TÍTULO

"SMITH-WATERMAN" PARALELO EN ARQUITECTURA DE *MANY-CORE* PARA BÚSQUEDAS EN BASES DE DATOS DE SECUENCIAS

AUTOR

Juan Lago Cabrera

Esta edición electrónica ha sido realizada en 2011

Director	Sergio Gálvez Rojas
Co-Tutores	Oswaldo Trelles Salazar y Gabriel Dorado Pérez
Curso	Máster en Bioinformática (2009/2010)
ISBN	978-84-694-5068-0
©	Juan Lago Cabrera
©	Para esta edición, la Universidad Internacional de Andalucía

Universidad Internacional de Andalucía, 2011

Reconocimiento-No comercial-Sin obras derivadas

Usted es libre de:

- Copiar, distribuir y comunicar públicamente la obra.

Bajo las condiciones siguientes:

- **Reconocimiento**. Debe reconocer los créditos de la obra de la manera. especificada por el autor o el licenciador (pero no de una manera que sugiera que tiene su apoyo o apoyan el uso que hace de su obra).
- **No comercial**. No puede utilizar esta obra para fines comerciales.
- **Sin obras derivadas**. No se puede alterar, transformar o generar una obra derivada a partir de esta obra.

- *Al reutilizar o distribuir la obra, tiene que dejar bien claro los términos de la licencia de esta obra.*
- *Alguna de estas condiciones puede no aplicarse si se obtiene el permiso del titular de los derechos de autor.*
- *Nada en esta licencia menoscaba o restringe los derechos morales del autor.*

Memoria del Proyecto de Fin de Máster

“Smith-Waterman” paralelo en arquitectura de many-core para búsquedas en bases de datos de secuencias

Autor: Juan Lago Cabrera

Tutor: Sergio Gálvez Rojas

Co-tutores: Oswaldo Trelles Salazar y Gabriel Dorado Pérez

Málaga, 10 de Abril de 2011

Universidad Internacional de Andalucía

Máster en Bioinformática (Curso 2009-2010)

Dedicado a Marisa, Iván y Daniel, mis grandes proyectos vitales.

Juan Lago

Contenido

Resumen del proyecto 1

Esta memoria se corresponde con el proyecto fin de Máster titulado **Algoritmo "Smith-Waterman" paralelo en arquitectura de microprocesador de muchos núcleos ("many-core") para búsquedas en bases de datos de secuencias de ácidos nucleicos y péptidos** del Máster en Bioinformática impartido por la Universidad Internacional de Andalucía en su edición 2009/2010.

Con este proyecto se pretende poner en práctica los conocimientos adquiridos durante el periodo de docencia del Máster, así como ampliar y profundizar en algunas áreas de interés relacionadas con el campo de la bioinformática desde un punto de vista práctico y útil para la comunidad científica.

El presente trabajo parte de un desarrollo previo consistente en una implementación paralela del algoritmo "Smith-Waterman" de alineamiento local de secuencias de ácidos nucleicos y péptidos sobre arquitectura hardware de microprocesador con muchos núcleos (del inglés, "many-core"), trabajo realizado dentro del grupo de investigación del Plan Andaluz de Investigación, Desarrollo e Innovación (PAIDI) denominado "Biotecnología Agroalimentaria" (Código AGR-248), con recursos cedidos para el presente proyecto como la tarjeta TILExpress-20G del fabricante Tilera <http://www.tilera.com>. Una descripción de este hardware se realiza en el capítulo "**Arquitectura hardware y software**".

Aunque esta implementación es igualmente válida para secuencias de residuos de nucleótidos de ácidos nucleicos, como el ácido desoxirribonucleico (ADN) y el ácido ribonucleico (ARN), así como de residuos de aminoácidos de péptidos (como las proteínas), el presente trabajo se centra en la comparación de secuencias de gran tamaño de 80 kilobases (kb) a 260 kb de longitud, siendo por tanto idónea para la comparación de secuencias de ADN (las secuencias de proteínas son muchos menores, con un tamaño medio de unos 300 residuos de aminoácidos, y llegando las mayores sólo hasta unos 27.000 residuos).

En este trabajo se modifica y extiende el uso de esta implementación del algoritmo para su utilización en búsquedas en bases de datos de secuencias mediante programación de guiones (del inglés, "scripts") en lenguaje Perl, que tiene amplia presencia en el ámbito de la bioinformática con multitud de referencias, códigos fuente y módulos disponibles, destacando especialmente la colección de módulos BioPerl <http://www.bioperl.org>.

La posibilidad de poder realizar alineamientos de gran tamaño de una secuencia problema contra una base de datos bajo una arquitectura con muchos núcleos es, por tanto, uno de los elementos diferenciadores del presente proyecto. También se justificará el ahorro de tiempo que se consigue al paralelizar varios alineamientos simultáneos utilizando la arquitectura con muchos núcleos disponible para el proyecto y que será descrita en los siguientes apartados.

Por tanto, los objetivos formativos e investigadores que se persiguen con este trabajo son:

i) Familiarización con el desarrollo de software en arquitecturas de vanguardia altamente paralelas.

ii) Revisión del estado del arte en la comparación de secuencias de ácidos nucleicos y péptidos desde un punto de vista algorítmico y también bajo implementaciones paralelas. Revisión de estudios en los que se mencionan alineamientos de secuencias de gran tamaño.

iii) Extensión de la implementación del algoritmo Smith-Waterman sobre arquitectura paralela utilizando hardware con muchos núcleos para su uso en la comparación de una secuencia interrogante (del inglés, "query") contra bases de datos en secuencias de gran tamaño.

iv) Desarrollo de guiones en lenguaje Perl para resolución de tareas complejas.

v) Estudio comparativo entre el trabajo realizado con otras implementaciones paralelas ampliamente referenciadas como es el caso del algoritmo CUDASW++. También se incluye una comparación con BLAST.

Para conseguir los objetivos propuestos, la presente memoria se estructura en varios capítulos.

El capítulo 2 (**Introducción**) presenta de forma detallada los objetivos del proyecto realizando una revisión bibliográfica del estado del arte.

En el capítulo 3 (**Fundamentos biológicos**) se realiza un análisis de los tamaños de varias de las bases de datos de secuencias de ácidos nucleicos más conocidas y utilizadas en la actualidad, centrándose en la justificación biológica de la comparación de secuencias de gran tamaño.

En el capítulo 4 (**Fundamentos algorítmicos**) se introducen los fundamentos de la comparación de secuencias. En particular, el problema de encontrar la subcadena común más larga entre dos cadenas de caracteres, así como la diferencia entre subcadenas y subsecuencias. Asimismo, se revisan las matrices de puntuación utilizadas y el esquema de penalización por la introducción de huecos (del inglés "gaps") durante el alineamiento. Posteriormente se realiza un repaso de varios algoritmos involucrados en la resolución de este tipo de problemas como son los basados en programación dinámica, codiciosa (del inglés, "greedy") y árbol de sufijos (del inglés, "suffix-tree"). Por último, se tratan también aspectos de complejidad algorítmica y se introduce la implementación utilizada en este proyecto, basada en el algoritmo de alineamiento de espacio lineal rápido (del inglés, "Fast Linear-Space Alignment"; FastLSA).

El capítulo 5 (**Arquitectura hardware y software**) presenta los entornos utilizados durante el desarrollo de la implementación paralela del algoritmo Smith-Waterman, así como el diseño del modelo de programación.

El capítulo 6 (**Implementación y resultados**) destaca algunos aspectos de la implementación realizada y una comparativa de resultados con otras implementaciones de referencia, como es el caso del algoritmo CUDASW++, analizando las ventajas e inconvenientes que presentan.

Finalmente, en los capítulos 7 y 8 (**Trabajos futuros y Conclusiones**, respectivamente) se proponen futuras líneas de trabajo y se resumen las principales conclusiones obtenidas durante la realización del proyecto.

Introducción 2

La bioinformática está alcanzando una relevancia cada vez mayor en las ciencias de la vida [1]. Las herramientas bioinformáticas más utilizadas en genómica y proteómica son las que permiten realizar búsquedas de semejanzas y diferencias entre secuencias de ácidos nucleicos (ADN y ARN) y péptidos. Existen tres variantes de dichas búsquedas:

i) Identificación de semejanzas entre distintas secuencias generadas mediante equipos de secuenciación de ácidos nucleicos (muchas secuencias cortas de decenas a cientos de bases), a fin de ensamblarlas en una secuencia continua no fragmentada (del inglés, "contig", que es una contracción de "contiguous"), como el algoritmo CAP3 [2].

ii) Alineamiento entre dos o más secuencias, como es el caso del algoritmo ClustalW [3].

iii) Identificación de semejanzas de una secuencia determinada con las secuencias existentes en bases de datos, como es el algoritmo representado por la herramienta de búsqueda de alineamiento local básico (del inglés, "Basic Local Alignment Search Tool"; BLAST) [4].

En relación a este último tipo de algoritmos (búsquedas de semejanzas en bases de datos), la identificación de las afinidades de una secuencia interrogante (o secuencia problema) permite determinar si existen secuencias que posean un nivel de semejanza igual o superior al umbral especificado por el usuario. El examen posterior de dichas secuencias y, en su caso, las anotaciones asociadas a las regiones comunes, pueden aportar información valiosa acerca de la información genética almacenada en la secuencia problema. Dicho con otras palabras, de este modo es posible inferir la función (naturaleza biológica) de una secuencia desconocida, por comparación con las ya publicadas e identificadas (p.ej., revelar la identidad inicialmente desconocida de un gen clonado y secuenciado).

En relación al segundo tipo de herramientas bioinformáticas descritas (alineamiento de secuencias), el algoritmo Smith-Waterman [5] permite encontrar cadenas comunes entre dos secuencias. No obstante, su coste cuadrático tanto en tiempo como en espacio de almacenamiento ha hecho que, en la práctica, su uso haya quedado relegado a ciertos casos de poca envergadura en los que dicho coste sea asumible. Este método de alineamiento local es especialmente inviable cuando las longitudes de las secuencias a comparar aumentan, ya que en tales casos se incrementan de forma significativa los tiempos de computación y requerimientos de memoria.

Respecto al tercer tipo de herramientas descritas, el algoritmo BLAST es un programa de amplio uso por su utilidad y calidad de resultados (del inglés, "hits") que es capaz de encontrar. De hecho, se trata del algoritmo bioinformático más usado por la comunidad científica. Por ello ha sido ampliamente discutido y refinado, existiendo variantes a las que se han añadido parámetros nuevos para aumentar su grado de afinamiento (véanse referencias [6,

7]). Incluso se han implementado versiones paralelas para dotarlo de mayor rendimiento (véanse referencias [8, 9]).

A pesar de ello, el BLAST recibió críticas [10] por tratarse de un algoritmo heurístico. De hecho, existe una propuesta para sustituir su aplicación –bajo determinadas circunstancias– por el óptimo Smith-Waterman, debido, especialmente, al carácter altamente paralelizable de éste último. Todas estas líneas se basan en el uso de múltiples elementos de proceso de alto rendimiento, tales como matrices de puerta de campo programable (del inglés, "Field Programmable Gate Arrays"; FPGA) [11, 12], unidades de procesamiento gráfico (del inglés, "Graphics Processing Unit"; GPU) [13, 14], uso simultáneo de varios núcleos en cada microprocesador [15, 16], e incluso soluciones híbridas en las que se usa una unidad de procesamiento central (del inglés, "Central Processing Unit"; CPU) y varias unidades GPU, como es el caso de la arquitectura de microprocesador de motor de célula de banda ancha (del inglés, "Cell Broadband Engine Architecture"; CBEA) [15, 17].

En este sentido, el uso de la arquitectura de dispositivo de cómputo unificado (del inglés, "Compute Unified Device Architecture"; CUDA) para la programación de aplicaciones en GPU de propósito general (del inglés, "General Purpose GPU"; GPGPU) se encuentra especialmente extendido, habiéndose obtenido con su utilización tan buenos resultados que compañías tan importantes como nVidia se han incorporado al mercado de la supercomputación (del inglés, "High Performance Computing"; HPC) con máquinas como la Tesla serie 20 [18]. Sin embargo, las arquitecturas de hardware de tipo CUDA y GPU tienen importantes deficiencias como, por ejemplo, bajos recursos de memoria caché, lo que hace que se incurra en una alta latencia cuando se accede a una cantidad de datos relativamente grande. Todo ello hace que algunos algoritmos no se puedan paralelizar eficientemente en estas plataformas.

Es en este punto donde las tecnologías de sistema en microprocesador de muchos núcleos (del inglés, "Many-Core System on Chip"; MCSoC) pueden aportar una gran potencia de cálculo a la par que una gran facilidad en el desarrollo de aplicaciones paralelas. Básicamente, un microprocesador MCSoC está formado por un alto número de núcleos que incorporan todos los recursos necesarios para poder ejecutar por sí mismos un sistema operativo (p.ej., Linux reducido). Por otro lado, existe el concepto de teja, azulejo o baldosa (del inglés, "tile") que puede considerarse como el mínimo bloque del microprocesador que posee circuitería independiente para poderse comunicar con el resto de tejas y con la memoria principal.

El microprocesador Intel de nube en chip único (del inglés, "Single-Chip Cloud"; SCC) es un buen exponente de esta tecnología: se encuentra formado por 48 núcleos en 24 tejas, de manera que cada dos núcleos comparten circuitería básica [19]. El microprocesador Intel "Knights Ferry" con arquitectura de muchos núcleos integrados (del inglés, "Many Integrated Core"; MIC) es otro modelo que utiliza dicha tecnología, y comparte con el anterior el hecho de que cada núcleo posee una arquitectura x86 para la que existen abundantes herramientas de software, lo cual puede facilitar el desarrollo de aplicaciones informáticas en dicho entorno. A

pesar de ello, Intel no comercializa todavía estos productos, que se encuentran en fase de experimentación por parte de la comunidad científica y de desarrolladores.

No sucede lo mismo con el microprocesador Tile64 de la compañía Tilera <http://www.tilera.com> [20]. Se trata de un microprocesador SoC que incorpora 64 tejas que pueden funcionar a 866 MHz cada una, con arquitectura de computación de conjunto de instrucciones reducidas (del inglés, "Reduced Instruction Set Computing"; RISC). La compañía Tilera ha solucionado los problemas de calentamiento que se producen en otros microprocesadores (como los de Intel) a costa de reducir las capacidades de cada una de sus tejas: frecuencia de reloj reducida en comparación con otros microprocesadores y carencia de coma flotante nativa. Por lo demás, este microprocesador está especialmente pensado para conmutación de paquetes y, para ello, incorpora una red interna a modo de malla inteligente (del inglés, "intelligent Mesh"; iMesh), que proporciona un ancho de banda interno de unos 31 Tbps y una excelente gestión de las memorias caché de nivel 2 y 3 (L2 y L3, respectivamente). Simplificando, puede considerarse que el microprocesador Tile64 está compuesto por 64 microprocesadores completos dentro de una pastilla, pero que trabajan con una memoria compartida común. Este chip se encuentra integrado en una tarjeta exprés de interconexión de componente periférico (del inglés, "Peripheral Component Interconnect Express"; PCIe) con circuitería de comunicaciones y cierta cantidad de memoria de acceso aleatorio dinámico (del inglés, "Dynamic Random Access Memory"; DRAM) que hace las veces de memoria principal compartida y disco de estado sólido (del inglés, "Solid State Disk"; SSD). En el caso de la tarjeta TILExpress-20G, este tamaño de memoria es de 8 GB.

