware disponibles y una FPGA. En la Figura 5.1 se detalla el consumo de potencia del sistema heterogéneo en cuestión al alinear la secuencia de consulta más larga contra la base de datos Environmental NR. A partir de la gráfica, se pueden apreciar las diferentes etapas que atraviesa esta implementación, las cuales fueron denotadas del (1) al (4). La etapa (1) corresponde a la estimación de la potencia de cómputo de cada uno de los dispositivos mientras que la etapa (2) referencia la adaptación y división de la base de datos para su posterior procesamiento paralelo. A continuación, la etapa (3) representa el cómputo de los alineamientos por parte del *host* y de la FPGA y, por último, la etapa (4) se debe a la ordenación de los puntajes de similitud.

Adicionalmente, resulta fácil notar el bajo consumo de potencia por parte de la FPGA,

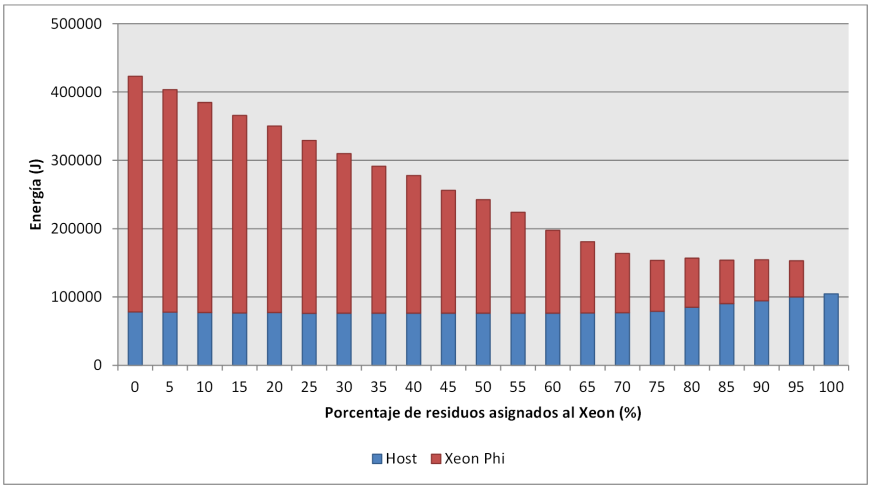

Figura 5.2: Consumo de energía de la implementación híbrida de SWIMM (con distribución estática) al variar la carga de trabajo entre los dispositivos

la cual presenta una curva casi plana durante toda la ejecución. Los leves descensos de consumo en la etapa (3) se deben al ocio en el que incurre la FPGA por la construcción de los SPs en el *host*. Con respecto a la CPU, su porción de la base de datos se compone de un único *chunk*, por lo que todos los hilos están activos mientras hay alineamientos para procesar. Es por eso que su consumo es constante y no presenta descensos durante la etapa (3). Por último, los períodos de menor consumo en la CPU coinciden con los períodos de ocio asociados a las transiciones de una etapa a otra.

5.4. Comparación de rendimiento, consumo de potencia y eficiencia energética

En la Tabla 5.1 se resumen los promedios de rendimiento y de consumo de potencia obtenidos en las dos arquitecturas heterogéneas utilizadas. Los valores corresponden al alineamiento de las 20 secuencias de consulta seleccionadas contra la base de datos Environmental NR. Como se puede observar, la computación basada en FPGA presenta uno de los valores de GCUPS más bajos, por lo que no representa la mejor opción desde el punto de vista del rendimiento. Sin embargo, sí lo puede ser desde la perspectiva del consumo de potencia; su bajo consumo (TDP menor a 25W) puede ser de utilidad en entornos con restricciones energéticas o cuando el consumo de potencia deba mantenerse debajo de cierto umbral. En ese sentido, también se observa que la utilización de un único hilo en el *host* es una manera conveniente de reducir el consumo de potencia (16 %) a expensas de una reducción más pequeña en el rendimiento (8 %)[3]. En sentido contrario, el uso de una arquitectura heterogénea basada en coprocesadores Xeon Phi no es una buena opción desde la óptica del consumo de potencia. Incluso tampoco lo es desde el punto de vista del rendimiento si sólo se computa en el Xeon Phi, ya que este modo presenta el valor más bajo de GCUPS. La incorporación del Xeon Phi al cómputo de los alineamientos provee poca mejora de rendimiento al *host*

[3]El número de hilos hardware utilizados en el *host* afecta principalmente a la etapa SW (construcción de los SPs y cómputo de los alineamientos en los que ocurrió *overflow*), aunque también incide en la etapa de preprocesamiento y en la de ordenación.

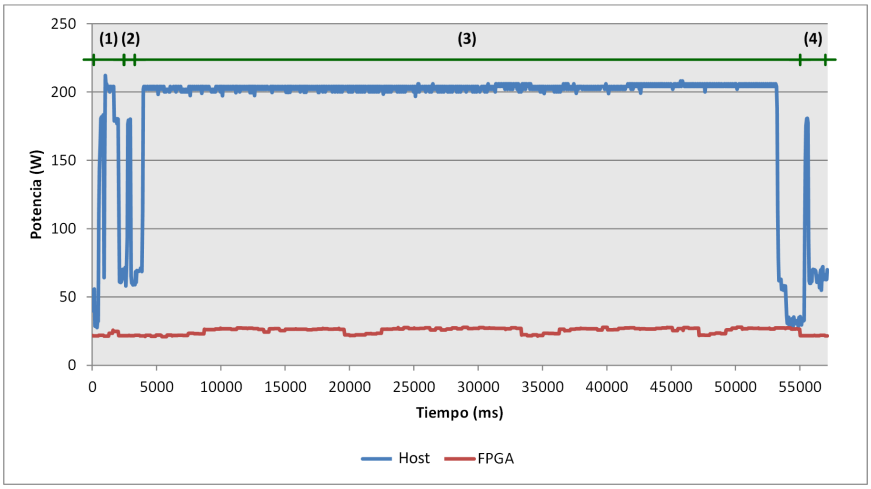

Figura 5.3: Perfil del consumo de potencia de OSWALD (versión híbrida) en el sistema basado en Xeon E5-2670.

y decrementa su cociente GCUPS/Watt. El pobre rendimiento alcanzado por el Xeon Phi se debe a la ausencia de capacidades vectoriales para explotar enteros de bajo rango. De hecho, este aspecto es una de las claves para mejorar la relación GCUPS/Watts, como se evidencia en el Xeon E5-2695 v3 (AVX2) comparado al Xeon E5-2670 (SSE). También se aprecia que el uso de Hyper-Threading (dos hilos hardware en lugar de uno) mejora la tasa GCUPS/Watt en ambas arquitecturas. Esto se debe a que el porcentaje de mejora de rendimiento por el uso de esta tecnología es mayor al porcentaje de aumento en el consumo de potencia en los dos casos. Por otra parte, la combinación de CPUs con GPUs puede ser una buena alternativa en sistemas que exploten las instrucciones SSE, pero que no dispongan de las extensiones AVX2. En el sistema basado en Xeon E5-2670, CUDASW++ 3.0 mejora los GCUPS y el cociente GCUPS/Watt de SWIMM (versión Xeon). Sin embargo, no es capaz de repetir sus prestaciones en el sistema basado en Xeon E5-2695 v3 debido a que CUDASW++ 3.0 sólo explota las instrucciones SSE. Por último, la computación híbrida basada en CPU-FPGA se ubica como la opción más eficiente energéticamente teniendo en cuenta que obtiene los cocientes GCUPS/Watt más altos en ambos sistemas. Aun más, la inclusión de una FPGA adicional mejora los cocientes en cuestión en las dos arquitecturas. Para concluir, resulta importante mencionar que la FPGA es el único acelerador que mejora las tasas GCUPS/Watt del *host* en ambos sistemas, destacando una mejora de aproximadamente 25 % en el sistema *menos* potente debido principalmente a una distribución más homogénea de la carga de trabajo entre el *host* y las FPGAs.