Habiendo quedado ya patente el potencial de este microprocesador para ejecutar el algoritmo Needleman-Wunsch (alineamiento global de secuencias) [21], procede estudiar su rendimiento en la ejecución del algoritmo Smith-Waterman (alineamiento local) en el presente trabajo, siendo este tipo de alineamiento el que tiene sentido para comparar secuencias remotamente emparentadas o con bajo grado de semejanza. El rendimiento se estudiará entre una secuencia problema y una base de datos de secuencias, ambas previamente cargadas en el sistema de ficheros del ordenador servidor, anfitrión u hospedador (del inglés, "host") que posee una tarjeta TILExpress-20G. Igualmente, se explorarán los límites admisibles de dicho algoritmo, tanto en longitud de la secuencia problema como en tamaño y longitud media de las secuencias de la base de datos en la que buscar. Estos aspectos deberían permitir superar las limitaciones de las GPU en este tipo de aplicaciones. Los resultados se contrastarán con las correspondientes implementaciones de última generación realizadas en CUDA para GPU y los métodos heurísticos ya establecidos como el BLAST, atendiendo tanto a velocidad como a calidad de resultados obtenidos.

Fundamentos biológicos 3

El análisis de las secuencias de ácidos nucleicos y péptidos se utiliza en multitud de situaciones, como son la construcción de árboles filogenéticos (dendrogramas), predicción de intrones/exones, zonas reguladoras y secuencias codificantes (del inglés, “CoDing Sequence”; CDS) o marcos abiertos de lectura (del inglés, “Open Reading-Frame”; ORF), entre otras.

El alineamiento de secuencias es una de las operaciones más utilizadas en la investigación de biología computacional. De este modo, pueden compararse, por ejemplo, las secuencias del proyecto Genoma Humano con otras secuencias (de la misma y de otras especies), a fin de identificar identidades y diferencias entre ellas, identificación de genes, posibles funciones comunes, etc.

Como un ejemplo práctico, la identificación de genes y el rol que desempeñan en los mecanismos de desarrollo de enfermedades han centrado gran parte de la atención de la comunidad científica. El alineamiento de secuencias puede contribuir a este objetivo. Así, por ejemplo, algunos científicos de la Universidad de Boston han descubierto recientemente [22] algunos marcadores genéticos responsables de la longevidad y se espera que pronto se pueda ofrecer un test para pronosticar la esperanza de vida de una persona, según su acervo genético (genoma). Asimismo, dicha herramienta bioinformática puede emplearse para desarrollar marcadores moleculares que sirvan para la mejora genética, el control de calidad y la trazabilidad [21].

Cada día se secuencian más genomas completos de diferentes organismos que incrementan, de forma exponencial, el tamaño de las bases de datos que los almacenan. Dicha información representa un recurso muy valioso para los investigadores en ciencias de la vida, pero al mismo tiempo puede constituir un problema computacional que también crece exponencialmente. Se necesitan por tanto métodos fiables, eficientes y baratos de alineamiento de secuencias para obtener el máximo beneficio de estas enormes bases de datos.

En este apartado de la memoria se hace una revisión de las principales bases de datos de ácido nucleicos y péptidos con objeto de conocer las frecuencias y los tamaños de las secuencias almacenadas. Asimismo, se hace una revisión bibliográfica en la que se encuentran referencias concretas de comparación de secuencias de gran tamaño.

Estudio de bases de datos de ácidos nucleicos y péptidos

La tabla 3.1 muestra la distribución de secuencias, según su tamaño, en varias de las bases de datos de ácidos nucleicos, como son GenBank del “National Center for Biotechnology Information” (NCBI), BaNk del “European Molecular Biology Laboratory” (EMBL) y (UniProtKB). Para la elaboración de esta tabla, se han realizado distintas búsquedas a fecha

17 de septiembre de 2010, según la información contenida en ellas. Aunque estos valores se incrementan constantemente, la tabla puede servir como aproximación válida para el análisis de tamaños que se quiere realizar. La figura 3.1 muestra esta misma distribución de forma gráfica.

Tabla 3.1. Tamaños de secuencias y su porcentaje en bases de datos de ácidos nucleicos y péptidos. Los datos fueron obtenidos a fecha de 17 de septiembre de 2010.

Rango de tamaños (residuos)	Número de entradas y porcentaje en bases de datos							
	Datos a 17/09/2010						Desglosado	
	EMBL-Bank	%	NCBI-Gen-Bank	%	UniProtKB (proteínas)	%	UniProtKB/ Swiss-Prot	UniProtKB/ TrEMBL
0-50	11.041.818	5,79%	10.213.240	5,09%	403.557	3,32%	11.324	392.233
51-100	12.961.279	6,80%	12.949.413	6,46%	1.253.094	10,31%	41.098	1.211.996
101-200	16.898.000	8,87%	17.023.040	8,49%	2.832.858	23,31%	113.761	2.719.097
201-300	11.967.423	6,28%	12.098.662	6,03%	2.545.217	20,94%	104.086	2.441.131
301-400	11.065.229	5,81%	11.185.895	5,58%	1.922.726	15,82%	90.755	1.831.971
401-500	14.574.183	7,65%	14.713.904	7,34%	1.290.189	10,61%	62.041	1.228.148
501-600	17.016.290	8,93%	17.256.763	8,60%	674.950	5,55%	33.468	641.482
601-700	18.335.159	9,62%	18.609.848	9,28%	379.771	3,12%	19.962	359.809
701-800	19.645.861	10,31%	20.014.114	9,98%	267.917	2,20%	11.880	256.037
801-900	14.775.222	7,75%	15.339.618	7,65%	178.860	1,47%	9.132	169.728
901-1k	7.228.176	3,79%	7.801.604	3,89%	105.482	0,87%	6.246	99.236
1k-2k	19.629.058	10,30%	24.320.861	12,13%	260.378	2,14%	13.520	246.858
2k-5k	9.354.946	4,91%	11.060.555	5,51%	38.300	0,32%	1.940	36.360
5k-10k	2.799.244	1,47%	3.584.731	1,79%	2.083	0,02%	121	1.962
10k-25k	1.895.269	0,99%	2.429.095	1,21%	160	0,00%	12	148
25k-50k	694.048	0,36%	917.813	0,46%	11	0,00%	2	9
50k-100k	339.716	0,18%	480.132	0,24%	0	0,00%	0	0
100k-200k	264.409	0,14%	345.576	0,17%	0	0,00%	0	0
200k-300k	73.112	0,04%	103.808	0,05%	0	0,00%	0	0
300k-400k	11.580	0,01%	28.692	0,01%	0	0,00%	0	0
400k-500k	4.239	0,00%	15.272	0,01%	0	0,00%	0	0
500k-1M	4.922	0,00%	29.926	0,01%	0	0,00%	0	0
>1M	3.443	0,00%	34.889	0,02%	0	0,00%	0	0
	190.582.626	**100,00%**	**200.557.451**	**100,00%**	**12.155.553**	**100,00%**	**519.348**	**11.636.205**

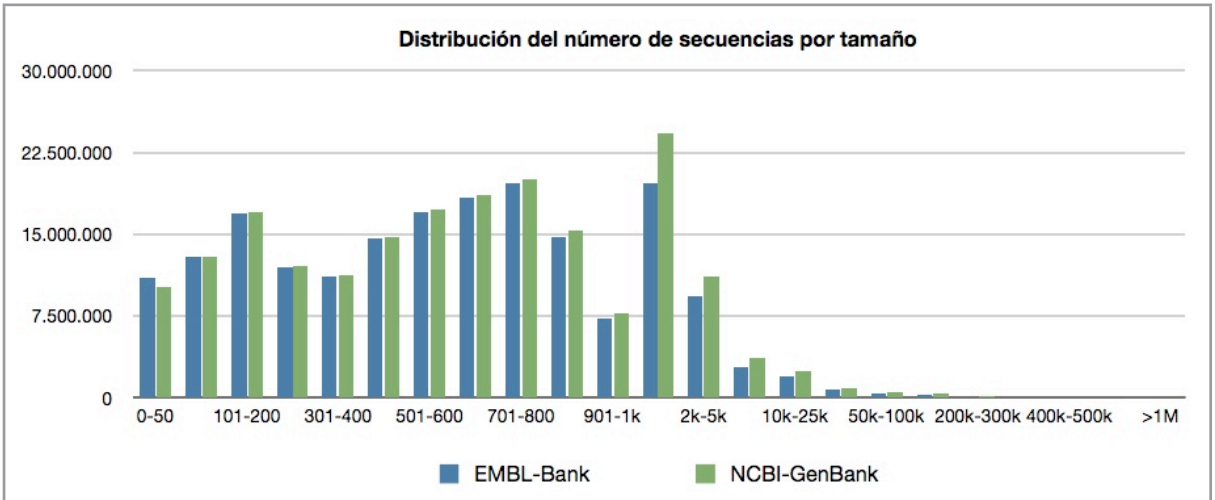


Figura 3.1. Histograma de secuencias por rango de longitudes.

Analizando los datos de la tabla 3.1, se pueden extraer algunas conclusiones interesantes. Así por ejemplo, en el caso de secuencias de ácidos nucleicos, se puede apreciar que el 91,9% de entradas del EMBL-Bank (el 90,5% en el caso del NCBI-GenBank) corresponden a secuencias de tamaño menor o igual a 2.000 bases (2 kb). En el caso de péptidos (UniProtKB), este porcentaje sube hasta el 99,67% del total de secuencias de péptidos almacenados, dejando tan sólo un 0,33% de secuencias de más de 2.000 residuos de aminoácidos. Esta disminución de tamaños de péptidos es lógica teniendo en cuenta dos factores: i) el código genético en el que se define la relación entre secuencias de tres nucleótidos (codógenos en el ADN genómico y codones en el ARN mensajero) y el correspondiente aminoácido que codifica en la proteína; y ii) sobre todo y con gran diferencia, la presencia de regiones no codificantes, como zonas reguladoras, intrones, ADN repetitivo y el llamado ADN basura en general (del ingles, "junk DNA"). De hecho, sorprendentemente, entre el 95% y el 99% de algunos genomas no tiene función conocida, aparte de la de autoperpetuarse, siendo considerado por algunos investigadores un parásito celular, hasta que se demuestre lo contrario. Dicho ADN representa una enorme carga energética para la célula, desafiando por ello las leyes básicas de ahorro energético y presión selectiva de la naturaleza. En definitiva, es uno de los misterios por resolver de la biología actual.

Resulta interesante analizar el número de bases de cada rango de tamaños de ácidos nucleicos. Esto se ilustra en la tabla 3.2 (y su correspondiente histograma mostrado en la figura 3.2), donde para cada fila, se ha multiplicado el número de secuencias por su tamaño (columna "total bases"), utilizando el valor medio del intervalo de los tamaños (columna "media del intervalo"). Así por ejemplo, en el caso de EMBL-Bank, para conocer cuántos nucleótidos hay en secuencias de tamaño entre 1 y 2 kb, se multiplica 1.500 (valor medio entre 1 y 2 kb) por 19.629.058 que es el total de secuencias existentes en dicha base de datos entre el rango indicado (1-2 kb), obteniéndose un total de 29.443.587.000 nucleótidos.

Tabla 3.2. Distribución porcentual de secuencias según tamaños de secuencias. Los datos fueron obtenidos a fecha de 17 de septiembre de 2010.

Rango de tamaños (residuos)	Número de entradas y porcentaje en base de datos										
	Datos a 17/09/2010										
	EMBL-Bank	%	media intervalo	total bases	% total	NCBI-Gen-Bank	%	total bases	% total	UniProtKB (proteínas)	%
0-50	11.041.818	5,79%	25	276.045.450	0,09%	10.213.240	5,09%	255.331.000	0,06%	403.557	3,32%
51-100	12.961.279	6,80%	75	965.615.286	0,31%	12.949.413	6,46%	964.731.269	0,22%	1.253.094	10,31%
101-200	16.898.000	8,87%	150	2.526.251.000	0,81%	17.023.040	8,49%	2.544.944.480	0,58%	2.832.858	23,31%
201-300	11.967.423	6,28%	250	2.985.872.039	0,96%	12.098.662	6,03%	3.018.616.169	0,68%	2.545.217	20,94%
301-400	11.065.229	5,81%	350	3.867.297.536	1,24%	11.185.895	5,58%	3.909.470.303	0,89%	1.922.726	15,82%
401-500	14.574.183	7,65%	450	6.551.095.259	2,10%	14.713.904	7,34%	6.613.899.848	1,50%	1.290.189	10,61%
501-600	17.016.290	8,93%	550	9.350.451.355	3,00%	17.256.763	8,60%	9.482.591.269	2,15%	674.950	5,55%
601-700	18.335.159	9,62%	650	11.908.685.771	3,82%	18.609.848	9,28%	12.087.096.276	2,74%	379.771	3,12%
701-800	19.645.861	10,31%	750	14.724.572.820	4,73%	20.014.114	9,98%	15.000.578.443	3,40%	267.917	2,20%
801-900	14.775.222	7,75%	850	12.551.551.089	4,03%	15.339.618	7,65%	13.031.005.491	2,95%	178.860	1,47%
901-1k	7.228.176	3,79%	950	6.863.153.112	2,20%	7.801.604	3,89%	7.407.622.998	1,68%	105.482	0,87%
1k-2k	19.629.058	10,30%	1.500	29.443.587.000	9,45%	24.320.861	12,13%	36.481.291.500	8,27%	260.378	2,14%
2k-5k	9.354.946	4,91%	3.500	32.742.311.000	10,51%	11.060.555	5,51%	38.711.942.500	8,77%	38.300	0,32%
5k-10k	2.799.244	1,47%	7.500	20.994.330.000	6,74%	3.584.731	1,79%	26.885.482.500	6,09%	2.083	0,02%
10k-25k	1.895.269	0,99%	17.500	33.167.207.500	10,65%	2.429.095	1,21%	42.509.162.500	9,63%	160	0,00%
25k-50k	694.048	0,36%	37.500	26.026.800.000	8,36%	917.813	0,46%	34.417.987.500	7,80%	11	0,00%
50k-100k	339.716	0,18%	75.000	25.478.700.000	8,18%	480.132	0,24%	36.009.900.000	8,16%	0	0,00%
100k-200k	264.409	0,14%	150.000	39.661.350.000	12,73%	345.576	0,17%	51.836.400.000	11,74%	0	0,00%
200k-300k	73.112	0,04%	250.000	18.278.000.000	5,87%	103.808	0,05%	25.952.000.000	5,88%	0	0,00%

Rango de tamaños (residuos)	Número de entradas y porcentaje en base de datos										
	Datos a 17/09/2010										
	EMBL-Bank	%	media intervalo	total bases	% total	NCBI-Gen-Bank	%	total bases	% total	UniProtKB (proteínas)	%
300k-400k	11.580	0,01%	350.000	4.053.000.000	1,30%	28.692	0,01%	10.042.200.000	2,28%	0	0,00%
400k-500k	4.239	0,00%	450.000	1.907.550.000	0,61%	15.272	0,01%	6.872.400.000	1,56%	0	0,00%
500k-1M	4.922	0,00%	750.000	3.691.500.000	1,19%	29.926	0,01%	22.444.500.000	5,09%	0	0,00%
>1M	3.443	0,00%	1.000.000	3.443.000.000	1,11%	34.889	0,02%	34.889.000.000	7,90%	0	0,00%
	190.582.626	**100,00%**		**311.457.926.214**		**200.557.451**	**100,00%**	**441.368.154.045**		**12.155.553**	**100,00%**

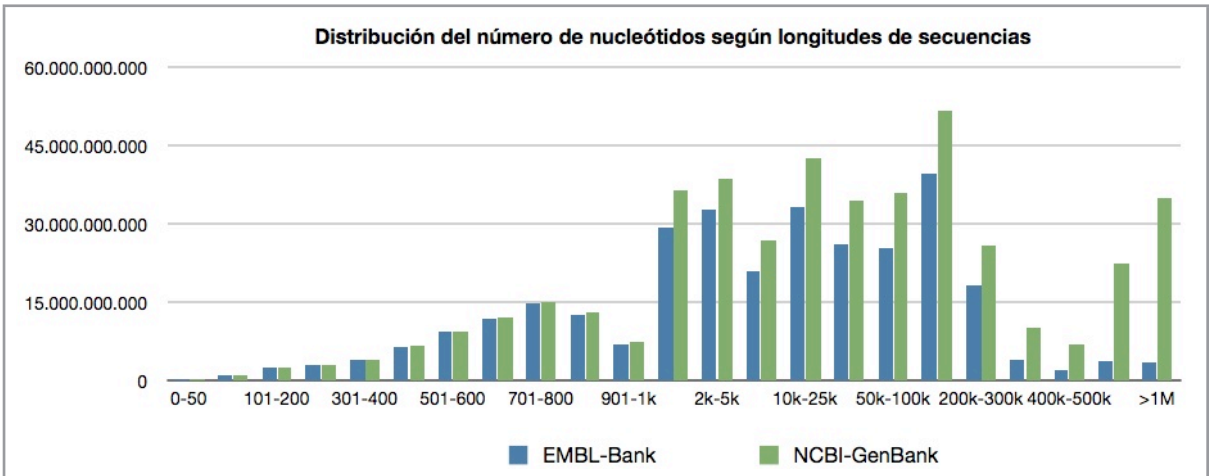


Figura 3.2. Histograma de secuencias por total de residuos de nucleótidos.

Se puede apreciar cómo en este nuevo análisis, las secuencias de tamaño menor o igual a 2 kb sólo suponen algo más del 32% del total de nucleótidos de la base de datos (datos para la base de datos EMBL-Bank).

Igualmente, se puede comprobar que el número de nucleótidos del total de secuencias entre tamaños 50 a 400 kb suponen casi un 30% del total de nucleótidos de la base de datos EMBL-Bank. Estas son, por tanto, las dimensiones aproximadas de las secuencias objetivo con las que se pretende aplicar el desarrollo del presente proyecto fin de Máster.

Breve reseña histórica

La secuenciación de ADN es un conjunto de métodos y técnicas bioquímicas cuya finalidad es la determinación del orden de los nucleótidos o –más específicamente– bases nitrogenadas de los mismos: adenina (A), citosina (C), guanina (G) y timina (T) en una secuencia de ADN. En el caso del ARN, la timina es sustituida por el uracilo (U). Hoy día los nuevos métodos de secuenciación de cadenas de ADN y ARN permiten secuenciar genomas enteros (miles de millones de bases) en horas. A título de ejemplo, el genoma humano se tardó en secuenciar unos 20 años, con un coste similar al proyecto Apolo para hacer llegar el hombre a la Luna: tres millardos (tres mil millones) de dólares (en inglés de Estados Unidos de América, “three billion dollars”). Hoy en día es posible hacerlo en una semana por unos 100.000 dólares y se espera que pueda realizarse en un día por unos 1.000 a 100 dólares en un periodo de tres a cinco años.

Así, en la actualidad, la pirosecuenciación o secuenciación 454, permite determinar la secuencia de ADN a gran escala, aplicable a genomas completos utilizando técnicas de luminiscencia. A diferencia del sistema Sanger, capaz de resolver 67.000 bases cada hora, el método 454 puede determinar la secuencia de 20 millones en 4,5 horas. Algunos modelos de secuenciadores del fabricante Roche, alcanzan los 400 millones de bases de alta calidad en 10 horas de ejecución.

En relación a futuros desarrollos, en el artículo "DNA tunneling detector embedded in a nanopore" [23] se describe la posible fabricación de un sistema de detección de residuos de nucleótidos haciendo pasar la secuencia de ADN por nanotubos con electrodos (ver figura 3.3) abriendo la puerta hacia la secuenciación ultrarrápida (hasta 10 millones de bases por segundo, en lugar de las 10 bases por segundo de algunas técnicas actuales) lo que permitiría acelerar y abaratar aún más el coste del proceso de adquisición de la secuencia, como se ha indicado anteriormente. De hecho, en teoría, este sistema permitiría secuenciar el genoma humano (tres mil millones de bases) en tan solo cinco horas y por unos cien mil dólares.

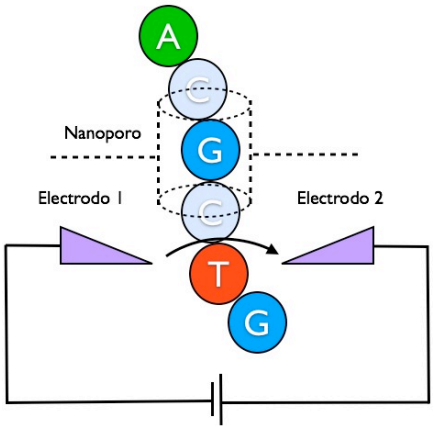


Figura 3.3. Reconocimiento de residuos de nucleótidos a través de nanoporos.

Este incremento en la velocidad de secuenciación (y el correspondiente abaratamiento de costes) junto con la posibilidad de contar con genomas completos, ha facilitado, entre otros, la aparición de una una gran cantidad de estudios de comparación de secuencias de gran tamaño, como la comparación de genes completos entre diferentes especies para determinar relaciones de parentesco o estudios de procedencia de ancestros comunes.

Así por ejemplo, Sonnhammer y Durbin [24] describieron un banco de trabajo con herramientas especialmente desarrolladas para el proyecto de secuenciación del genoma del gusano *Caenorhabditis elegans*, realizando alineamientos múltiples contra el cósmido *ZK637* cuya longitud supera los 40 kb.

Hardison et al [25] realizaron comparaciones de secuencias largas (entre 20 y 100 kb) de humano y ratón consiguiendo extraer conclusiones sobre nuevos elementos de regulación génica.

Chao et al [26] desarrollaron una herramienta, denominada *sim2*, para construir alineamientos locales de dos secuencias largas, aplicándola a un ejemplo de *Escherichia coli* de más de 130 kb de tamaño contra un grupo de 66 secuencias relacionadas en GenBank.

Delcher et al [27] presentaron el algoritmo MUMmer (se comentará brevemente en el apartado de fundamentos algorítmicos) y su aplicación en varios ejemplos biológicos como la comparación de secuencias de alrededor de 900 kb de dos estirpes de *Mycobacterium tubercolisis*, y de secuencias de 500 y 800 kb de dos especies menos similares de *Mycoplasma*. Por último, compararon dos secuencias aún más distantes, como son una subsecuencia de 227 kb del cromosoma humano 12p13 contra una subsecuencia de aproximadamente el mismo tamaño del cromosoma 6 de ratón.

De nuevo Hardison et al, esta vez en otro trabajo [28], analizaron el alineamiento de secuencias largas como excelentes guías de la historia evolutiva de las agrupaciones (del inglés, “cluster”) de genes de globinas beta de mamíferos. En su artículo mostraron un ejemplo de alineamiento de 70 kb.

Finalmente, es importante mencionar que no se ha pretendido realizar un estudio exhaustivo para la justificación biológica de la comparación de secuencias de gran tamaño, ya que ello queda fuera del ámbito de este proyecto fin de Máster. Tal y como mencionaron también Chao et al [26] en “A local alignment tool for very long DNA sequences”, realizar una evaluación sistemática del significado biológico de alineamientos de gran tamaño sería un trabajo arduo a la vez que interesante.

Fundamentos algorítmicos 4

Una secuencia de ácidos nucleicos o péptidos se puede modelar fácilmente como una cadena de símbolos a partir de un alfabeto dado. En el caso de ácidos nucleicos, el alfabeto es:

A_n = {A, C, T | U, G} (T o U, dependiendo de si se trata de ADN o ARN, respectivamente)

En el caso de los péptidos, el alfabeto está formado por las letras asociadas a cada uno de los 20 aminoácidos naturales que forman parte de los péptidos sintetizados en los ribosomas de las células, es decir:

$$A_p = \{A, R, N, D, C, Q, E, G, H, I, L, K, M, F, P, S, T, W, Y, V\}$$

Existen otros dos aminoácidos codificados por el código genético en algunas circunstancias y en algunos organismos: son la seleniocisteína (Sec, U) y la pirrolisina (Pyl, O). Sin embargo, no añaden ninguna restricción adicional de cara al planteamiento algorítmico para la resolución del problema de la comparación de secuencias.

De esta forma, dos secuencias de nucleótidos S_1 y S_2 podrían representarse como las siguientes cadenas de caracteres:

S_1 = ACCTGCGATGCTA

S_2 = ACGTGCAATGGTA

La comparación de cadenas de caracteres es un campo con entidad propia de las ciencias de la computación que tiene aplicaciones muy interesantes dentro del campo de la bioinformática. Evaluar las semejanzas y diferencias entre dos secuencias puede arrojar información importante sobre la relación funcional, estructural y evolutiva de ambas secuencias. Además, puede servir para desarrollar marcadores moleculares de aplicación práctica, como se ha indicado anteriormente. El grado de similitud es una medida muy utilizada para extraer conclusiones sobre si existe identidad u homología entre ambas secuencias.

Uno de los retos más importantes a la hora de comparar dos secuencias de ácidos nucleicos o péptidos es encontrar cuál es la subsecuencia común de mayor tamaño existente entre ambas. No debe confundirse el término subcadena con el de subsecuencia. Desde el punto de vista de las ciencias de la computación, una subcadena es una porción consecutiva de símbolos de la cadena, mientras que una subsecuencia no es necesariamente consecutiva. Dicho de otra forma, una subcadena de una cadena es siempre una subsecuencia, pero una subsecuencia de una cadena no es necesariamente una subcadena. El concepto de subsecuencia es crucial en los casos en los que se utilizan interrupciones o huecos para obtener un mejor alineamiento de las secuencias a comparar.

Ejemplo: sea la cadena S = ACGATCGTG

Una subcadena de *S* es “ATCG” (ACG**ATCG**TG)

Una subsecuencia de *S* es “ACGG” (ACG**A**T**CG**T**G**)

Las soluciones que han sido utilizadas más ampliamente para la resolución del problema de encontrar la subsecuencia (y subcadena) común de mayor tamaño entre dos cadenas de caracteres son la utilización de la programación dinámica y árbol de sufijos. También se usan los algoritmos codiciosos para el caso de secuencias muy similares en las que las diferencias se deben principalmente a errores de secuenciación.

Estas técnicas se describirán a lo largo del presente capítulo, pero antes se comentarán tanto las matrices de puntuación como el esquema de penalización por huecos durante el proceso de alineamiento.

Matrices de puntuación

Las matrices de sustitución o puntuación se utilizan en biología evolutiva para describir el ritmo al que un carácter en una secuencia de ácido nucleico o péptido cambia a otro carácter a lo largo del tiempo. Se utilizan ampliamente en el contexto del alineamiento de secuencias (tanto de residuos de nucleótidos como de residuos de aminoácidos), asignando pesos a las distintas combinaciones de cambios posibles. De este modo permiten determinar la bondad de un alineamiento entre dos (o más) secuencias, teniendo en cuenta la puntuación final obtenida tras el alineamiento.

Estas matrices están, por tanto, presentes en los distintos algoritmos que se comentarán durante este capítulo, como por ejemplo los basados en programación dinámica (tanto en alineamiento local como global) y son fundamentales pues indican, mediante un dato numérico concreto, cuál es la puntuación de cada emparejamiento de cada elemento de las dos secuencias a comparar.

Las matrices de sustitución más utilizadas en el caso de alineamiento de péptidos son las denominadas PAM y BLOSUM. Son matrices de probabilidades logarítmicas, en las que se reflejan la probabilidad de transformación de un residuo de aminoácido en otro.

En el caso de alineamiento de residuos de nucleótidos, que es el caso en el que se centra el presente proyecto, una de las matrices utilizada es la denominada EDNAFULL (también conocida como NUC4.4). Esta matriz puntúa con 5 la coincidencia de elementos, con –4 la no coincidencia, empleando distintas puntuaciones según los distintos códigos de ambigüedades de residuos de nucleótidos, tal y como se muestra en la figura 4.1.

```
#
# This matrix was created by Todd Lowe   12/10/92
#
# Uses ambiguous nucleotide codes, probabilities rounded to
#  nearest integer
#
# Lowest score = -4, Highest score = 5
#
   A   T   G   C   S   W   R   Y   K   M   B   V   H   D   N
A  5  -4  -4  -4  -4   1   1  -4  -4   1  -4  -1  -1  -1  -2
T -4   5  -4  -4  -4   1  -4   1   1  -4  -1  -4  -1  -1  -2
G -4  -4   5  -4   1  -4   1  -4   1  -4  -1  -1  -4  -1  -2
C -4  -4  -4   5   1  -4  -4   1  -4   1  -1  -1  -1  -4  -2
S -4  -4   1   1  -1  -4  -2  -2  -2  -2  -1  -1  -3  -3  -1
W  1   1  -4  -4  -4  -1  -2  -2  -2  -2  -3  -3  -1  -1  -1
R  1  -4   1  -4  -2  -2  -1  -4  -2  -2  -3  -1  -3  -1  -1
Y -4   1  -4   1  -2  -2  -4  -1  -2  -2  -1  -3  -1  -3  -1
K -4   1   1  -4  -2  -2  -2  -2  -1  -4  -1  -3  -3  -1  -1
M  1  -4  -4   1  -2  -2  -2  -2  -4  -1  -3  -1  -1  -3  -1
B -4  -1  -1  -1  -1  -3  -3  -1  -1  -3  -1  -2  -2  -2  -1
V -1  -4  -1  -1  -1  -3  -1  -3  -3  -1  -2  -1  -2  -2  -1
H -1  -1  -4  -1  -3  -1  -3  -1  -3  -1  -2  -2  -1  -2  -1
D -1  -1  -1  -4  -3  -1  -1  -3  -1  -3  -2  -2  -2  -1  -1
N -2  -2  -2  -2  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1
```

Figura 4.1. Matriz de pesos NUC.4.4 o EDNAFULL.

En esta figura se emplean los siguientes códigos de nomenclatura de nucleótidos definido por la International Union of Pure and Applied Chemistry (IUPAC). Estos códigos y su equivalencia en bases nitrogenadas se muestran en la tabla 4.1:

Tabla 4.1. Códigos de nucleótidos según IUPAC.

Código IUPAC	Base nitrogenada
A	Adenina
C	Citosina
G	Guanina
T (o U)	Timina (o Uracilo)
R	A or G
Y	C or T
S	G or C
W	A or T
K	G or T
M	A or C
B	C or G or T
D	A or G or T
H	A or C or T
V	A or C or G
N	Any base
. o -	Gap

La matriz EDNAFULL fue creada por Todd Lowe en 1992 y aunque nunca se publicaron sus datos, se aplicó para el trabajo comentado en el artículo “dbEST – database for expressed sequence tags” [29]. La matriz se construye teniendo en cuenta las probabilidades de mutación de cada residuo y cada elemento de la matriz representa el entero más cercano a la puntuación de probabilidad.

La matriz EDNAFULL se puede descargar del “National Center for Biotechnology Information” (NCBI): <ftp://ftp.ncbi.nih.gov/blast/matrices>.

Este directorio (carpeta) del servidor FTP del NCBI contiene todas las matrices que utilizan la implementación del algoritmo BLAST del NCBI. La matriz EDNAFULL también es utilizada ampliamente en el conjunto de aplicaciones EMBOSS (del inglés, "European Molecular Biology Open Source Suite"). Estos han sido algunos motivos por los que también se ha tenido en cuenta en el presente proyecto. Otras matrices utilizadas han sido MATCH e IDENTITY, también descargadas del NCBI y utilizadas por BLAST.

Penalización por huecos y huecos afines

Es importante tener en cuenta que las secuencias de ácidos nucleicos y péptidos pueden cambiar, ganar o perder residuos a lo largo de la evolución.

Las matrices de comparación (de peso o sustitución) están basadas en mutaciones que conducen a la sustitución de un residuo por otro (cambio de base o cambio de aminoácido). Por otro lado, es sabido que existen mutaciones que consisten en inserciones o deleciones denominados "indels" (contracción del inglés "insertions & deletions") de un residuo o varios. La introducción de espacios en un alineamiento puede servir para optimizarlo, además de poder reflejar de forma más realista la evolución de la secuencia en cuestión.

Uno de los métodos más simples de penalizar un alineamiento por introducción de huecos es un modelo lineal, en el que la penalización crece linealmente con el tamaño del hueco. Así, si la longitud del hueco es *x*, la función de penalización sería el de la figura 4.2, donde *d* es una constante de penalización por hueco:

$$\gamma(x) = x * d$$

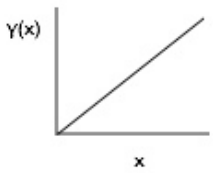


Figura 4.2. Esquema de penalización lineal.