5.5. Comparación con trabajos relacionados

Existe amplia disponibilidad en la literatura de trabajos que investigan el rendimiento y la eficiencia energética de diferentes dispositivos de procesamiento como pueden ser CPUs, GPUs, FPGAs, entre otros. Sin embargo, sólo dos lo hacen en el contexto del algoritmo SW. En el año 2011, Zou *et al.* [103] presentaron implementaciones para diferentes dispositivos (CPU, GPU y FPGA) y evaluaron su rendimiento y consumo de potencia. En forma similar a la Sección 5.4, ellos encontraron que las FPGAs son más eficientes energéticamente (en término de GCUPS/Watt) que las CPUs y las GPUs, y a su vez, que las GPUs superan en

Sistema	Unidades de cómputo	Núcleos	GCUPS	Potencia (Watt)	GCUPS/Watt
Basado en Xeon E5-2695 v3	Host	28	309.3	228.2	1.355
	Host	56	354.8	240	1.478
	FPGA*	1	53.5	69	0.775
	FPGA*	28	58.4	83.1	0.702
	Host + Xeon Phi	56+228	380	450.5	0.843
	Host + GPU	56+2496	298.8	328.2	0.910
	Host + FPGA	56	401.1	265.6	1.510
	Host + 2×FPGA	56	441.6	291.2	1.516
Basado en Xeon E5-2670	Host	16	110.1	209.7	0.525
	Host	32	127.5	230	0.554
	Xeon Phi	228	41.9	268.2	0.156
	Host + Xeon Phi	32+228	165.5	438.5	0.377
	Host + GPU	32+2496	206.2	303.2	0.680
	Host + FPGA	32	178.9	253.1	0.707
	Host + 2×FPGA	32	225.1	271	0.830

* El *host* recomputa los alineamientos en donde ocurrió *overflow*

Tabla 5.1: Resumen de promedios de rendimiento y de consumo de potencia en las dos arquitecturas heterogéneas utilizadas

este aspecto a las CPUs. Sin embargo, los coeficientes de mejora son muy diferentes a los de la Sección 5.4; en particular, las FPGAs logran una mejora de 50× comparado a las CPUs y de 26× comparado a las GPUs, mientras que éstas últimas alcanzan una mejora de 2× respecto a las CPUs. Esta diferencia significativa se debe al menos a tres razones. En primer lugar, los valores de consumo de potencia empleados para calcular las tasas GCUPS/Watt no incluyen el consumo del *host* en los casos de las FPGAs y las GPUs, lo que claramente beneficia a estos dos últimos dispositivos. En segundo lugar, las implementaciones propuestas para CPUs y GPUs son sub-óptimas, lo que aumenta el factor de mejora de las FPGAs. En el caso de las CPUs, los autores reportan pocos detalles sobre la implementación: explotan multihilado a través de OpenMP y emplean el esquema inter-tarea mediante las extensiones SSE. A diferencia de SWIPE y SWIMM, los alineamientos se computan con datos vectoriales de 16 bits, lo que reduce el número de alineamientos simultáneos a 8. En cuanto a los GCUPS, reportan una tasa por hilo igual a 0.55 mientras que Farrar logró 2.9 cuatro años antes [7]. Por su parte, la propuesta para GPUs brinda más detalles de implementación y de las técnicas de optimización utilizadas. Sin embargo, también presenta rendimientos inferiores a los de propuestas anteriores. Mientras que Zou *et al.* alcanzaron 13.71 GCUPS sobre una NVIDIA GeForce GTX 280, Youngchao Liu *et al.* reportaron 16.9 GCUPS un año antes empleando la misma GPU [102]. También un año antes, Kentie reportó 21.36 GCUPS sobre una placa gráfica de características muy parecidas (NVIDIA GeForce GTX 275) [42]. En tercer y último lugar, los experimentos para medir el rendimiento y el consumo fueron realizados con una única secuencia de tamaño fijo y una base de datos sintética conformada por secuencias del mismo tamaño con residuos aleatorios. Este último aspecto favorece a la implementación FPGA por su modo de procesamiento.

En el año 2012, Benkrid *et al.* [114] evaluaron el rendimiento y el consumo de potencia de implementaciones para diferentes dispositivos (CPU, GPU, Cell/BE y FPGA). De acuerdo a los GCUPS/Watt obtenidos, los autores encuentran a la FPGA como el dispositivo más eficiente energéticamente seguido por el Cell/BE, la GPU y la CPU. Estos resultados coinciden con la Sección 5.4 pero al igual que el trabajo de Zou *et al.*, los coeficientes de mejora varían significativamente; en particular, las FPGAs obtienen una mejora de 500× en comparación a las CPUs y de 23× en comparación a las GPUs, mientras que éstas últimas

superan por 25× a las CPUs. Estas diferencias se deben a motivos similares al caso anterior. En primer lugar, el consumo de potencia del *host* no es considerado al momento de calcular los GCUPS/Watt de la versiones con aceleradores. En segundo lugar, los autores exponen que para hacer la comparación más justa, emplean dispositivos fabricados con la misma tecnología (90nm), lo que favorece a las versiones con aceleradores teniendo en cuenta que las CPUs son más avanzadas en este aspecto. En tercer lugar, como los autores también evalúan el costo de programación y productividad de cada dispositivo, emplean implementaciones propias desarrolladas por un grupo de alumnos de doctorado que resultan sub-óptimas. De hecho, los autores son conscientes de este aspecto, ya que reconocen en su trabajo la existencia de implementaciones más *rápidas* y, aun más, indican que los coeficientes de mejora se verían disminuidos ampliamente en caso de usarlas.

5.6. Resumen y conclusiones

En este capítulo, se estudió y evaluó la eficiencia energética de diferentes sistemas heterogéneos para el alineamiento de secuencias de proteínas mediante el algoritmo SW. Para ello, no se realizaron pruebas especiales sino que se monitorizó el consumo de potencia por parte de los diferentes sistemas en los experimentos realizados en los Capítulos 3 y 4.