Sin embargo, normalmente los huecos pueden aparecer también en grupos, debido a mecanismos de error que producen inserciones de uno o varios residuos en la secuencia. Un esquema de penalización más realista consiste en penalizar más la apertura y menos la extensión o prolongación de huecos existentes mediante una función convexa, como se muestran en la curva de color rojo de la figura 4.3.

El inconveniente de esta aproximación es que el cálculo del alineamiento bajo este modelo incrementa su complejidad tanto en espacio como en tiempo, lo que lo hace poco práctico para secuencias largas. Por este motivo, se suele utilizar ampliamente una

aproximación lineal de la función convexa que es conocida con el nombre de hueco afín (del inglés, “affine gap”).

$$\gamma(x) = e + (x - 1) * d \text{ para } x \geq 1$$

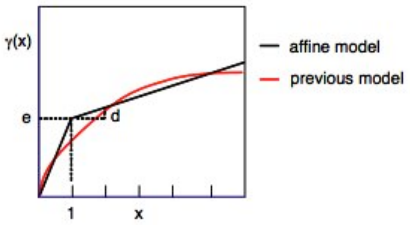


Figura 4.3. Esquema de penalización de huecos afines.

donde:

e = penalización por apertura de hueco

x = número total de huecos

d = penalización por extensión del hueco

Se podría pensar en una generalización de la función de hueco afín utilizando una función lineal por partes que aproxime mejor a la curva de la función convexa anteriormente mencionada. Sin embargo, esta aproximación añade complejidad algorítmica, al tener que manejar varias matrices adicionales, como se verá más adelante cuando se hable del método de programación dinámica.

Algunos estudios de la distribución de tamaños de *indels* han encontrado cierta consistencia en que obedecen a una ley de potencia. Esto ha hecho que algunos investigadores propongan modelos logarítmicos de penalización por huecos cómo métodos más realistas, biológicamente hablando, que el propuesto por el modelo de hueco afín. Sin embargo, Cartwright [30] demostró que, en contraste con estas expectativas, los alineamientos basados en costes logarítmicos producen alineamientos peores que los más precisos producidos por el esquema de hueco afín, que además siguen siendo aproximaciones más realistas biológicamente y muy manejables, algorítmicamente hablando.

Programación dinámica

El término programación dinámica (del inglés, “dynamic programming”) fue introducido originalmente por Richard Bellman [31] en los años 40 para describir el proceso de solucionar problemas complejos mediante su descomposición en problemas más sencillos. Este proceso es aplicable especialmente en aquellos problemas que tienen las siguientes propiedades:

1. Se pueden descomponer en subproblemas que se solapan y que son ligeramente más pequeños, de forma recursiva hasta llegar al caso fácil, cuya resolución es trivial.

2. Tienen subestructura óptima, es decir, la solución óptima se puede construir eficientemente a partir de las soluciones óptimas de sus subproblemas.

Es una aproximación similar al principio "divide y vencerás" utilizado en muchos algoritmos computacionales. El problema original se descompone en subproblemas. Los subproblemas se resuelven de forma recursiva y las soluciones obtenidas se combinan para conseguir la solución al problema original. Cuando los subproblemas no son independientes entre sí es cuando el método de programación dinámica es aplicable.

El desarrollo de un algoritmo de programación dinámica se puede descomponer en los siguientes cuatro pasos [32, 33]:

1. Caracterizar la estructura de una solución óptima.
2. Definir de forma recursiva el grado de una solución óptima.
3. Obtener el grado de una solución óptima de abajo a arriba (del inglés, "bottom-up"),
4. Construir una solución óptima a partir de la información calculada.

Alineamiento global y local

En 1970, Needleman y Wunsch [34] propusieron un algoritmo para establecer el grado de similitud entre dos secuencias genéticas determinando su alineación de forma global utilizando programación dinámica. Los algoritmos de similitud global como éste optimizan el alineamiento global de dos secuencias, que incluso pueden incluir grandes extensiones de menor similitud.

Años más tarde, en 1981, Smith y Waterman [5] introdujeron una pequeña modificación (pero de gran impacto) para permitir alinear pares de subsecuencias, obteniendo distintas alineaciones locales. En este caso, se busca sólo subsecuencias relativamente conservadas de forma que una comparación simple de dos secuencias puede producir varios alineamientos distintos de subsecuencias. Las regiones no conservadas de las secuencias comparadas no contribuyen de esta forma a la medida de similitud.

El algoritmo de Smith y Waterman permite determinar, de todos los alineamientos distintos posibles de subsecuencias, cuál es el que produce una alineación óptima, de forma que no haya otro par de subsecuencias con mayor similitud (identidad).

Una motivación para el alineamiento local es, por tanto, la dificultad para obtener alineamientos correctos en regiones de baja similitud entre secuencias biológicas distantes, debido fundamentalmente a que las mutaciones producidas introducen mucho "ruido" a lo largo del tiempo.

Por su importancia y dado que es el algoritmo en el que se centra el presente trabajo, se describe brevemente a continuación. Dicho algoritmo consta fundamentalmente de tres pasos:

1. Construir una matriz de puntuación entre la secuencia A y B.
2. Obtener la puntuación máxima de la matriz.
3. Recorrer desde ese punto donde se encuentra la puntuación máxima y hacia atrás el camino que produce la alineación óptima entre las dos secuencias.

Dadas dos secuencias *A* y *B* de longitudes *n* y *m* respectivamente, el algoritmo construye una matriz *H* de dimensiones (*n*+1) x (*m*+1), donde la primera fila y la primera columna se inicializan a 0.

$$H_{k0} = H_{0l} = 0 \text{ para } 0 \leq k \leq n \text{ y } 0 \leq l \leq m$$

El resto de elementos $H_{i,j}$ de la matriz se van calculando mediante la siguiente función:

$H_{i,j}$ = max : 0 (es la diferencia básica con Needleman-Wunsch)

$H_{i-1,j}$ – hueco en caso de borrado

$H_{i,j-1}$ – hueco en caso de inserción

$H_{i-1,j-1} + s(A_i, B_j)$ si se contempla emparejamiento entre A_i y B_j (no necesariamente deben coincidir, se tiene en cuenta el peso de la matriz de similitud)

Donde $1 \leq i \leq n$, $1 \leq j \leq m$ y $s(A_i, B_j)$ es la función de similitud entre dos elementos del alfabeto en cuestión (residuos de nucleótidos o de aminoácidos) y el hueco es la penalización que se introduce cuando dos elementos no coinciden (por iniciar o continuar una discontinuidad).

El 0 en la función *max* se incluye para prevenir el cálculo de similitudes negativas, indicando la no similitud entre A_i y B_j. Esta es la diferencia básica con respecto al algoritmo propuesto por Needleman-Wunsch.

El par de segmentos con mayor similitud se encuentra a partir de localizar el elemento de mayor puntuación de la matriz *H*. A partir de este elemento y mediante un procedimiento de vuelta atrás, se obtiene el mejor alineamiento hasta llegar a un elemento con valor igual a 0. Por tanto, este procedimiento permite identificar los segmentos pero también produce el correspondiente alineamiento, introduciendo los huecos necesarios.

La gran ventaja de este algoritmo, además de la seriedad matemática que aporta para el cálculo de los segmentos de mayor similitud, es que se puede trasladar fácilmente a un programa de ordenador. Sin embargo, este algoritmo tiene complejidad cuadrática O ($n \times m$) tanto en tiempo como en espacio. El tamaño de la matriz completa (del inglés, "Full Matrix"*; FM*) de programación dinámica es $n \times m$.

Esta complejidad tanto en espacio como en tiempo ha provocado la generación de otros algoritmos más rápidos (aunque menos precisos y meticulosos) para el cálculo de la puntuación. Tal es el caso del BLAST [4], que en una primera búsqueda rápida intenta encontrar similitudes de subcadenas con un tamaño determinado para en una segunda fase, aplicar un algoritmo más costoso computacionalmente como puede ser Smith-Waterman.

El BLAST es muy útil y eficiente, siendo actualmente la herramienta más utilizada en el alineamiento de secuencias por pares que, aunque no garantiza encontrar el alineamiento óptimo, sí aproxima de forma significativa los resultados, consiguiendo un importante ahorro de tiempo utilizando algoritmos heurísticos. Esto es particularmente relevante cuando se trata de encontrar identidades entre una secuencia dada y la ingente cantidad de secuencias existentes en las bases de datos actuales.

Introducción de huecos

Con la inclusión de huecos se pretende encontrar el mínimo número de mutaciones necesarias para convertir una secuencia en otra. El algoritmo propuesto por Smith y Waterman permite la inserción/borrado de huecos de longitud arbitraria con coste lineal. Sin embargo, ya se ha visto anteriormente que una mejor aproximación a esta "evolución" biológica es posible con esquema de huecos afines.

Gotoh [35] introdujo una variante en el algoritmo de Smith-Waterman para incluir la aproximación de huecos afines, sin penalizar excesivamente al algoritmo y manteniendo su coste cuadrático $n \times m$. Esta es la función de penalización por huecos que más se suele implementar en sistemas computacionales que adoptan el algoritmo Smith-Waterman.

En esta nueva variante, es necesario distinguir especialmente dos estados: la apertura de nuevo hueco o la continuación de un hueco previamente abierto. Para ello, se hace necesario introducir nuevas matrices de programación dinámica en el algoritmo que de forma general quedaría como sigue:

Inicialización: $V(i, 0) = d + (i - i) * e$

$V(0, j) = d + (j - 1) * e$

Iteración: $V(i, j) = \max \{ F(i, j), G(i, j), H(i, j) \}$

$F(i, j) = V(i - 1, j - 1) + s(x_i, y_j)$

$G(i, j) = \max \{ V(i - 1, j) - d , G(i - 1, j) - e \}$

$H(i, j) \max \{ V(i, j - 1) - d , H(i, j - 1) - e \}$

Finalización: V(i, j) contiene el mejor alineamiento

Donde:

F(i, j) mantiene la puntuación del alineamiento cuando x_i se alinea con y_j.

G(i, j) puntuación cuando x_i se alinea con un hueco después de y_j.

H(i, j) puntuación cuando y_j se alinea con un hueco después de x_i.

V(i, j) máximo de las 3 anteriores; es decir, la mejor puntuación del alineamiento.

Caso 1: apertura de un nuevo hueco (x_i se alinea con y_j).

$x_1 \ldots x_{i-1} \mid x_i \mid x_{i+1}$

$y_1 \ldots y_{j-1} \mid y_j \mid -$ $\quad G(i + 1, j) = V(i, j) - d$

Caso 2: continuación de un hueco previo (x_i se alinea con un hueco).

$x_1 \ldots x_{i-1} \mid x_i \mid x_{i+1}$

$y_1 \ldots y_{j-1} \mid - \mid -$ $\quad G(i + 1, j) = G(i, j) - e$

Algoritmo FastLSA

Se acaba de ver que para dos secuencias de ácidos nucleicos o péptidos de longitudes *m* y *n*, respectivamente, los algoritmos de alineamiento basados en programación dinámica como Needleman-Wunsch y Smith-Waterman tienen una complejidad O ($m \times n$) en tiempo y espacio, por lo que se suelen llamar algoritmos de matriz completa, como se ha indicado anteriormente. Este requisito de espacio significa que para secuencias de longitudes muy largas, las estructuras de datos pueden no caber en la memoria principal del ordenador. Incluso para secuencias algo más cortas, puede que tampoco entren en memoria caché. Así, por ejemplo, para dos secuencias pequeñas de 10.000 residuos de nucleótidos cada una, se requiere una cantidad prohibitiva de almacenamiento (100.000.000 entradas).

Hirschberg [36] introdujo con su algoritmo una mejora en los requerimientos de espacio reduciéndolo a un espacio lineal de O [min(*m*, *n*)] pero requiriendo aproximadamente el doble de operaciones que con los algoritmos de *FM*. Este autor fue el primero en ver que el cálculo

del primer paso del algoritmo de programación dinámica (el cálculo de la matriz de puntuaciones del alineamiento) se puede realizar usando espacio lineal, almacenando únicamente la última fila calculada.

Charter et al [37] introdujeron en el año 2000 el algoritmo de alineamiento de espacio lineal rápido (del inglés, "Fast Linear-Space Alignment"; FastLSA) que se adapta a la cantidad de memoria disponible. El algoritmo permite al usuario intercambiar espacio de almacenamiento por operaciones consiguiendo un comportamiento lineal en el mejor de los casos, y cuadrático en el peor, dependiendo de las características del sistema computacional con el que se cuente. Menos espacio significa que algunos valores se tienen que recalcular en algún momento. En la práctica, el algoritmo FastLSA suele ser más rápido que los algoritmos basados en FM.

La base del algoritmo FastLSA consiste en subdividir el problema de forma recursiva en un número de piezas o partes $k \geq 2$, usando almacenamiento que está limitado por $2 \times k \times n$, y permitiendo utilizar la memoria adicional disponible para reducir el tiempo de ejecución. Cuando $k = m$, el algoritmo es equivalente a los algoritmos FM. Se puede ver el algoritmo FastLSA como generalización del algoritmo de Hirschberg en dos aspectos:

1. El mejor alineamiento se aproxima a un camino diagonal de la matriz por lo que subdividir el problema vertical y horizontalmente tiene sentido para este tipo de aplicaciones (en lugar de una dimensión, como ocurre con Hirschberg).
2. Dado que todos los elementos de la matriz deben ser visitados al menos una vez, partir los datos recursivamente en k ≥ 2 piezas permite utilizar el almacenamiento adicional para reducir recálculos.

Dadas dos cadenas de longitud m y n ($m \leq n$) y R unidades de memoria, se trata de encontrar la forma más eficiente en coste de hacer el alineamiento de ambas cadenas. Si R es mayor que $m \times n$, se podría usar un algoritmo de matriz completa o *FM,* puesto que cabría completamente en memoria. Si éste no es el caso, se tiene que usar una variante que tenga en cuenta esta limitación de memoria. El algoritmo propuesto por Hirschberg funcionaría y usaría $2 \times m$ bloques de memoria, pero no tendría en cuenta el hecho de que hubiese más memoria disponible. Hirschberg divide recursivamente el problema a la mitad, almacenando la información de cada fila.

Sin embargo, si el problema se divide a nivel de fila y columna y se guardan los extremos de cada una de ellas, se podría encontrar el alineamiento recalculando pocos valores. Esto se puede apreciar en la figura 4.4, donde la zona sombreada corresponde con los valores que se tienen que recalcular. Se puede apreciar como en el caso del algoritmo de Hirschberg (panel a) se tienen que recalcular más valores que en el caso propuesto por FastLSA (panel b), en el que se ha utilizado un valor de $k = 5$.

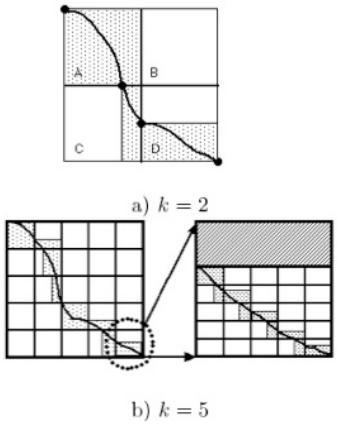


Figura 4.4. Comparación entre los algoritmos Hirschberg y FastLSA .

El algoritmo FastLSA fue integrado en herramientas comerciales como GeneTool de la empresa BioTools.

En 2003, Driga et al [38] realizaron una versión paralela de *FastLSA* usando un método simple pero efectivo de paralelismo de frente de ola (del inglés, “wavefront parallelism”). El resultado de los experimentos realizados mostraron que la versión paralela de *FastLSA* consigue mejorar los tiempos casi de forma lineal para un sistema dotado de ocho procesadores. También concluyeron que la eficiencia de esta implementación paralela aumenta con el tamaño de las secuencias a alinear.

Algoritmos codiciosos

Cuando las dos secuencias a alinear sólo se diferencian por errores de secuenciación (no producidos por procesos evolutivos, siendo las secuencias muy similares), se pueden aplicar otro tipo de algoritmos como los algoritmos codiciosos (del inglés, “greedy algorithms”) que son más rápidos que los algoritmos de programación dinámica tradicionales.

Efectivamente, Zhang et al [6], introdujeron en su artículo un nuevo algoritmo de alineamiento con un rendimiento particularmente bueno y demostraron que produce el mismo alineamiento que los métodos basados en programación dinámica, llegando a ejecutarse hasta unas 10 veces más rápidamente (según los datos usados). Existe actualmente una implementación de este algoritmo que se utiliza en una aplicación que reúne o ensambla la base de datos UniGene del NCBI. Para el estudio, han supuesto una tasa de error del 3% en la secuenciación.

También conviene destacar el trabajo de Chao et al [39], que desarrollaron una aplicación informática a la que denominaron *sim3*, que resuelven el siguiente problema computacional: dadas dos secuencias de ADN donde una de ellas es más corta que la otra y muy similar a una región contigua de la secuencia más larga, *sim3* determina cuál es esa región

dentro de la secuencia larga y calcula el conjunto óptimo de cambios de residuos de nucleótido simple necesario (inserciones, deleciones o sustituciones) para convertir la secuencia corta en esa región similar.

En estos casos, el esquema de puntuación para el alineamiento se diseña de forma que se consideren los errores en la secuenciación y no los cambios producidos por procesos evolutivos. Este es el motivo por el que, en estos casos, no se utiliza un esquema de huecos afines, consiguiendo por tanto una simplificación del algoritmo que permite agilizar su velocidad de ejecución. El algoritmo *sim3* puede alinear una secuencia de 100 kb contra otra de un millón de bases en sólo varios segundos en una estación de trabajo (del inglés, "workstation").

Siendo m y n las longitudes de las dos secuencias, E el número mínimo de operaciones (la distancia entre ambas secuencias) y suponiendo que E es muy pequeño comparado con m y n, *sim3* se ejecuta en el peor de los casos en un tiempo de $O(\min(m, n) \times E)$ y en espacio de $O(m + n)$.

Árbol de sufijos

El árbol de sufijos (del inglés, "suffix-tree") no es realmente un método de alineamiento de secuencias, sino una forma de representar una secuencia.

Se trata de una estructura de datos en la que se representan los sufijos de una cadena dada, de forma que facilite realizar, de forma muy rápida, algunas de las operaciones más frecuentes con cadenas, como puede ser la operación de comparación.

El árbol de sufijos de una cadena S de longitud m se define como aquel árbol que:

1. Los caminos desde la raíz del árbol hasta las hojas tienen una relación uno a uno con los sufijos de S.
2. Las ramas no tienen cadenas vacías.
3. Todos los nodos internos del árbol (excepto el raíz) tienen al menos dos hijos.

Como ejemplo, la figura 4.5 muestra el árbol de sufijos para la cadena BANANA. Cada subcadena termina con el carácter especial $. Los seis caminos posibles desde el nodo raíz se corresponden con cada uno de los seis sufijos posibles.

BANANA$

ANANA$

NANA$

ANA$

NA$

A$

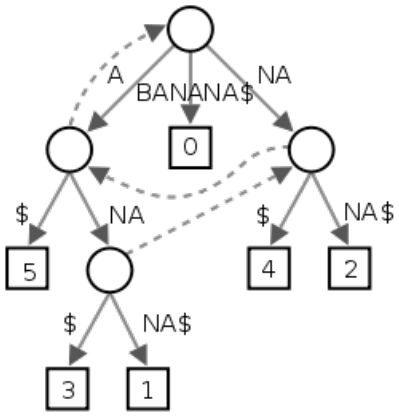


Figura 4.5. Árbol de sufijos para la cadena BANANA.

El concepto de árbol de sufijos fue introducido por Weiner en 1973 y le supuso el reconocimiento de “algoritmo del año” por Donald Knuth en 1973. La propuesta inicial de Weiner fue posteriormente simplificada y mejorada por Ukkonen [40] quien propuso un algoritmo lineal, O (*n*), en tiempo para la construcción del árbol.

Cualquier cadena de longitud *m* se puede descomponer en *m* sufijos que se pueden almacenar en un árbol de sufijos. Crear esta estructura de este tipo para una cadena de longitud *m* requiere un tiempo de orden O (*m*). La búsqueda de un patrón de longitud *n* en esta cadena requiere O (*n*).

La gran ventaja de los árboles de sufijos es que pueden servir para encontrar patrones en secuencias de ácidos nucleicos y péptidos. Esto hace que este tipo de estructuras sean útiles para una amplia gama de aplicaciones de la bioinformática como el alineamiento de múltiples genomas o la identificación de repeticiones de secuencias.

Hay varios algoritmos y aplicaciones basados en el concepto de árboles de sufijos, como son MUMmer [27] y el alineador de genomas múltiples (del inglés, “Multiple Genome Aligner”; MGA) [41]. Aunque ambos utilizan la misma estructura de datos basada en árboles de sufijos, emplean aproximaciones diferentes para hacer el alineamiento. Así, MUMmer encuentra el máximo de parejas únicas (del inglés, “Maximal Unique Matches”; MUM) que son las secuencias que sólo ocurren una única vez en ambas secuencias a comparar. Luego las ordena para encontrar el conjunto de MUM más largo y en el mismo orden en ambas secuencias y utiliza estos conjuntos como punto de anclaje para hacer el alineamiento. Los huecos entre dos MUM se rellenan utilizando el alineamiento de Needleman y Wunsch.

El algoritmo MUMmer se utiliza normalmente para comparar dos genomas, mientras que MGA calcula la secuencia más larga sin solapamiento del máximo de parejas múltiples exactas (del inglés, “Maximal Multiple Exact Matches”; multiMEMs) utilizándolo como punto de partida para el alineamiento múltiple.

Arquitectura hardware y software 5

Actualmente la ingeniería informática evoluciona y se desarrolla a gran velocidad, con sistemas hardware que avanzan hacia nuevas arquitecturas multiprocesador y con soporte de software hacia nuevos modelos de programación así como nuevos esquemas de abstracción, con el objeto de obtener la máxima eficiencia del hardware subyacente a un bajo coste.

Conviene distinguir entre procesadores multinúcleo (del inglés, “multi-core”) que tienen pocos núcleos, de los procesadores de muchos núcleos (del inglés, “many-core”) que tienen decenas de núcleos, como el usado en este trabajo (64 núcleos). Aunque estas tecnologías ofrecen una oportunidad para cambiar las reglas del juego en la mejora del rendimiento del procesamiento y en la eficiencia energética, también traen consigo grandes retos en el nuevo diseño y la programación. Conforme la industria de los microprocesadores avanza, las tres P de eficiencia energética, rendimiento y programabilidad (del inglés, “Power efficiency, Performance and Programmability”, respectivamente) son la vara de medir por la que se juzgan a las distintas arquitecturas.

La demanda de aplicaciones que requieren gran capacidad de procesamiento está presente en prácticamente todos los ámbitos, desde el mercado de sistemas embebidos hasta el ordenador personal de escritorio. Esta demanda sigue aumentando a un ritmo acelerado. Así, por ejemplo, los sistemas de vídeo modernos requieren de 10 a 100 veces más potencia de cómputo que el de hace unos años debido al aumento de resolución (de la definición estándar a alta definición), algoritmos de compresión más sofisticadas (MPEG2 a H.264), y un mayor número de canales.

Otro ejemplo claro de esta gran demanda de procesamiento está el campo de la bioinformática, donde el incremento de la capacidad de los modernos secuenciadores de ácidos nucleicos hacen que la cantidad de información almacenada en las bases de datos de genómica o proteómica crezcan a un ritmo exponencial, demandando nuevos sistemas y arquitecturas más potentes para la búsqueda y análisis de patrones para los distintos estudios e investigaciones.

Esta amplia gama de dominios de aplicación que usan las tecnologías de muchos núcleos, viene acompañada de una gran variedad de definiciones e implementaciones de este concepto. De hecho, existen tantas de ellas que se podría concluir que muchos núcleos se refiere a un solo chip que contiene muchos motores de procesamiento visiblemente distintos, cada uno con control independiente. En cierto sentido, esto puede ser visto como el estilo de múltiples instrucciones con múltiples datos (del inglés, “Multiple-Instruction-Multiple-Data”; MIMD). Pero incluso tomando esta definición simplificada, existen numerosas implementaciones diferentes de muchos núcleos.

Ley de Moore y brecha tecnológica

La ley de Moore describe una tendencia a largo plazo en la historia del hardware de ordenadores. El número de transistores que se pueden colocar a bajo coste en un circuito integrado se duplica aproximadamente cada dos años [42].

En los últimos años, ha quedado patente que el modelo de procesamiento tiene que evolucionar si no quiere incrementar la brecha desde el punto de vista del rendimiento. De hecho, el rendimiento conseguido ha seguido la Ley de Moore hasta el año 2002 aproximadamente, debido a técnicas como el encauzamiento, canalización o segmentación (del inglés, "pipelining"), diseños de procesamiento paralelo (del inglés, "superscalar") y con muchos hilos (del inglés, "multithreading") [43]. Sin embargo, a partir del 2002 la escala de rendimiento se vino abajo y estos tres elementos han sido insuficientes para mantener el ritmo de crecimiento en el procesamiento (figura 5.1).

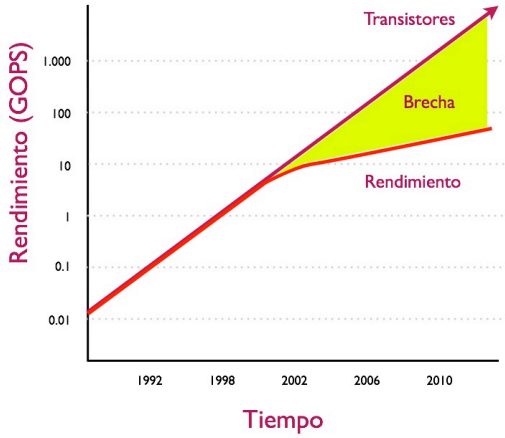


Figura 5.1. Gráfica de la brecha tecnológica según la ley de Moore.

Hay dos factores que podrían ayudar a minimizar la brecha tecnológica que puede causar la Ley de Moore. En primer lugar está que las aplicaciones actuales y las cargas de trabajo de hoy en día tienden al uso extendido del paralelismo. Ya se han visto algunos ejemplos como el procesamiento de vídeo, o incluso en el caso del campo de la bioinformática, existiendo muchos otros ejemplos más, como es el caso de la retransmisión de paquetes del protocolo de Internet (del inglés, "IP forwarding"), las redes inalámbricas (descodificación y filtrado), seguridad en dispositivos cortafuegos (del inglés, "firewalls") basados en estándares de cifrado avanzado (del inglés, "Advanced Encryption Standar"; AES), e incluso en automoción (control de motor, multitud de sensores).

El segundo factor está en la capacidad hoy día de poder integrar varios núcleos de procesamiento dentro de un mismo chip. Además, en términos de ahorro de energía, este nivel

de integración es mucho más eficiente que el uso de componentes discretos en una placa base.

El rendimiento y la eficiencia energética son las principales ventajas que aportan los sistemas de muchos núcleos frente a los sistemas de microprocesador simple. Sin embargo, no es trivial mejorar estos dos factores conforme el número de núcleos crece a decenas, centenas, e incluso miles de núcleos. Uno de los principales retos a la hora de trabajar con un número elevado de núcleos (aparte de la eficiencia del proceso de fabricación litográfica del propio microprocesador y de la generación y disipación de calor generado por el funcionamiento del chip) está en la red que interconecta estos núcleos entre sí y con la memoria principal.

Los sistemas de muchos núcleos actuales se basan en las vías de comunicación del chip (del inglés, "bus") o anillos para su interconexión. El aumento (del inglés, "escalate") de estas vías de comunicación puede ser problemático, constituyendo un auténtico cuello de botella. Sólo un núcleo a la vez puede utilizar el bus en un momento dado y la función de control del bus suele ser centralizada. Otra topología típica en los sistema de muchos núcleos actuales es la malla (matriz "bidimensional" de núcleos), que suele ser la forma más efectiva a la hora de interconectar núcleos cuando el número de éstos crece.

Arquitectura del microprocesador Tile64

A continuación se describe la arquitectura del microprocesador utilizada en la implementación paralela del algoritmo Smith-Waterman.

Tilera es una compañía estadounidense de semiconductores líder del sector en el diseño de microprocesadores de muchos núcleos cuyo número puede incrementarse con relativa facilidad. La compañía fue fundada en octubre de 2004, lanzando al mercado su primer producto en agosto del 2007: el microprocesador Tile64 de 64 núcleos (figura 5.2).

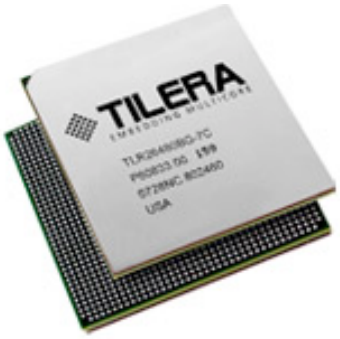


Figura 5.2. Chip Tile64. Foto extraída de la documentación del fabricante.

La arquitectura de microprocesador Tile64 supone un paso significativo en la búsqueda, durante décadas, del aprovechamiento de la potencia de múltiples microprocesadores para conseguir el rendimiento esperado en la ejecución de aplicaciones de computación paralela intensiva. Esta arquitectura ha evolucionado hasta convertirse en el primer diseño del mercado en incorporar un gran número de microprocesadores de propósito general en un único chip.

Características principales

Tile64 es un microprocesador RISC producido mediante un proceso de fabricación litográfica de 90 nm, que contiene 64 elementos o núcleos de propósito general (denominados tejas), que se comunican entre sí por medio de una red de interconexión denominada iMesh, con un ancho de banda de 31 Tbps, como se ha indicado anteriormente.

Cada teja o núcleo es capaz de ejecutar un sistema operativo Linux propio, con una frecuencia de reloj de 500-866 MHz y un uso eficiente de energía (15-22 W a 700 MHz, con todos los núcleos activos), consiguiendo alcanzar un total de 0,166 tera instrucciones por segundo (del inglés, “Tera-Instructions Per Secod”; TIPS), con operaciones de 32 bits gracias a su “pipeline” de paralelismo a nivel de instrucción. Cada teja contiene 8 KB de “caché” L1 de instrucciones y datos accesible por un amortiguador lateral de traducción (del inglés, “Translation Look-aside Buffer”; TLB) y una memoria caché L2 de 64 KB.

Adicionalmente, las cachés L2 de todos los núcleos se agrupan formando una caché L3 de 4 MB para el microprocesador completo. Es necesario dedicar una teja para gestionar estas capacidades, reduciendo el total de tejas efectivas para la ejecución de las aplicaciones.

La figura 5.3 muestra la arquitectura del microprocesador Tile64:

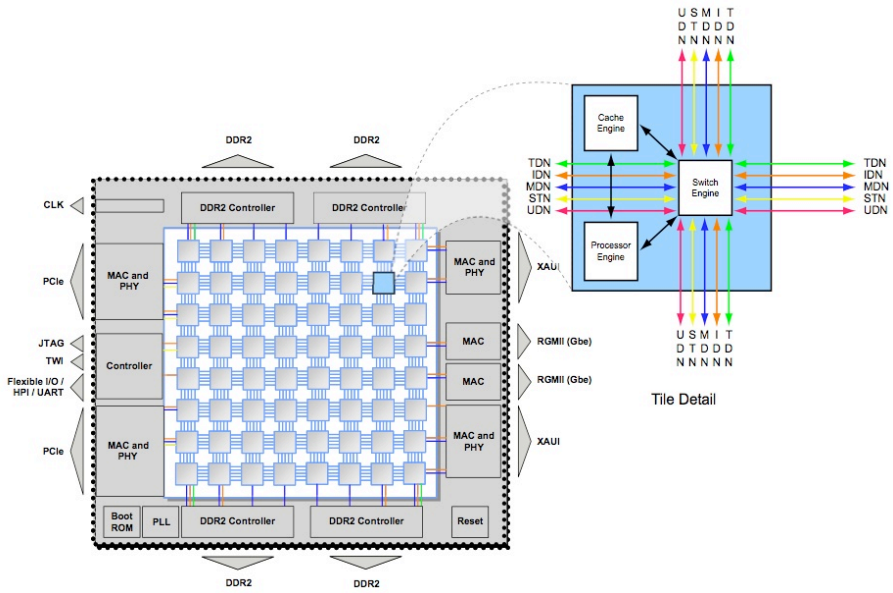


Figura 5.3. Arquitectura del microprocesador Tile64. Esquema extraído de la documentación del fabricante.