En primer lugar, se analizó el consumo de potencia de las implementaciones híbridas de SWIMM y de OSWALD en forma individual. Posteriormente, se analizó comparativamente el rendimiento y el consumo de potencia obtenidos en las diferentes arquitecturas heterogéneas utilizadas. A continuación se detallan las principales conclusiones del análisis experimental:

- Algunas de las técnicas que, además de incrementar los GCUPS, mejoraron la eficiencia energética son:
  - Explotar mayor paralelismo de datos enteros de bajo rango, como se evidenció en el sistema basado en Xeon E5-2695 v3 (AVX2) comparado al basado en Xeon E5-2670 (SSE).
  - Usar Hyper-Threading. El uso de esta tecnología provocó un incremento mayor en el rendimiento que el correspondiente al consumo de potencia en ambas arquitecturas.
  - Lograr una carga de trabajo equilibrada entre *host* y aceleradores. De esta manera, no sólo se mejoraron las prestaciones sino que también se redujo el consumo energético.
- En cuanto a la computación basada únicamente en aceleradores (sin ejecución simultánea del *host*):
  - La FPGA puede no ser la mejor opción desde el punto de vista del rendimiento. Sin embargo, sí lo puede ser desde la perspectiva del consumo de potencia; su bajo consumo puede ser de utilidad cuando éste aspecto sea la mayor prioridad.
  - El Xeon Phi no representa una buena elección tanto desde la óptica del rendimiento como del consumo de potencia.
- A través de SWIMM, las CPUs presentaron un buen balance entre rendimiento y consumo. Sus prestaciones mejoraron significativamente al explotar las extensiones AVX2. Respecto a la combinación con aceleradores:

- El uso de arquitecturas heterogéneas basadas en coprocesadores Xeon Phi no fue una buena opción en términos del consumo de potencia. La integración del Xeon Phi al cómputo de los alineamientos proveyó poca mejora de rendimiento al *host* y decrementó su cociente GCUPS/Watt. Se espera que la incorporación futura de las extensiones AVX-512 cambie esta situación.
- En el sistema basado en Xeon E5-2670, CUDASW++ 3.0 mejoró los GCUPS y el cociente GCUPS/Watt de SWIMM (versión Xeon). Sin embargo, no fue capaz de repetir sus prestaciones en el sistema basado en Xeon E5-2695 v3 debido a que CUDASW++ 3.0 sólo explota las instrucciones SSE. Por ese motivo, la combinación de CPUs con GPUs puede ser una buena alternativa en sistemas que puedan explotar las instrucciones SSE, pero que no dispongan de las extensiones AVX2. El desarrollo de una herramienta que dé soporte a las extensiones AVX2 permitiría aumentar la eficiencia energética de esta clase de sistemas.
- La computación híbrida basada en CPU-FPGA se ubicó como la opción más eficiente energéticamente teniendo en cuenta que OSWALD obtuvo los cocientes GCUPS/Watt más altos en ambos sistemas. Aun más, la FPGA fue el único acelerador que mejoró las tasas GCUPS/Watt del *host* en ambos sistemas.

Aunque existen otras evaluaciones de eficiencia energética en el contexto de SW, éstas poseen uno o más limitaciones que reducen su utilidad en el mundo real. En esta evaluación, se han empleado secuencias de consulta y bases de datos representativas de la comunidad bioinformática, potentes arquitecturas orientadas a HPC y eficientes implementaciones que han demostrado ser las más rápidas de su clase. En base a estas condiciones y teniendo en cuenta el rol de SW en la bioinformática, se espera que esta evaluación le resulte útil a cualquier centro de investigación y desarrollo de este área al momento de seleccionar un sistema HPC para sus aplicaciones de acuerdo con las prioridades que posea.

Capítulo 6

Conclusiones y líneas de trabajo futuro

En este capítulo se exponen las conclusiones de esta tesis (Sección 6.1) así como las posibles líneas de trabajo futuro (Sección 6.2).

6.1. Conclusiones

El problema del consumo energético se presenta como uno de los mayores obstáculos para el diseño de sistemas que sean capaces de alcanzar la escala de los Exaflops. Por lo tanto, la comunidad científica está en la búsqueda de diferentes maneras de mejorar la eficiencia energética de los sistemas HPC. Una tendencia reciente para incrementar el poder computacional y al mismo tiempo limitar el consumo de potencia de estos sistemas consiste en incorporarles aceleradores y coprocesadores, como pueden ser las GPUs de AMD y NVIDIA o los coprocesadores Xeon Phi de Intel. Por otra parte, las FPGAs aparecen como una opción promisoria para HPC debido a su capacidad de cómputo creciente, su bajo consumo energético y al desarrollo de nuevas herramientas que facilitan su programación. Estos sistemas híbridos que emplean diferentes recursos de procesamiento se denominan sistemas heterogéneos y son capaces de obtener mejores cocientes FLOPS/Watt.

Entre las áreas que se ven afectadas por los problemas actuales de los sistemas HPC se encuentra la bioinformática, debido al crecimiento exponencial que ha experimentado la información biológica en los últimos años y a que cuenta con un número creciente de aplicaciones que requieren de cómputo de altas prestaciones para alcanzar tiempos de respuesta aceptables. Una de esas aplicaciones es el alineamiento de secuencias, la cual representa una operación fundamental en cualquier investigación biológica. El alineamiento consiste en comparar dos o más secuencias biológicas y su propósito es detectar qué regiones comparten una historia evolutiva común. El algoritmo SW es un método popular para el alineamiento local de secuencias que ha sido utilizado como base para otros algoritmos posteriores y como patrón con el cual comparar otras técnicas de alineamiento. Sin embargo, debido a que su complejidad computacional es cuadrática, en la práctica se emplean diversas heurísticas que permiten reducir el tiempo de ejecución a costo de una pérdida en la sensibilidad de los resultados.

De forma de procesar el creciente volumen de información biológica con tiempos de respuesta aceptables, resulta necesario desarrollar nuevas herramientas computacionales que sean capaces de acelerar primitivas claves y algoritmos elementales eficientemente en térmi-

nos de rendimiento y consumo energético. Por ese motivo, esta tesis se ha planteado como objetivo general *evaluar el rendimiento y la eficiencia energética de sistemas para cómputo de altas prestaciones al acelerar el alineamiento de secuencias biológicas mediante el método de Smith-Waterman.* A continuación, se detallan los objetivos específicos y se describe cómo se cumplieron los mismos:

- *Describir y comparar las propuestas científicas para el alineamiento de secuencias biológicas utilizando sistemas para cómputo de altas prestaciones con el objeto de caracterizarlas.*

En el Capítulo 2 se estudiaron las posibles formas de paralelizar el algoritmo SW y se describieron las implementaciones existentes sobre diferentes plataformas de procesamiento: CPU, GPU, FPGA y Xeon Phi. El análisis para cada dispositivo incluyó la evolución temporal de sus implementaciones así como también una descripción detallada de los aportes, las limitaciones, los resultados y la experimentación realizada en cada uno de los trabajos. Este análisis fue posible gracias al estudio de las diferentes arquitecturas y modelos de programación además del de los distintos algoritmos para alineamiento de secuencias biológicas llevado al inicio de dicho capítulo.

- *Identificar los factores que inciden en el rendimiento y la eficiencia energética de los sistemas para cómputo de altas prestaciones al acelerar el alineamiento de secuencias.*

Algunos de los factores que inciden en el rendimiento de las implementaciones SW pudieron ser identificados durante el análisis realizado en el Capítulo 2, lo cual resultó de utilidad al momento de desarrollar las implementaciones presentadas en los dos capítulos siguientes. El resto de ellos fueron reconocidos a partir del trabajo experimental realizado en los Capítulos 3 y 4 y se resumen al final de cada uno. Los factores que inciden en la eficiencia energética pudieron ser identificados a través de la monitorización del consumo de potencia de los experimentos llevados a cabo en los Capítulos 3 y 4 y el posterior análisis realizado en el Capítulo 5. Al igual que el caso anterior, el final del Capítulo 5 resume estos factores.