Como se puede apreciar, los núcleos están dispuestos en una malla “bidimensional” de 8x8 y están interconectados unos con otros en una red de interconexión denominada iMesh. Esta arquitectura puede incrementar su densidad/complejidad con relativa facilidad, proporcionando un alto ancho de banda y una baja latencia en la comunicación entre tejas. El chip también incluye elementos de comunicación con la memoria externa e interfaces de entrada y salida, pero sobre todo es un microprocesador con muchos núcleos completamente programable. Cada núcleo está compuesto por tres elementos principales:

- ***Motor de procesamiento*** (del inglés, “**processor engine**”): cada teja implementa un microprocesador de enteros de 32 bits utilizando una arquitectura de palabras de órdenes de tres vías (del inglés, “three-way Verly Long Instruction Word”; 3-way VLIW) con su propio contador de programa (del inglés, “Programme Counter”; PC), caché y acceso directo a memoria (del inglés, “Direct Memory Access”; DMA). Una teja es capaz de ejecutar hasta tres operaciones de 32 bits por ciclo de reloj.
- ***Motor de intercambio*** (del inglés, “**switch engine**”): la red de interconexión iMesh se implementa por medio del motor de intercambio, que descarga de las tareas de enrutamiento de los datos al motor de procesamiento. Cada teja contiene un “router” que facilita la transferencia de datos entre tejas e implementa funciones de buffering y control de flujo permitiendo que el usuario no se tenga que preocupar de la sincronización de las tejas.
- ***Motor de caché*** (del inglés, “**cache engine**”): contiene los TLB, cachés y los secuenciadores de caché. Cada teja tiene 16K L1 (8K de instrucciones y 8K de datos) y una caché L2 de 64K. También contiene un motor DMA que gestiona el intercambio de datos entre las tejas y la memoria externa por un lado, y entre las tejas por otro lado.

Tarjeta PCI Express

La arquitectura Tile64 se ensambla en una tarjeta PCI Express denominada TILExpress-20G que incorpora capacidad de programación bajo lenguaje C/C++, como se verá en el siguiente apartado. Como puede verse en la figura 5.4, se trata de una tarjeta de aspecto sencillo, relativamente pequeña y que no aparenta su enorme potencial de cálculo.

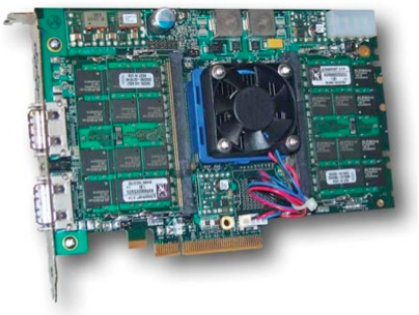


Figura 5.4. Tarjeta TILExpress-20G. Foto extraída de la documentación del fabricante.

Entorno de desarrollo

Una de las características más importantes de las tarjetas Tilera es la posibilidad de desarrollar software específico para aprovechar la potencia de procesamiento de su arquitectura de muchos procesadores.

Así, el desarrollo de aplicaciones sobre el microprocesador Tile64 se agiliza enormemente gracias al entorno de desarrollo multinúcleo de Tilera (del inglés, "Tilera's Multicore Development Environment"; TileMDE o MDE) basado en el proyecto Eclipse, si bien hubiera sido más apropiado denominarlo "de muchos núcleos" ("many-core") por parte del fabricante Tilera, según se ha comentado anteriormente. El MDE agrupa bajo un entorno gráfico común un conjunto de herramientas de comandos para trabajar con las tarjetas Tile64 proporcionando un entorno integrado de desarrollo (del inglés, "Integrated Development Environment"; IDE). Este IDE incorpora un compilador de C/C++ y un simulador del hardware de Tile64, de forma que se puede programar, compilar y ejecutar aplicaciones sin necesidad de disponer de una tarjeta física instalada en el ordenador de desarrollo, simulando el comportamiento de cada teja, como puede comprobarse en <http://www.sicuma.uma.es/manycore/index.jsp?seccion=m64nw/simulation>.

La figura 5.5 muestra el aspecto del entorno de desarrollo TileMDE:

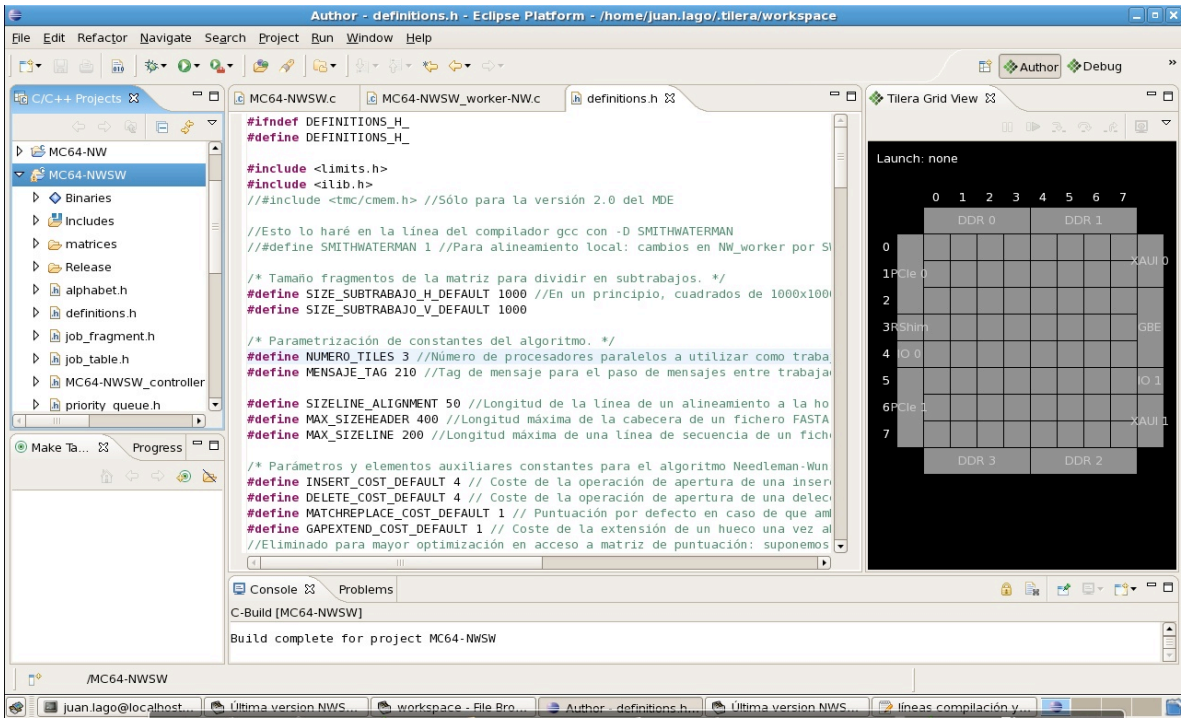


Figura 5.5. Entorno de desarrollo TileMDE.

No obstante, se ha podido comprobar que el simulador es bastante más lento que cuando se utiliza sobre la tarjeta directamente por lo que el rendimiento para el testeo de aplicaciones reales en modo simulador es muy bajo (normalmente hay que restringir el número de tejas con las que se simula para conseguir rendimientos aceptables). Cuando se ejecuta un programa bajo el simulador o también bajo la tarjeta TILExpress, el IDE muestra de forma visual y en tiempo real el uso efectivo de cada teja. En cualquier caso, el simulador es una excelente herramienta visual para demostrar el funcionamiento en paralelo del microprocesador Tile64 (véase el enlace de Internet anteriormente mencionado).

A continuación se describe brevemente el proceso de programación y carga de trabajos en la tarjeta:

i) Inicialización de la tarjeta TILExpress y carga de un vmlinux en cada teja. Las tejas dedicadas a operaciones de Input/Output (I/O) y otras configuraciones de hardware también se configuran durante este proceso de arranque.

ii) Carga de los ficheros del ordenador anfitrión al sistema de ficheros de la tarjeta. Aquí se incluyen los ficheros de trabajo (datos) así como los propios archivos ejecutables (programas) generados por el compilador Tile-GCC.

iii) Lanzamiento de un archivo ejecutable en una teja. Normalmente se suele lanzar un programa controlador que se encargará de planificar la ejecución de código en el resto de tejas, coordinando el resultado y suministrando nuevos trabajos. Otra opción es replicar la misma aplicación en tantos núcleos como sea necesario, ejecutándose en paralelo a partir de este punto.

iv) Cuando termina todo el trabajo, los ficheros de resultados se descargan al ordenador anfitrión para que se puedan utilizar con el propósito que se defina.

Todo este proceso se puede personalizar permitiendo al programador tener un control sobre la gestión del procesamiento y el uso de memoria. Además, se deben tener en consideración los siguientes aspectos de programación:

- La tarjeta puede acceder directamente al sistema de ficheros del ordenador anfitrión.
- Cada teja puede comunicarse de manera independiente con el ordenador anfitrión a través de canales dedicados.
- Se pueden conectar varias tarjetas TILExpress y distribuir la carga de trabajo entre ellas.
- Se pueden agrupar varias tejas bajo un criterio de afinidad funcional.
- La coordinación y comunicación se puede realizar por mecanismos de memoria compartida, paso de mensaje y otras estrategias.

Especial mención merece la biblioteca iLib (véase el siguiente apartado), que permite programar la ejecución paralela y establecer mecanismos de comunicación mediante una interfaz de programación de aplicaciones (del inglés, "Application Programming Interface"; API). Esta biblioteca hace que sea más fácil portar aplicaciones paralelas existentes con objeto de aprovechar el paralelismo que permite el microprocesador Tile64.

Biblioteca iLib

Tilera suministra con sus tarjetas la biblioteca iLib que permite utilizar la arquitectura de su tecnología iMesh de muchos núcleos. Esta biblioteca provee funciones de gestión de procesos y de memoria así como también de comunicación entre procesos y sincronización, simplificando el trabajo de migración de arquitecturas de un núcleo (del inglés, "monocore") a entornos de multiproceso con muchos núcleos.

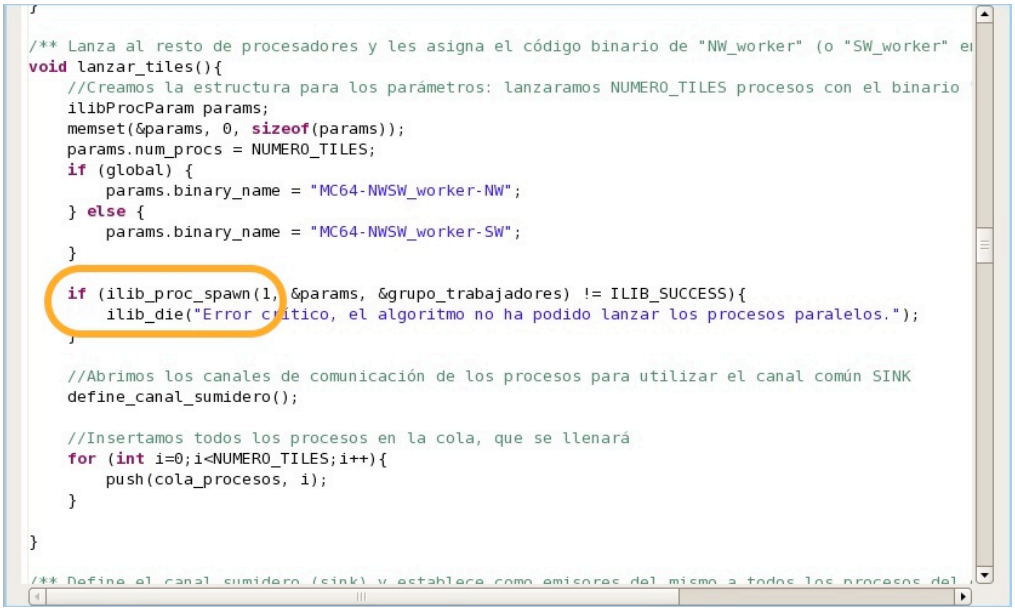
La tarjeta es programable mediante ANSI C estándar a través de una API de funciones que permite al programador gestionar los siguientes elementos clave para la programación en este entorno de muchos núcleos:

Gestión de procesos (del inglés, "**process management**"): con objeto de aprovechar al máximo todos los núcleos del microprocesador Tile64, el desarrollador necesita crear un proceso denominado trabajador (del inglés, "*worker*") en cada uno de los núcleos. Generalmente una aplicación iLib comienza con un sólo proceso (normalmente en el teja 0,0 de la matriz "bidimensional" de núcleos). Posteriormente, este proceso de arranque puede crear nuevos procesos (uno por núcleo), con objeto de utilizar el resto de núcleos disponibles. Una vez que todos los procesos están ejecutándose, se utilizan las primitivas de comunicación de la biblioteca iLib (canales, mensajes y memoria compartida) con objeto de orquestar el trabajo a realizar.

Canales (del inglés, "**channels**"): consisten en interfaces de tipo enchufe (del inglés, "socket") para la transferencia de datos. Los canales implementan una conexión de tipo cola (del inglés, "First In First Out"; FIFO) entre los procesos de envío y recepción, siendo especialmente útiles en aplicaciones donde los datos fluyen de un proceso al siguiente de forma repetida. La biblioteca iLib proporciona varios tipos de canales cada uno con distintas características y compromiso entre flexibilidad y rendimiento.

Paso de mensajes (del inglés "message passing"): el paso de mensajes es apropiado para transferencias de gran cantidad de datos. La API de mensajería soporta operaciones de envío y recepción no bloqueantes, lo que permite que el trabajo pueda solaparse a la espera de mensajes.

La figura 5.6 muestra un ejemplo de código fuente en lenguaje C utilizando algunas llamadas de la biblioteca iLib (`ilib_proc_spawn`, `ilib_die`):



```
}

/** Lanza al resto de procesadores y les asigna el código binario de "NW_worker" (o "SW_worker" e
void lanzar_tiles(){
    //Creamos la estructura para los parámetros: lanzaramos NUMERO_TILES procesos con el binario
    ilibProcParam params;
    memset(&params, 0, sizeof(params));
    params.num_procs = NUMERO_TILES;
    if (global) {
        params.binary_name = "MC64-NWSW_worker-NW";
    } else {
        params.binary_name = "MC64-NWSW_worker-SW";
    }

    if (ilib_proc_spawn(1, &params, &grupo_trabajadores) != ILIB_SUCCESS){
        ilib_die("Error crítico, el algoritmo no ha podido lanzar los procesos paralelos.");
    }

    //Abrimos los canales de comunicación de los procesos para utilizar el canal común SINK
    define_canal_sumidero();

    //Insertamos todos los procesos en la cola, que se llenará
    for (int i=0;i<NUMERO_TILES;i++){
        push(cola_procesos, i);
    }

}

/** Define el canal sumidero (sink) y establece como emisores del mismo a todos los procesos del
```

Figura 5.6. Ejemplo de código fuente en el que se utiliza la biblioteca iLib.

Implementación y resultados 6

Descripción del objetivo

El objetivo de este trabajo es alinear secuencias de residuos de nucleótidos (secuencia interrogante) de gran tamaño (aproximadamente entre 80 y 260 kb) con las secuencias de una base de datos. Las secuencias están representadas en formato FASTA y almacenadas localmente en el servidor que tiene instalada la tarjeta Tilera.