- *Diseñar y desarrollar soluciones algorítmicas para aquellos sistemas de cómputo de altas prestaciones que no tengan implementaciones disponibles o, en caso contrario, que superen a las existentes en cuanto a aceleración.*

Como se mencionó en la Sección 1.3, las GPUs son el acelerador dominante en la comunidad de HPC al día de hoy y existe vasta investigación científica sobre el uso de esta clase de aceleradores para el alineamiento de secuencias. Por ese motivo, se decidió priorizar el desarrollo de nuevas soluciones algorítmicas para sistemas heterogéneos basados en Xeon Phi y basados en FPGA. Al inicio de esta tesis no existían implementaciones disponibles para el alineamiento de secuencias basadas en coprocesadores Xeon Phi. En sentido opuesto, sí existían antecedentes en la aplicación de FPGAs para el procesamiento de secuencias biológicas, aunque estas implementaciones fueron desarrolladas con HDLs tradicionales y en su mayoría poseen una o más limitaciones que restringen su uso en el mundo real.

El Capítulo 3 presenta soluciones algorítmicas para sistemas heterogéneos basados en Xeon Phi. Como punto de partida, se desarrollaron y optimizaron implementaciones para las CPUs y los Xeon Phi en forma individual antes de combinarlos en una implementación híbrida. Las implementaciones desarrolladas fueron compiladas en una única herramienta a la cual se la llamó SWIMM. A partir del análisis realizado en el Capítulo 2, se identificó a SWIPE como la herramienta más rápida para búsquedas de similitud en CPU. Mediante

los experimentos realizados en el Capítulo 3, se mostró que la versión SSE de SWIMM es equiparable con SWIPE mientras que la versión AVX2 logró superarlo ampliamente alcanzado diferencias de hasta 1.4×. Cabe destacar que, hasta donde llega el conocimiento del tesista, esta es la primera implementación SW para el conjunto de instrucciones AVX2. En cuanto a los Xeon Phi, al inicio de esta investigación no existían implementaciones disponibles para este coprocesador. Sin embargo, en el año 2014 aparecieron SWAPHI y XSW 2.0. Mientras que SWAPHI sólo explota el coprocesador, XSW 2.0 es capaz de aprovechar la potencia del *host* en simultáneo. A través de los experimentos realizados, se mostró que la versión KNC de SWIMM resulta competitiva con SWAPHI, siendo capaz de superarlo para secuencias medianas y largas. Complementariamente, la versión híbrida de SWIMM con distribución dinámica (SSE+KNC) superó notablemente a su alternativa XSW 2.0, logrando aceleraciones de hasta 3.9×.

El Capítulo 4 presenta soluciones algorítmicas para sistemas heterogéneos basados en aceleradores FPGA. A diferencia de las implementaciones disponibles en la literatura, se decidió explorar los beneficios de utilizar una tecnología innovadora, como lo es OpenCL en el ámbito de las FPGAs, en lugar de HDLs tradicionales como VHDL o Verilog. En primer lugar, se exploró el rendimiento y el consumo de recursos de diferentes *kernels* de forma de encontrar la configuración más beneficiosa. Posteriormente, se desarrolló una versión híbrida empleando el código de SWIMM. Estas implementaciones también fueron compiladas en una única herramienta a la cual se la llamó OSWALD. Hasta donde llega el conocimiento del tesista, ésta la primera implementación utilizando OpenCL sobre FPGAs para búsquedas de similitud. Además, en base al análisis realizado en el Capítulo 2, es una de las pocas implementaciones que es completamente funcional y general para esta clase de sistemas, además de facilitar la portabilidad por el empleo de OpenCL. Desafortunadamente, la ausencia del código fuente impide una comparación con otras implementaciones basadas en FPGA y aunque un análisis teórico es posible, los diferentes dispositivos, tecnologías y enfoques empleados no sólo complican una comparación directa sino que también dificultan hacerla en forma justa.

- *Medir y analizar el rendimiento y la eficiencia energética de las herramientas seleccionadas y de las propuestas desarrolladas.*

Los Capítulos 3 y 4 evalúan el rendimiento de diferentes sistemas heterogéneos al acelerar las búsquedas de similitud SW. Por su parte, el Capítulo 5 relaciona el rendimiento alcanzado con el consumo de potencia y evalúa la eficiencia energética de estos sistemas. De acuerdo al análisis realizado, se puede afirmar que los sistemas basados únicamente en CPU son capaces de obtener un buen balance entre rendimiento y consumo de potencia a partir de la explotación de multihilado e instrucciones vectoriales, destacándose aquellos que disponen de las extensiones AVX2. Por otra parte, la incorporación de aceleradores al cómputo del *host* demostró mejorar el rendimiento global del sistema en todos los casos, aunque la proporción de mejora varió de acuerdo al acelerador elegido. Desafortunadamente no ocurrió lo mismo en cuanto a la eficiencia energética. Los Xeon Phi no representan una buena opción para este tipo de aplicación. La ausencia de capacidades vectoriales para enteros de bajo rango es la causa principal del pobre rendimiento de este coprocesador, lo que provoca que el incremento en el consumo energético sea mucho mayor que la ganancia de rendimiento. En sentido opuesto, las GPUs sí pueden explotar este tipo de datos y es por ello que se puede aprovechar su gran poder computacional para acelerar los alineamientos. Aunque el aumento en el consumo de potencia es elevado, la mejora lograda por el uso de esta clase de aceleradores resulta superior, lo que se traduce en una mayor eficiencia energética. En

el caso de las FPGAs, su capacidad de reconfiguración hace posible adaptar el hardware a los requerimientos del algoritmo SW. Si bien el rendimiento alcanzable de estos aceleradores puede ser inferior al correspondiente a las GPUs, su bajo consumo de potencia lleva a mayores tasas de eficiencia energética.

Resulta importante mencionar que, si bien se utilizó un conjunto limitado de procesadores y aceleradores para el trabajo experimental, se considera que estos son representativos del resto y por ese motivo es posible generalizar los resultados obtenidos. En el caso de las CPUs, Intel es el amplio dominador del mercado de procesadores. Lo mismo ocurre con las GPUs y la empresa NVIDIA. Los Xeon Phi son los únicos de su tipo y entre los distintos modelos no hay diferencias significativas. En cuanto a las FPGAs, Altera y Xilinx son las empresas que dominan ese mercado. Lamentablemente, no fue posible disponer de alguna placa de Xilinx de forma de complementar el estudio. De todas maneras, ambas empresas ofrecen aceleradores y herramientas con capacidades similares, por lo que se cree que los resultados obtenidos con las FPGAs de Altera son trasladables a las de Xilinx. Una muestra de su paridad es el hecho de que ambas empresas han competido por el liderazgo de mercado durante las últimas décadas y ninguna ha logrado aventajar significativamente a la otra.

Por último, a diferencia de otras evaluaciones de rendimiento y eficiencia energética en el contexto de SW, en esta tesis se han empleado secuencias de consulta y bases de datos representativas de la comunidad bioinformática, potentes arquitecturas orientadas a HPC y eficientes implementaciones que han demostrado ser las más rápidas de su clase. Por ese motivo se considera que la evaluación presentada puede ser de mayor utilidad en el mundo real.

En base al cumplimiento de los objetivos específicos, se considera que se ha cumplido con el objetivo general planteado al inicio de esta investigación. De acuerdo a los resultados obtenidos y a las contribuciones realizadas, se espera que esta tesis aporte a una mayor adopción de SW por parte de la comunidad bioinformática y a un procesamiento más eficiente de los alineamientos de secuencias biológicas en términos de rendimiento y consumo energético.

6.2. Líneas de trabajo futuro

Algunas de las líneas de trabajo futuro que se desprenden de esta tesis pueden ser:

- Desarrollar soluciones algorítmicas para las nuevas generaciones de procesadores Xeon y Xeon Phi. Como se mencionó al final del Capítulo 3, Intel ha anunciado que unificará el conjunto de instrucciones en las próximas generaciones de los Xeon y los Xeon Phi al integrar las extensiones AVX-512 en ambos dispositivos. Este conjunto de instrucciones dispone de capacidades para explotar enteros de bajo rango, lo que representa una de los principales factores para obtener alto rendimiento. Teniendo en cuenta los resultados obtenidos con la metodología empleada, se espera un incremento notable en el rendimiento de ambos dispositivos.

- Explorar la explotación del acelerador PEZY-SC. Este acelerador, del cual poco se sabe [120], ha irrumpido recientemente en los primeros puestos del ranking Green500 y también se ha incorporado al Top500. En base a eso, este acelerador puede ser una buena opción para acelerar las búsquedas de similitud de secuencias biológicas.

- Explorar la explotación de dispositivos no convencionales en HPC para la aceleración del alineamiento de secuencias, como pueden ser los DSPs o las arquitecturas basadas en procesadores ARM. Si bien estos dispositivos poseen capacidades computacionales limitadas, su bajo consumo energético los vuelve atractivos desde el punto de vista de la eficiencia energética.
- Extender la evaluación de rendimiento y eficiencia energética a otros dominios. Así como el algoritmo SW resulta representativo de la bioinformática, interesa extender el análisis realizado a aplicaciones que resulten características de otras áreas.

Referencias

[1] GenBank. [Online]. Available: http://www.ncbi.nlm.nih.gov/genbank/

[2] W. Stallings, *Computer Organization and Architecture. Designing for Performance.* Pre, 2010.

[3] H. Sutter. (2009, 08) The Free Lunch Is Over. A Fundamental Turn Toward Concurrency in Software. Dr Dobb's Journal. [Online]. Available: http://www.gotw.ca/publications/concurrency-ddj.htm

[4] T. Rognes, "Faster Smith-Waterman database searches with inter-sequence SIMD parallelization," *BMC Bioinformatics*, vol. 12:221, 2011.

[5] A. Wozniak, "Using video-oriented instructions to speed up sequence comparison," *Computer applications in the biosciences : CABIOS*, vol. 13, no. 2, pp. 145–150, 1997. [Online]. Available: http://bioinformatics.oxfordjournals.org/content/13/2/145.abstract

[6] T. Rognes and E. Seeberg, "Six-fold speed-up of Smith-Waterman sequence database searches using parallel processing on common microprocessors," *Bioinformatics*, vol. 16, no. 8, pp. 699–706, 2000. [Online]. Available: http://bioinformatics.oxfordjournals.org/content/16/8/699.abstract

[7] M. Farrar, "Striped Smith-Waterman speeds database searches six time over other SIMD implementations," *Bioinformatics*, vol. 23 (2), pp. 156–161, 2007.

[8] B. Alpern, L. Carter, and K. Su Gatlin, "Microparallelism and high-performance protein matching," in *Proceedings of the 1995 ACM/IEEE Conference on Supercomputing*, ser. Supercomputing '95. New York, NY, USA: ACM, 1995. [Online]. Available: http://doi.acm.org/10.1145/224170.224222

[9] W. R. Rudnicki, A. Jankowski, A. Modzelewski, A. Piotrowski, and A. Zadrozny, "The new SIMD implementation of the Smith-Waterman algorithm on Cell microprocessor," *Fundam. Inform.*, vol. 96, no. 1-2, pp. 181–194, 2009. [Online]. Available: http://dx.doi.org/10.3233/FI-2009-173

[10] Top500. [Online]. Available: www.top5000.org

[11] W. Feng, "The importance of being low power in high performance computing," *CTWatch Quarterly*, vol. 1, pp. 12–21, 2005.

[12] W. Feng, X. Feng, and R. Ge, "Green supercomputing comes of age," *IT Professional*, vol. 10, no. 1, pp. 17–23, 2008.

[13] J. Evans, "On performance and energy management in high performance computing systems," in *Parallel Processing Workshops (ICPPW), 2010 39th International Conference on*, Sept 2010, pp. 445–452.

[14] Green500. [Online]. Available: www.green500.org

[15] M. Holliday, "Low-power high performance computing," Master's thesis, The University of Edinburgh, 2012.

[16] J. Dongarra, M. Gates, A. Haidar, Y. Jia, K. Kabir, P. Luszczek, , and S. Tomov, "HPC Programming on Intel Many-Integrated-Core Hardware with MAGMA Port to Xeon Phi," *Scientific Programming*, vol. 2015, p. 11, 2015.

[17] J. Mielikainen, B. Huang, and A. H.-L. Huang, "Intel Xeon Phi accelerated Weather Research and Forecasting (WRF) Goddard microphysics scheme," Geosci. Model Dev. Discuss, Tech. Rep., 2014. [Online]. Available: http://www.geosci-model-dev-discuss.net/7/8941/2014/gmdd-7-8941-2014-print.pdf

[18] A. Vladimirov, "Scientific Computing with Intel Xeon Phi Coprocessors," Colfax International, Tech. Rep., 2014. [Online]. Available: http://www.colfax-intl.com/nd/resources/Files/Stanford-Conference-2015-Scientific-Computing-with-Intel-Xeon-Phi-Coprocessors.pdf

[19] N. Hemsoth. (2014, September) Will 2015 be the year of the FPGA? HPC Wire. [Online]. Available: http://www.hpcwire.com/2014/09/04/will-2015-year-fpga/

[20] (2015, 06) Intel to acquire altera. HPC Wire. [Online]. Available: http://www.hpcwire.com/off-the-wire/intel-to-acquire-altera/

[21] M. Vestias and H. Neto, "Trends of CPU, GPU and FPGA for high-performance computing," in *Field Programmable Logic and Applications (FPL), 2014 24th International Conference on*, Sept 2014, pp. 1–6.

[22] S. O. Settle, "High-performance Dynamic Programming on FPGAs with OpenCL," 2014.

[23] Xilinx Inc. (2015, 12) SDAccel Development Environment. [Online]. Available: http://www.xilinx.com/products/design-tools/software-zone/sdaccel.html

[24] Altera Corporation. (2015, 12) Altera SDK for OpenCL. [Online]. Available: https://www.altera.com/products/design-software/embedded-software-developers/opencl/overview.html

[25] National Center for Biotechnology Information. [Online]. Available: http://www.ncbi.nlm.nih.gov/

[26] T. K. Atwood and D. J. Parry-Smith, *Introducción a la Bioinformática*, Pearson, Ed. Prentice Hall, 2002.

[27] J. Felsenstein, *Inferring phylogenies*. Sinauer Associates, 2003.

[28] M. P. Miller, J. D. Parker, S. W. Rissing, and S. Kumar, "Quantifying the intragenic distribution of human disease mutations," *Annals of Human Genetics*, vol. 67, no. 6, pp. 567–579, 2003.