Como se ha visto en el apartado 4, el algoritmo Smith-Waterman consta fundamentalmente de dos etapas: fase de avance o fase 1 (cálculo de la matriz de puntuación) y fase de retroceso o fase 2 (obtención del alineamiento óptimo a partir de la celda con mayor puntuación).

La fase 2 sólo se ejecuta si la puntuación obtenida en el alineamiento durante la fase 1 supera un determinado umbral indicado por el usuario. Por tanto, este valor de umbral es otro de los parámetros que contempla la aplicación desarrollada. En el caso de que tras la ejecución de la fase 1 no se supere el umbral indicado, la aplicación libera la memoria correspondiente de la tarjeta Tilera, eliminando todos los ficheros intermedios generados para la fase 2, pues en esta situación no son necesarios.

Finalmente la aplicación muestra un listado de todas las secuencias de la base de datos que han superado el umbral indicado, ordenados de mayor a menor puntuación, y genera los correspondientes ficheros de alineamientos en formato FASTA.

Igualmente, se realiza la ejecución de varias fases 1 y varias fases 2 simultáneamente, aprovechando la arquitectura de muchos núcleos de la tarjeta TILExpress-20G. Para ello se realiza un reparto equitativo en función de los tiempos de cada fase, y se comprueba si esto supone un ahorro de tiempo comparado con la ejecución secuencial de alineamientos uno a uno.

Hay que destacar, por tanto, que se diferencian dos niveles de paralelismo: el primero, que se da durante la fase uno del algoritmo Smith-Waterman, donde la puntuación del alineamiento óptimo se calcula de forma paralela (es el trabajo de partida del presente trabajo); y el segundo, que se produce al realizar varios procesos (fases uno y dos del algoritmo) de forma simultánea gestionando los recursos y núcleos disponibles para una correcta coordinación del proceso de alineamiento con cada una de las entradas de la base de datos seleccionada. Este es el aspecto en el que se centrará el desarrollo software de este trabajo.

La herramienta desarrollada está disponible mediante servicio Web accesible desde Internet en <http://www.sicuma.uma.es/manycore/index.jsp?seccion=m64sssw/run>, de forma que permite lanzar trabajos en modo similar a otros servidores Web de bioinformática.

Desarrollo previo

El desarrollo referido en el presente proyecto ha partido de la implementación de los algoritmos Needleman-Wunsch y Smith-Waterman (MC64-NW/SW) para alinear dos secuencias, puesta a disposición pública en Internet <http://www.sicuma.uma.es/manycore/index.jsp>. Esta última implementación de referencia (Smith-Waterman) produce el alineamiento local de mayor similitud entre dos secuencias de partida: es lo que se denomina alineamiento local por pares (del inglés "pairwise alignment"). Se basa en una ejecución paralela del algoritmo FastLSA en la que la matriz de filas y columnas se almacena en SSD y no en memoria principal, debido a las limitaciones de esta última en la arquitectura de 32 bits propia del microprocesador Tile64. Este algoritmo requiere los siguientes parámetros:

- --1pass (opcional) o --2pass (por defecto): lanza el algoritmo completo (2pass) o sólo la primera fase para calcular la puntuación del alineamiento.
- --global (opcional; por defecto) o --local: lanza el algoritmo de alineamiento global Needleman-Wunsch (global) o el de alineamiento Smith-Waterman (local).
- Nombre del fichero FASTA con la secuencia interrogante (original a comparar).
- Nombre del fichero FASTA con la secuencia sujeto (del inglés, "subject") con la que comparar.
- Nombre del fichero de salida de texto con las (sub)secuencias FASTA alineadas.
- Nombre del fichero de salida de texto con información acerca del procesamiento realizado para obtener el alineamiento (tiempos de cómputo en cada una de las fases).
- Coste de la operación de apertura de una inserción (hueco en la secuencia original): INSERT_COST.
- Coste de la operación de apertura de una deleción (hueco en la secuencia con la que comparar): DELETE _COST.
- Puntuación por defecto en caso de que ambos nucleótidos coincidan, pero no están en la matriz de puntuaciones: MATCHREPLACE_COST.
- Coste de la extensión de un hueco una vez abierto: GAPEXTEND_COST.
- Fichero de matriz de puntuaciones a utilizar para el cálculo del alineamiento: NUC4.4, MATCH, DAYHOFF, GONNET y numerosas versiones de BLOSUM y PAM.
- Tamaño horizontal de un trabajo (*k* horizontal). La secuencia horizontal (la secuencia que se coloca en la parte superior de la matriz completa de alineamiento) se divide en varias partes o bloques de tamaño *k*. Este parámetro es requerido por el algoritmo FastLSA.
- Tamaño vertical de un trabajo (*k* vertical). Idem para la secuencia vertical.
- Número de tejas a utilizar (sólo durante fase 1).

Aunque ciertos parámetros son de uso poco común (especialmente la diferenciación entre coste de inserción y de deleción), esta implementación los ha incluido con objeto de satisfacer todas las necesidades teóricas que puedan surgir.

Adaptación de MC64-NW/SW

El funcionamiento de MC64-NW/SW está especialmente orientado al alineamiento de secuencias muy largas; e.g., entre 100 kb y un Mb, de tal manera que su rendimiento en actualizaciones de megacélulas por segundo (del inglés, “Mega Cell Updates Per Second”; MegaCUPS o MCUPS) aumenta sensiblemente a medida que las secuencias a alinear se hacen más grandes. Con dicha orientación, cada ejecución del algoritmo hace acopio de todos las tejas del microprocesador Tile64 y sólo son liberados cuando dicha ejecución finaliza. Sin embargo, aunque el diseño de MC64-NW/SW saca el máximo rendimiento de todas las tejas en la fase de avance de FastLSA, la fase de retroceso no se encuentra paralelizada debido a su naturaleza de complejidad lineal. Con esta filosofía, la fase de avance utiliza 60 tejas (ver figura 6.1) como máximo mientras que la de retroceso siempre emplea tan sólo dos tejas.

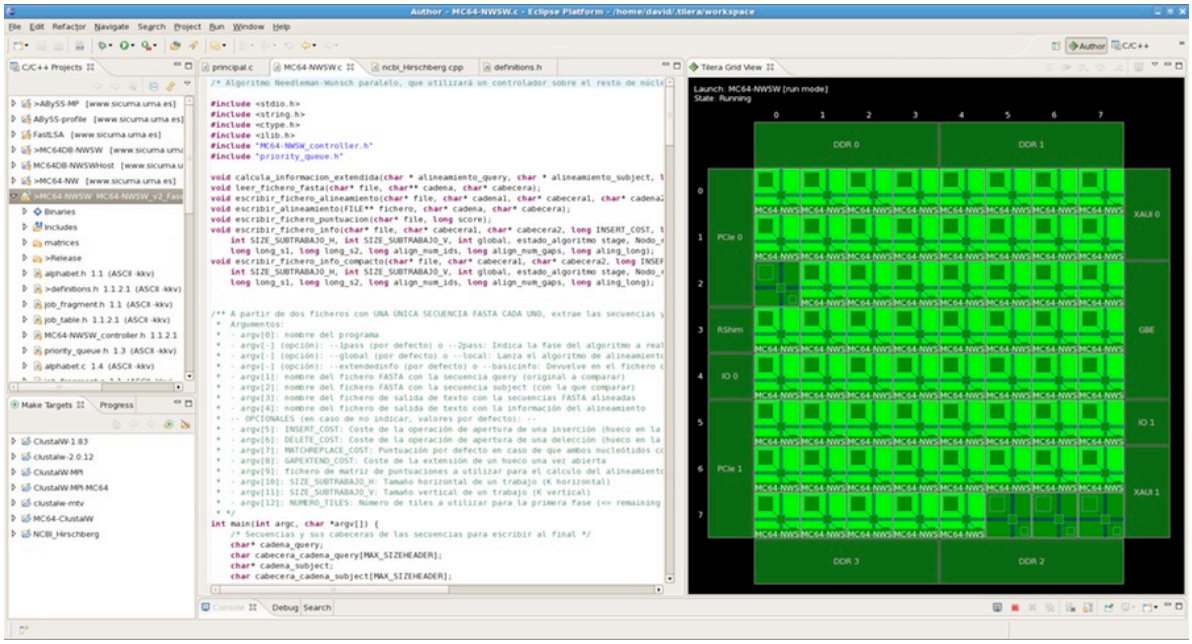


Figura 6.1. Fase de avance con todas las tejas trabajando (excepto los cuatro tiles reservados).

Aunque a primera vista pueda pensarse que las otras 58 tejas quedan libres para ejecutar nuevas tareas durante el retroceso, la realidad es bien diferente: el diseño de la tarjeta es tal que, una vez que una teja se ha incorporado a los cálculos (en la fase de avance) ya no se puede liberar en ningún caso (ver figura 6.2). Es más, cuando dicha teja finaliza la ejecución de la tarea que se le ordenó pasa a un estado irrecuperable hasta que finalice por completo el

proceso raíz que solicitó su intervención en los cálculos. Básicamente, puede asumirse que una teja hace las funciones de un hilo (del inglés “thread”), de forma que sus estados vienen a ser los mismos: disponible, seleccionado o nuevo, en ejecución, bloqueado y muerto.

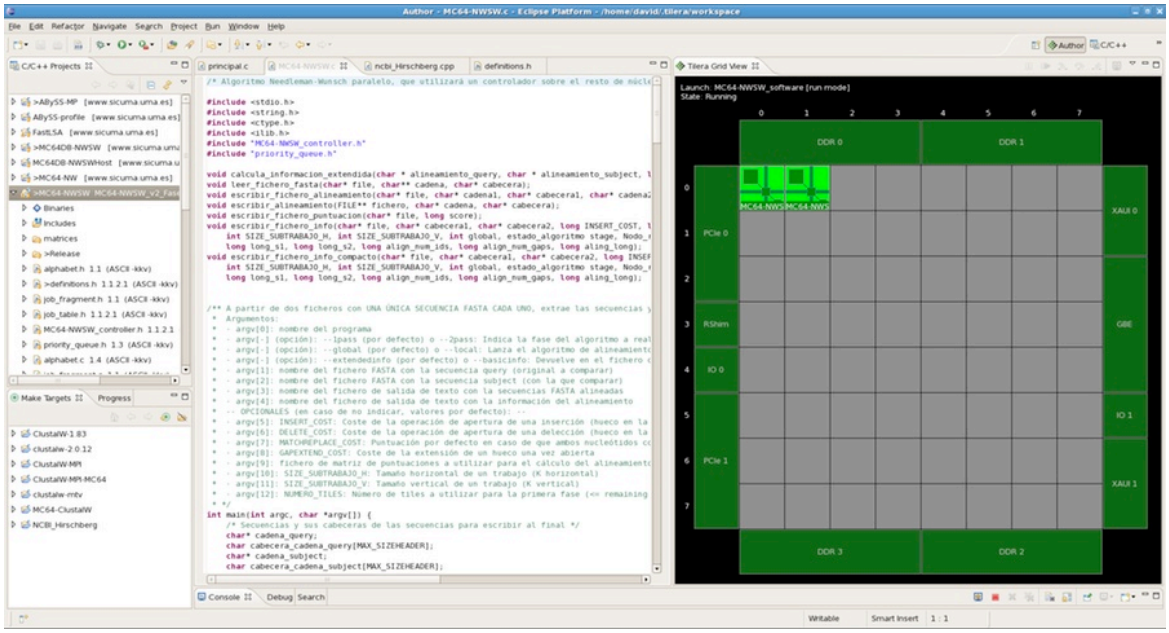


Figura 6.2. Fase de retroceso con sólo dos tejas trabajando. El resto, en gris, no se pueden utilizar.

Por todo lo anterior, para poder obtener un mayor rendimiento de la tarjeta y poder usar las 58 tejas ocupadas mientras se está ejecutando una fase de retroceso, ha sido necesario modificar el algoritmo MC64-NW/SW de tal manera que el proceso raíz que ejecuta la fase de avance sea distinto de aquél que ejecuta la fase de retroceso. En otras palabras, en lugar de generar un solo ejecutable que controle la ejecución de ambas fases, se ha tenido que diseñar dos ejecutables distintos (uno para el avance y otro para el retroceso).

Dado que el algoritmo original utiliza ciertas variables locales de control y de datos cuyo ámbito es común a ambas fases, este cambio previamente indicado tiene la complejidad añadida de la necesidad de guardar los valores de dichas variables en un fichero de estado que sirve de comunicación entre los ejecutables. Por suerte, dado que la matriz de filas y columnas se almacena originalmente en SSD (nótese que dicha matriz se usa en ambas fases del algoritmo), el fichero de estado resulta relativamente pequeño y rápido de almacenar, por lo que todo este cambio no se traduce en un retraso perceptible en el tiempo de ejecución total del algoritmo. La figura 6.3 muestra la disponibilidad (en verde) del resto de tejas cuando se está ejecutando la fase de retroceso y tras realizar los cambios comentados.

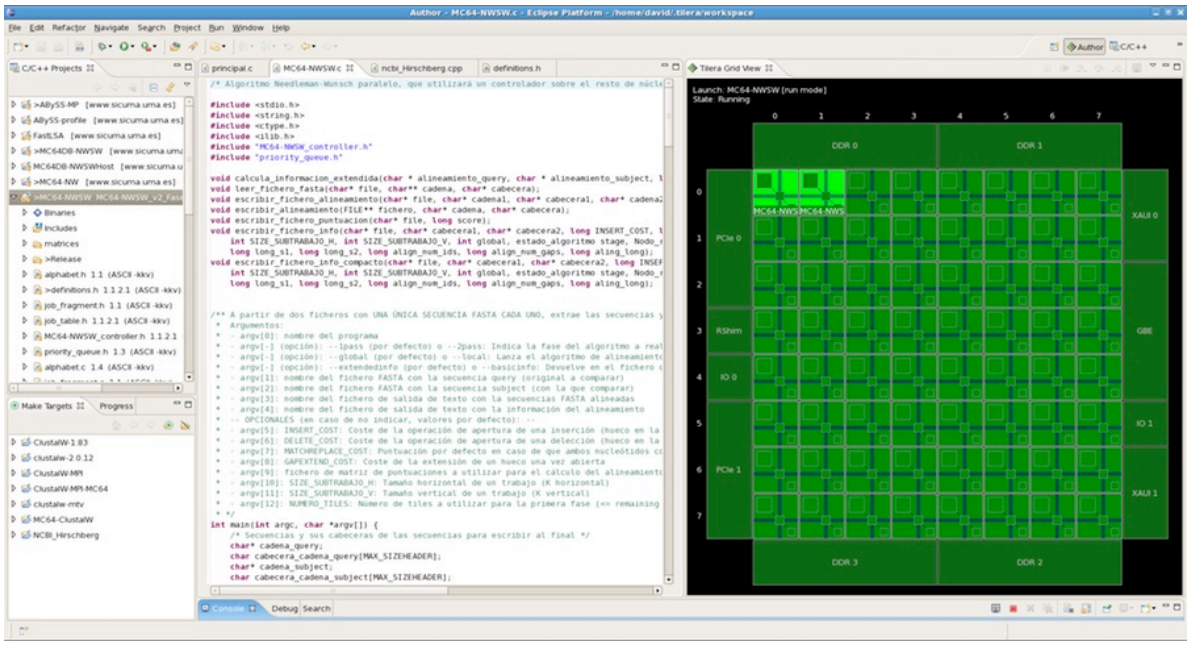


Figura 6.3. Fase 2 utilizando dos tejas, quedando el resto disponible para otros procesos.

Esta nueva implementación se denomina MC64-NW/SW2S, donde el sufijo 2S hace referencia a la existencia de ejecutables distintos para cada una de las dos fases (del inglés, “stages”).