[29] J. W. Thomas, J. W. Touchman, R. W. Blakesley, G. G. Bouffard, S. M. Beckstrom-Sternberg, E. H. Margulies, M. Blanchette, A. C. Siepel, P. J. Thomas, J. C. McDowell, B. Maskeri, N. F. Hansen, M. S. Schwartz, R. J. Weber, W. J. Kent, D. Karolchik, T. C. Bruen, R. Bevan, D. J. Cutler, S. Schwartz, L. Elnitski, J. R. Idol, A. B. Prasad, S. Q. Lee-Lin, V. V. Maduro, T. J. Summers, M. E. Portnoy, N. L. Dietrich, N. Akhter, K. Ayele, B. Benjamin, K. Cariaga, C. P. Brinkley, S. Y. Brooks, S. Granite, X. Guan, J. Gupta, P. Haghighi, S. L. Ho, M. C. Huang, E. Karlins, P. L. Laric, R. Legaspi, M. J. Lim, Q. L. Maduro, C. A. Masiello, S. D. Mastrian, J. C. McCloskey, R. Pearson, S. Stantripop, E. E. Tiongson, J. T. Tran, C. Tsurgeon, J. L. Vogt, M. A. Walker, K. D. Wetherby, L. S. Wiggins, A. C. Young, L. H. Zhang, K. Osoegawa, B. Zhu, B. Zhao, C. L. Shu, P. J. De Jong, C. E. Lawrence, A. F. Smit, A. Chakravarti, D. Haussler, P. Green, W. Miller, and E. D. Green, "Comparative analyses of multi-species sequences from targeted genomic regions." *Nature*, vol. 424, no. 6950, pp. 788–93, 2003.

[30] E. F. Kirkness, V. Bafna, A. L. Halpern, S. Levy, K. Remington, D. B. Rusch, A. L. Delcher, M. Pop, W. Wang, C. M. Fraser, and J. C. Venter, "The Dog Genome: Survey Sequencing and Comparative Analysis," *Science*, vol. 301, no. 5641, pp. 1898–1903, Sep. 2003.

[31] B. G. Hall, "Simple and accurate estimation of ancestral protein sequences." *Proceedings of the National Academy of Sciences of the United States of America*, vol. 103, no. 14, pp. 5431–5436, Apr. 2006.

[32] B. Schmidt, *Bioinformatics: High Performance Parallel Computer Architectures*, B. Schmidt, Ed. CRC Press, 2010.

[33] S. B. Needleman and C. D. Wunsch, "A general method applicable to the search for similarities in the amino acid sequence of two proteins," *Journal of Molecular Biology*, vol. 48, no. 3, pp. 443–453, Mar. 1970.

[34] A. Isaev, *Introduction to Mathematical Methods in Bioinformatics*. Springer, 2006.

[35] T. F. Smith and M. S. Waterman, "Identification of common molecular subsequences," *Journal of Molecular Biology*, vol. 147, no. 1, pp. 195–197, March 1981.

[36] A. Siegel. (2011, July) What are the advantages/disadvantages of using Protein Alignment vs DNA Alignment? [Online]. Available: https://www.ocf.berkeley.edu/~asiegel/posts/?p=14

[37] F. Opperdoes. Arguments in favour of a phylogenetic analysis of the corresponding protein rather than the dna. [Online]. Available: http://www.icp.ucl.ac.be/~opperd/private/arguments.html

[38] K. Asanovic, R. Bodik, J. Demmel, T. Keaveny, K. Keutzer, J. Kubiatowicz, N. Morgan, D. Patterson, K. Sen, J. Wawrzynek, D. Wessel, and K. Yelick, "A view of the parallel computing landscape," *Commun. ACM*, vol. 52, no. 10, pp. 56–67, Oct. 2009. [Online]. Available: http://doi.acm.org/10.1145/1562764.1562783

[39] Y. Liu and B. Schmidt, "SWAPHI: Smith-Waterman protein database search on Xeon Phi coprocessors," in *25th IEEE International Conference on Application-specific Systems, Architectures and Processors (ASAP 2014)*, 2014.

[40] L. Wang, Y. Chan, X. Duan, H. Lan, X. Meng, and W. Liu, "XSW: Accelerating Biological Database Search on Xeon Phi," in *Fourth International Workshop on Accelerators and Hybrid Exascale Systems 2014*, 2014.

[41] European Bioinformatics Institute. (2012, 12) What is bioinformatics? [Online]. Available: http://www.ebi.ac.uk/

[42] M. A. Kentie, "Biological Sequence Alignment Using Graphics Processing Units," Master's thesis, Faculty of Electrical Engineering, Mathematics and Computer Science. Delft University of Technology, 2010.

[43] P. Higgs and T. Attwood, *Bioinformatics and Molecular Evolution.* Wiley-Blackwell, 2005.

[44] O. Gotoh, "An improved algorithm for matching biological sequences," in *Journal of Molecular Biology*, vol. 162, 1981, pp. 705–708.

[45] E. Vermij, "Genetic sequence alignment on a supercomputing platform," Master's thesis, Faculty of Electrical Engineering, Mathematics and Computer Science. Delft University of Technology., 2011.

[46] S. F. Altschul, W. Gish, W. Miller, E. W. Myers, and D. J. Lipman, "Basic local alignment search tool." *Journal of molecular biology*, vol. 215, no. 3, pp. 403–410, Oct. 1990.

[47] D. Lipman and W. Pearson, "Rapid and sensitive protein similarity searches." *Science*, vol. 227, pp. 1435–1441, 1985.

[48] A. Muhammadzadeh, "MR-CUDASW - GPU accelerated Smith-Waterman algorithm for medium-length (meta)genomic data," Master's thesis, Department of Computer Science. University of Saskatchewan, 2014.

[49] All About The Human Genome Project (HGP). [Online]. Available: http://www.genome.gov/10001772

[50] International Nucleotide Sequence Database Collaboration. [Online]. Available: http://www.insdc.org/

[51] DNA Data Bank of Japan. [Online]. Available: http://www.ddbj.nig.ac.jp/

[52] European Molecular Biology Laboratory. [Online]. Available: http://www.embl.org/

[53] UniProt. [Online]. Available: http://www.uniprot.org/

[54] Reference Sequence (RefSeq). [Online]. Available: http://www.ncbi.nlm.nih.gov/refseq/

[55] G. Hager and G. Wellein, *Introduction to High Performance Computing for Scientists and Engineers*, H. Simon, Ed. CRC P, 2011.

[56] W. Wulf and S. McKee, "Hitting the memory wall: Implications of the obvious," *ACM Computer Architecture News*, vol. 23, pp. 20–24, 1995.

[57] V. Eijkhout, E. Chow, and R. van de Geijn, *Introduction to High-Performance Scientific Computing*, 2015.

[58] M. McCool, "Scalable programming models for massively multicore processors," in *Proceeding of the IEEE*, vol. 96, no. 5, 2007, pp. 816–831.

[59] M. Giles and I. Reguly, "Trends in high-performance computing for engineering calculations," *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, vol. 372, no. 2022, p. 20130319, 2014.

[60] F. J. Pollack, "New Microarchitecture Challenges in the Coming Generations of CMOS Process Technologies (Keynote Address)(Abstract Only)," in *Proceedings of the 32Nd Annual ACM/IEEE International Symposium on Microarchitecture*, ser. MICRO 32. Washington, DC, USA: IEEE Computer Society, 1999, pp. 2–. [Online]. Available: http://dl.acm.org/citation.cfm?id=320080.320082

[61] Intel. (2015, 06) Tecnología intel® turbo boost 2.0. [Online]. Available: http://www.intel.la/content/www/xl/es/architecture-and-technology/turbo-boost/turbo-boost-technology.html

[62] A. Patrizio. (2015, 09) Intel's Xeon roadmap for 2016 leaks. IT World. [Online]. Available: http://www.nextplatform.com/wp-content/uploads/2015/05/intel-kdm-roadmap-1.jpg

[63] U. Pirzada. (2015, 09) Intel's Skylake Purley Family of Microprocessors Will Boast upto 28 Cores and 56 Threads - Next Generation Xeon Platform Landing in 2016. [Online]. Available: http://wccftech.com/intel-skylake-purley-platform-upto-28-cores-56-threads/

[64] B. Howse and R. Smith. (2015, 07) Tick Tock On The Rocks: Intel Delays 10nm, Adds 3rd Gen 14nm Core Product Kaby Lake. [Online]. Available: http://www.anandtech.com/show/9447/intel-10nm-and-kaby-lake

[65] P. Bright. (2015, 07) Intel confirms tick-tock-shattering Kaby Lake processor as Moore s Law falters. [Online]. Available: http://arstechnica.com/gadgets/2015/07/intel-confirms-tick-tock-shattering-kaby-lake-processor-as-moores-law-falters/

[66] M. Crider. (2015, 07) APU, GPU, WTF? A guide to AMD s desktop processor line-up. [Online]. Available: http://www.digitaltrends.com/computing/apu-gpu-wtf-a-guide-to-amds-desktop-processor-line-up/

[67] K. Moammer. (2015, 10) Amd zen cpu microarchitecture details leaked in patch - doubles down on ipc and floating point throughput. [Online]. Available: http://wccftech.com/amd-zen-cpu-core-microarchitecture-detailed/2/

[68] William Robert Dawson Boyd III, "Massively parallel algorithms for method of characteristics neutral particle transport on shared memory computer architectures," Master's thesis, Massachusetts Institute of Technology, 2014.

[69] M. Harris. (2013, 11) Unified memory in cuda 6. NVIDIA. [Online]. Available: http://devblogs.nvidia.com/parallelforall/unified-memory-in-cuda-6/

[70] B. Leback, D. Miles, and M. Wolfe, "Tesla vs. Xeon Phi vs. Radeon. A Compiler Writer's Perspective," PGI, Tech. Rep., 2013.

[71] A. R. Brodtkorb, C. Dyken, T. R. Hagen, J. M. Hjelmervik, and O. O. Storaasli., "State-of-the-art in heterogeneous computing," *Programming*, vol. 18, pp. 1–33, 2010.

[72] OpenACC organization. (2015, 12) OpenACC. [Online]. Available: http://www.openacc.org/

[73] A. R. Brodtkorb, T. R. Hagen, and M. L. Sætra, "Graphics processing unit (GPU) programming strategies and trends in GPU computing," *Journal of Parallel and Distributed Computing*, vol. 73, pp. 4–13, 2013.

[74] S. Cook, *CUDA Programming: A Developer's Guide to Parallel Computing with GPUs*, T. Green and R. Day, Eds. Morgan Kauffman, 2013.

[75] M. Harris. (2014, 9) Maxwell: The Most Advanced CUDA GPU Ever Made. [Online]. Available: http://devblogs.nvidia.com/parallelforall/maxwell-most-advanced-cuda-gpu-ever-made/

[76] NVIDIA, "NVIDIA GeForce GTX 980 White Paper."

[77] R. Smith. (2011, 12) AMD's Graphics Core Next Preview: AMD's New GPU, Architected For Compute. AnadTech. [Online]. Available: http://www.anandtech.com/show/4455/amds-graphics-core-next-preview-amd-architects-for-compute

[78] AMD, "AMD Graphics Core Next (GCN) Architecture - White paper," AMD, Tech. Rep., 2012. [Online]. Available: https://www.amd.com/Documents/GCN_Architecture_whitepaper.pdf

[79] ——. (2015, 11) High bandwidth memory. [Online]. Available: http://www.amd.com/en-us/innovations/software-technologies/hbm

[80] L. Seiler, D. Carmean, E. Sprangle, T. Forsyth, P. Dubey, S. Junkins, A. Lake, R. Cavin, R. Espasa, E. Grochowski, T. Juan, M. Abrash, J. Sugerman, and P. Hanrahan, "Larrabee: A many-core x86 architecture for visual computing," *IEEE Micro*, vol. 29, no. 1, pp. 10–21, 2009.

[81] U. Pirzada. (2015, 07) IntelŽs Xeon Phi 14nm Knights Landing Co-Processors Detailed - OmniPath Architecture 100 Series and 16GB HMC on a 2.5D Interposer. WCCF Tech. [Online]. Available: http://wccftech.com/intel-14nm-knight-landing-xeon-phi-co-processor-omni-path-100/

[82] M. N. M. Isa, "High performance reconfigurable architectures for biological sequence alignment," Ph.D. dissertation, The University Of Edinburgh, 2013.

[83] R. Garg. (2013, 10) A Look at Altera's OpenCL SDK for FPGAs. AnadTech. [Online]. Available: http://www.anandtech.com/show/7334/a-look-at-alteras-opencl-sdk-for-fpgas

[84] T. Rauber and G. Rünger., *Parallel Programming for Multicore and Cluster Systems*. Springer-Verlag Berlin Heidelberg, 2010.

[85] J. Dongarra, I. Foster, G. Fox, W. Gropp, K. Kennedy, L. Torczon, and A. White, *The Sourcebook of Parallel Computing*. Morgan Kauffman, 2003.

[86] OpenMP Architecture Review Board. (2013, 07) OpenMP Application Program Interface - Version 4.0. [Online]. Available: http://www.openmp.org/mp-documents/OpenMP4.0.0.pdf

[87] NVIDIA. (2015, 08) CUDA C Programming Guide. [Online]. Available: http://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html

[88] Altera Corporation, "Altera SDK for OpenCL Programming Guide, Version 14.0," 2014.

[89] Khronos Group, "The OpenCL Specification. version 2.0," 2014. [Online]. Available: https://www.khronos.org/registry/cl/specs/opencl-2.0.pdf

[90] ——, "The OpenCL Specification. version 1.0," 2009. [Online]. Available: https://www.khronos.org/registry/cl/specs/opencl-1.0.pdf

[91] ——, "The OpenCL Specification. version 1.1," 2011. [Online]. Available: https://www.khronos.org/registry/cl/specs/opencl-1.1.pdf

[92] ——, "The OpenCL Specification. version 1.2," 2012. [Online]. Available: https://www.khronos.org/registry/cl/specs/opencl-1.2.pdf

[93] S. Demirsoy. (2013) How OpenCL enables easy access to FPGA performance? [Online]. Available: https://www.oerc.ox.ac.uk/sites/default/files/uploads/OpenCL_presentation_oxford.pdf

[94] F. D. Igual, C. García, G. Botella, L. Piñuel, M. Prieto-Matías, and F. Tirado, "Non-negative matrix factorization on low-power architectures and accelerators: A comparative study," *Computers & Electrical Engineering*, vol. 46, pp. 139–156, 2015. [Online]. Available: http://dx.doi.org/10.1016/j.compeleceng.2015.03.035

[95] Y. Liu, D. L. Maskell, and B. Schmidt, "CUDASW++: optimizing Smith-Waterman sequence database searches for CUDA-enabled graphics processing units," *BMC Research Notes*, vol. 2:73, 2009.

[96] M. Farrar. Optimizing Smith-Waterman for the Cell Broad-band Engine. [Online]. Available: http://farrar.michael.googlepages.com/SW-CellBE.pdf

[97] A. Szalkowski, C. Ledergerber, P. Krahenbuhl, and C. Dessimoz, "SWPS3 - fast multi-threaded vectorized Smith-Waterman for IBM Cell/B.E. and x86/SSE2," *BMC Research Notes*, vol. 1, p. 107, 2008.

[98] M. Zhao, W.-P. Lee, E. P. Garrison, and G. T. Marth, "SSW Library: An SIMD Smith-Waterman C/C++ Library for Use in Genomic Applications," *PLoS ONE*, vol. 8, no. 12, p. e82138, 12 2013. [Online]. Available: http://dx.doi.org/10.1371%2Fjournal.pone.0082138

[99] W. Liu, B. Schmidt, G. Voss, A. Schroder, and W. Muller-Wittig, "Bio-sequence database scanning on a GPU," in *Parallel and Distributed Processing Symposium, 2006. IPDPS 2006. 20th International*, April 2006, pp. 8 pp.–.

[100] Y. Liu, W. Huang, J. Johnson, and S. Vaidya, "GPU Accelerated Smith-Waterman," *Lecture Notes in Computer Science*, vol. 3994, pp. 188–195, 2006.

[101] S. Manavski and G. Valle, "CUDA compatible GPU cards as efficient hardware accelerators for Smith-Waterman sequence alignment," *BMC Bioinformatics*, vol. 9, no. Suppl 2, p. S10, 2008. [Online]. Available: http://www.biomedcentral.com/1471-2105/9/S2/S10

[102] Y. Liu, B. Schmidt, and D. Maskell, "CUDASW++2.0: enhanced Smith-Waterman protein database search on CUDA-enabled GPUs based on SIMT and virtualized SIMD abstractions," *BMC Research Notes*, vol. 3, no. 1, p. 93, 2010. [Online]. Available: http://www.biomedcentral.com/1756-0500/3/93

[103] D. Zou, Y. Dou, and F. Xia, "Optimization schemes and performance evaluation of Smith-Waterman algorithm on CPU, GPU and FPGA," *Concurrency and Computation: Practice and Experience*, vol. 24, no. 14, pp. 1625–1644, 2012. [Online]. Available: http://dx.doi.org/10.1002/cpe.1913

[104] A. Khalafallah, H. Elbabb, O. Mahmoud, and A. Elshamy, "Optimizing Smith-Waterman algorithm on Graphics Processing Unit," in *Computer Technology and Development (ICCTD), 2010 2nd International Conference on*, Nov 2010, pp. 650–654.

[105] P. Borovska and M. Lazarova, "Parallel Models for Sequence Alignment on CPU and GPU," in *Proceedings of the 12th International Conference on Computer Systems and Technologies*, ser. CompSysTech '11. New York, NY, USA: ACM, 2011, pp. 210–215. [Online]. Available: http://doi.acm.org/10.1145/2023607.2023644

[106] Y. Liu, A. Wirawan, and B. Schmidt, "CUDASW++ 3.0: accelerating Smith-Waterman protein database search by coupling CPU and GPU SIMD instructions," vol. 14:117, 2013.

[107] L. Hasan and Z. Al-Ars, *Computational Biology and Applied Bioinformatics*. InTech, September 2011, ch. 9, pp. 187–202.

[108] S. Dydel and P. Bala, "Large Scale Protein Sequence Alignment Using FPGA Reprogrammable Logic Devices," in *Field Programmable Logic and Application*, ser. Lecture Notes in Computer Science, J. Becker, M. Platzner, and S. Vernalde, Eds. Springer Berlin Heidelberg, 2004, vol. 3203, pp. 23–32. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-30117-2_5

[109] T. Oliver, B. Schmidt, and D. Maskell, "Reconfigurable architectures for bio-sequence database scanning on fpgas," *Circuits and Systems II: Express Briefs, IEEE Transactions on*, vol. 52, no. 12, pp. 851–855, Dec 2005.

[110] P. Zhang, G. Tan, and G. R. Gao, "Implementation of the smith-waterman algorithm on a reconfigurable supercomputing platform," in *Proceedings of the 1st International Workshop on High-performance Reconfigurable Computing Technology and Applications: Held in Conjunction with SC07*, ser. HPRCTA

'07. New York, NY, USA: ACM, 2007, pp. 39–48. [Online]. Available: http://doi.acm.org/10.1145/1328554.1328565

[111] T. Van Court and M. Herbordt, "Families of FPGA-based algorithms for approximate string matching," in *Application-Specific Systems, Architectures and Processors, 2004. Proceedings. 15th IEEE International Conference on*, Sept 2004, pp. 354–364.

[112] K. Benkrid, Y. Liu, and A. Benkrid, "A Highly Parameterized and Efficient FPGA-Based Skeleton for Pairwise Biological Sequence Alignment," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 17, no. 4, pp. 561–570, April 2009.

[113] M. Isa, K. Benkrid, T. Clayton, C. Ling, and A. Erdogan, "An FPGA-based parameterised and scalable optimal solutions for pairwise biological sequence analysis," in *Adaptive Hardware and Systems (AHS), 2011 NASA/ESA Conference on*, June 2011, pp. 344–351.

[114] K. Benkrid, A. Akoglu, C. Ling, Y. Song, Y. Liu, and X. Tian, "High Performance Biological Pairwise Sequence Alignment: FPGA Versus GPU Versus Cell BE Versus GPP," *Int. J. Reconfig. Comput.*, vol. 2012, pp. 7:7–7:7, Jan. 2012. [Online]. Available: http://dx.doi.org/10.1155/2012/752910

[115] L. Wang, Y. Chan, X. Duan, H. Lan, X. Meng, and W. Liu. (2014) XSW 2.0: A fast Smith-Waterman Algorithm Implementation on Intel Xeon Phi Coprocessors. [Online]. Available: http://sdu-hpcl.github.io/XSW/

[116] E. Rucci, "Computación eficiente del alineamiento de secuencias de adn sobre cluster de multicores," Master's thesis, Universidad Nacional de La Plata, Argentina. Available online at: http://hdl.handle.net/10915/27737, 2013.

[117] "Front matter," in *High Performance Parallelism Pearls*, J. R. Jeffers, Ed. Boston: Morgan Kaufmann, 2015, pp. i – ii.

[118] F. D. Igual, L. M. Jara, J. I. Gómez-Pérez, L. Piñuel, and M. Prieto-Matías, "A power measurement environment for pcie accelerators," *Computer Science - Research and Development*, vol. 30, no. 2, pp. 115–124, 2014. [Online]. Available: http://dx.doi.org/10.1007/s00450-014-0266-8

[119] S. Barrachina, M. Barreda, S. Catalán, M. F. Dolz, G. Fabregat, R. Mayo, and E. Quintana-Ortí, "An integrated framework for power-performance analysis of parallel scientific workloads," pp. 114–119, 2013.

[120] V. Hindriksen. (2015, 08) The knowns and unknowns of the PEZY-SC accelerator at RIKEN. Stream Computing. [Online]. Available: http://streamcomputing.eu/blog/2015-08-02/the-knowns-and-unknowns-of-the-pezy-sc-accelerator-at-riken/