MC64-NW/SW2S y su uso en bases de datos

El rendimiento en MCUPS de MC64-NW/SW (ver figura 6.4) mejora a medida que las secuencias a alinear son más largas, hasta llegar a un Mb, porque es en este límite en el que se saca mayor partido a todos los recursos de la tarjeta TILExpress-20G con valores de *k* al límite de lo permisible en función de la memoria disponible que se usa como SSD; i.e. como sistema de ficheros. Además, tales valores de *k* son lo suficientemente grandes como para aprovechar al máximo la capacidad de cómputo de cada teja sin comprometer el tiempo total de ejecución debido a un tráfico de trabajos excesivo a través de la red interna iMesh.

Este hecho lleva a reflexionar sobre cómo reutilizar las 58 tejas comentadas en el epígrafe anterior, toda vez que la fase de avance de un alineamiento ha finalizado y se ha iniciado su fase de retroceso utilizando sólo dos tejas de las 60 disponibles. En tal situación, si bien se dispone de potencia de cómputo suficiente como para abordar un nuevo alineamiento de secuencias largas, la memoria que se requeriría para ello supera a la actualmente disponible puesto que, no debe olvidarse que la fase de retroceso del primer alineamiento utiliza una gran cantidad de memoria que no puede liberarse hasta que haya finalizado. La conclusión es que el tamaño máximo de las secuencias a alinear en paralelo debe ser lo suficientemente pequeño como para que puedan coexistir simultáneamente en memoria todos los datos requeridos por

cada alineamiento individual, aprovechando así la potencia de cómputo que proporcionan las tejas disponibles.

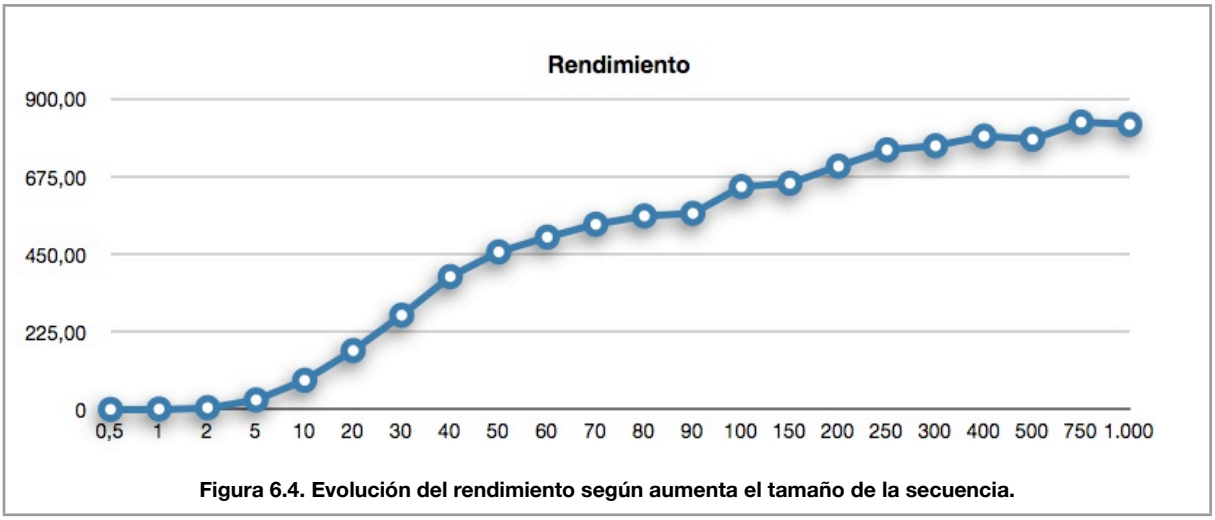


Figura 6.4. Evolución del rendimiento según aumenta el tamaño de la secuencia.

De esta manera, por ejemplo, si se inicia un alineamiento de secuencia de un Mb, será imposible iniciar la ejecución de ningún otro alineamiento hasta que se haya completado la fase de retroceso del primero, aunque haya 58 tejas libres. Esto lleva a la necesidad de establecer un consenso entre longitud máxima de secuencias a alinear y máximo número de tejas que se está dispuesto a desaprovechar.

Por otro lado, y al margen de lo anterior, el algoritmo MC64-NW/SW original aprovecha todas las tejas disponibles en función de la relación que existe entre las longitudes de las secuencias a alinear y el valor de *k* usado internamente. En otras palabras, si, por ejemplo, las secuencias de partida son de longitud 100 kb y se elige un *k* de 2.500, entonces la matriz de trabajos tendrá un tamaño de 40×40. Debido a la naturaleza de FastLSA, el número máximo de trabajos que pueden ejecutarse en paralelo (sólo la fase de avance es paralela) es igual a la dimensión de dicha matriz (en el caso de que las longitudes de las secuencias a alinear no sean iguales, debe escogerse la dimensión más pequeña) por lo que jamás podrá sacarse partido a más de 40 tejas trabajando a la vez. En conclusión, hay situaciones en las que resulta contraproducente asignar todas las tejas libres a un trabajo. Es más, con una matriz de 40×40, si se numeran las tejas del 1 al 40, puede decirse que, por ejemplo, la teja40 sólo trabaja una vez (diagonal principal ascendente), la teja39 trabaja tres veces (diagonal principal ascendente y sus dos adyacentes), la teja38 trabaja cinco veces y así sucesivamente hasta la teja1 que trabaja 79 veces (nótese que 1+3+5+…+79=40+[2*(0+1+2+3+…+39)]=40+2*[(0+39)/2*40] =40+39*40=1600=40x40), por lo que el aprovechamiento de las tejas dista mucho de ser óptimo. Si bien esta situación puede ser aceptable en MC64-NW/SW cuando el objetivo es hacer únicamente un alineamiento, la situación cambia drásticamente cuando se desea obtener el máximo partido a todas las tejas durante la mayor parte del tiempo.

En general, el aprovechamiento máximo de todas las capacidades de una TILExpress-20G cuando se desean hacer numerosos alineamientos por pares depende de la cantidad de memoria disponible. Además, se deben tener en cuenta la memoria local,

compartida y SSD, el número de tejas libres y las longitudes de las secuencias que intervienen en cada alineamiento (en nuestro caso, una de las secuencias participa en todos los alineamientos). La única solución factible consiste en aplicar un algoritmo de planificación que maximice dicho uso y que, a su vez, se base en una tabla de tiempos con tres dimensiones: longitud de la diagonal, tejas asignadas y valor de *k*. Es decir, partiendo de pruebas empíricas que digan cuánto tiempo tarda en resolverse una primera fase con determinado ***k***, **n** tejas asignadas, y **d** tejas como máximo trabajando en paralelo, podría rellenarse una matriz **M** tal que permitiese aplicar posteriormente un algoritmo de planificación que genere el orden en que deben lanzarse los alineamientos, así como el valor de *k* que debe utilizarse y el número de tejas que se deben asignar para que el proceso completo se ejecute en el mínimo tiempo posible.

Además de lo anterior, es necesario añadir un componente casuístico adicional. El objetivo es proporcionar el alineamiento local completo, al igual que hacen programas como el BLAST, en aquellas situaciones en que el alineamiento local pueda tener sentido; e.g., en caso de que la puntuación más elevada de la matriz de programación dinámica (del inglés, “Dynamic Programming Matrix”; DPM) supere cierto umbral. Por tanto, cualquier planificación que pueda hacerse *a priori* puede quedar desbaratada por la necesidad de decidir dinámicamente qué cálculos adicionales (fase de retroceso) deben hacerse, en función de la información que proporcionen las fases de avance de cada alineamiento. Esto puede implicar un recálculo de la planificación cada vez que fuera necesario incluir una ejecución imprevista de una fase de retroceso. Por supuesto, esta visión parte de la base de que el número de resultados será muy inferior al número de secuencias en la base de datos. De otra forma, la planificación debería hacerse considerando que siempre será necesario ejecutar la fase de retroceso, y sólo replanificarse en caso de que dicha ejecución resultase ser no necesaria.

Todo este estudio ha hecho que se tome la decisión de realizar únicamente un mecanismo de planificación que permita estimar la ganancia de rendimiento ante una ejecución secuencial de los alineamientos. Además, se van a tomar valores de ***k*** en función del tamaño de la DPM, y un valor de **n** fijo asumiendo que las secuencias a alinear son siempre de una longitud no excesivamente distinta y no superior a cierto valor crítico. Esta última decisión también permite estimar de antemano un límite por exceso en el número máximo de trabajos que pueden estar almacenados en la memoria de la tarjeta TILExpress-20G. Los resultados y conclusiones que se obtienen con esta planificación, sirven además para decidir si merece la pena que un futuro trabajo aborde un mecanismo de planificación mucho más sofisticado.

Se han encontrado referencias a la complejidad de este asunto también en Driga et al [38]. Dichos autores indican que sería interesante desarrollar una aproximación heurística simple y fiable para encontrar el mejor valor de *k* que asegure que el tiempo total del alineamiento se aproxime al óptimo teórico. No obstante, tal aproximación queda fuera del alcance también de su estudio.

La tabla 6.1. muestra la relación entre el mejor valor de *k* y los tamaños de las secuencias que se ha obtenido a partir de pruebas empíricas.

Tabla 6.1. Aproximación de mejores valores de k en función del tamaño de las secuencias.

Longitud n (kb)	Mejor *k*	Tiempo (segundos)	n^2	Rendimiento n^2/t (MCUPS)
0,5	250	0,71	0,25	0,35
1	200	0,70	1,00	1,43
2	200	0,76	4,00	5,26
5	300	0,90	25,00	27,78
10	300	1,18	100,00	84,75
20	500	2,34	400,00	170,94
30	500	3,29	900,00	273,56
40	675	4,15	1.600,00	385,54
50	675	5,47	2.500,00	457,04
60	700	7,20	3.600,00	500,00
70	700	9,12	4.900,00	537,28
80	800	11,40	6.400,00	561,40
90	800	14,24	8.100,00	568,82
100	900	15,47	10.000,00	646,41
150	1.400	34,29	22.500,00	656,17
200	1.700	56,68	40.000,00	705,72
250	1.700	83,00	62.500,00	753,01
300	1.750	117,66	90.000,00	764,92
400	1.750	201,77	160.000,00	792,98
500	1.750	318,96	250.000,00	783,80
750	1.850	674,90	562.500,00	833,46
1.000	3.000	1.209,20	1.000.000,00	826,99

Diseño de la solución propuesta

El entorno hardware y software con el que se parte es un servidor Quad Core Intel Xeon 2.0 GHz con 8 GB de memoria RAM DDR2, con sistema operativo CentOS 5 y versión del kernel 2.6.18-128.el5. Este servidor tiene instalada una tarjeta TILExpress-20G.

El desarrollo de aplicaciones para el microprocesador Tile64 implica la creación de ejecutables en el servidor y la carga de los mismos en la tarjeta, que ejecuta un sistema basado también en Linux. Para facilitar este proceso de desarrollo, Tilera suministra una serie de utilidades que funcionan en ambas partes (servidor y procesador Tile64), como se puede apreciar en la figura 6.5:

- El programa *tile-monitor* (monitor de tejas) se ejecuta en el microprocesador del servidor. Gestiona la comunicación entre las herramientas de desarrollo que funcionan en el servidor. Entre ellas se encuentran, por ejemplo, el IDE y los depuradores, así como el proceso *shepherd* que se ejecuta en el microprocesador Tile64.
- El proceso *shepherd* (pastor) permite el desarrollo y ejecución de aplicaciones comunicándose con la parte del servidor del *tile-monitor* y desde allí con el IDE basado en Eclipse. Además, el proceso *shepherd* controla y lee el estado de los procesos que se ejecutan en las tejas y facilita esta información al proceso *tile-monitor*.

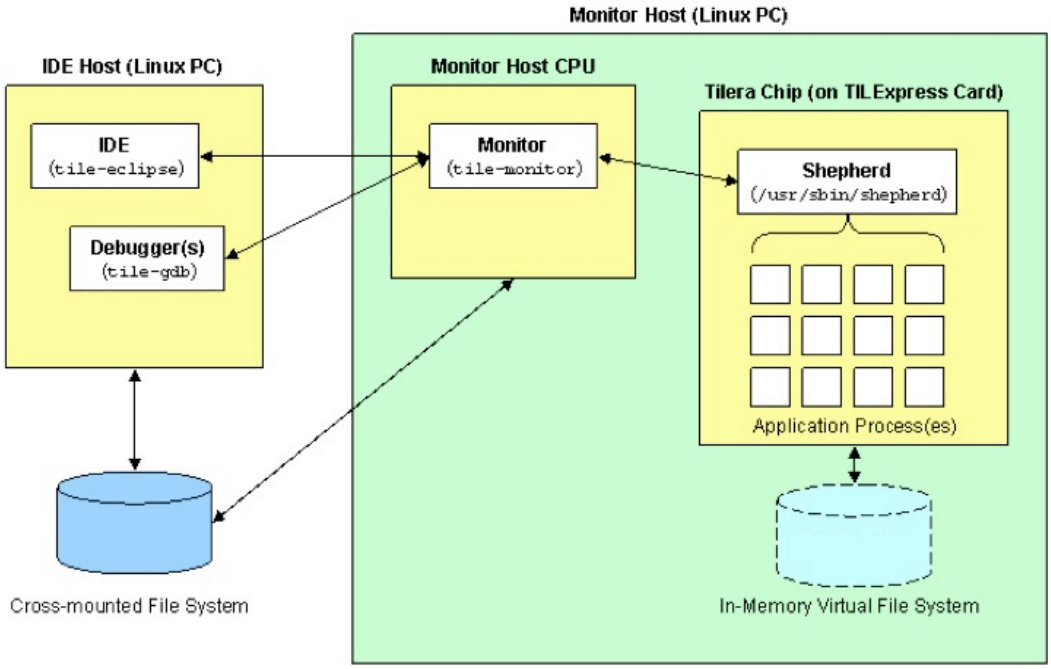


Figura 6.5. Utilidades y su ubicación respecto al servidor y el microprocesador Tile64.

De esta forma, para el lanzamiento de las distintas fases del algoritmo Smith-Waterman, se usa de la utilidad `tile-monitor` desde la línea de comandos del sistema. Esta herramienta tiene dos modos de funcionamiento: simulación (por software) y hardware (ejecución en la tarjeta TILExpress-20G).

El modo de funcionamiento se indica con el parámetro `--simulator` o `--pci` respectivamente. Por ejemplo, para arrancar utilizando el simulador (no la tarjeta) se utiliza la siguiente sintaxis:

```
$ tile-monitor --simulator
```

Se puede obtener ayuda del comando `tile-monitor` desde la misma línea de comando, usando las siguientes opciones:

```
$ tile-monitor --help-examples
```

```
$ tile-monitor --help-commands
```

```
$ tile-monitor --help_options
```

Las páginas de ayuda se encuentran igualmente disponibles en la línea de comandos:

```
$ tile-man tile-monitor
```

El comando `tile-monitor` se puede utilizar por lotes (del inglés, “batch”) o de forma interactiva (con intervención del usuario en cada instrucción).

```
$ tile-monitor --pci
```

`Command:`

En modo interactivo, aparece un nuevo carácter de interacción (del inglés, “prompt”) que sustituye al del sistema (el símbolo $ tradicional de los sistema Unix) y que indica que `tile-monitor` está esperando el siguiente comando.

Entre los comandos más interesantes que se utilizarán en el diseño de la solución propuesta se encuentra `shepherd-tiles`, que permite indicar cuántas tejas (e incluso concretar cuáles exactamente) se dedicarán en el próximo comando de lanzamiento de aplicación. Por otro lado, se utilizará el modo de lanzamiento de aplicaciones en segundo plano (del inglés, “background”) mediante el modificador **&** de Unix.

Utilizando estos dos recursos (comando shepherd-tiles y ejecución en segundo plano), se realiza un reparto de las tejas de la tarjeta y se asigna cada bloque (conjunto de tejas) a un determinado tipo de tarea (aplicación software), como se explica a continuación.

Las 64 tejas del microprocesador Tile64 se pueden ver como una matriz “bidimensional” de 8 filas x 8 columnas (figura 6.6) en el que el elemento o teja de la esquina superior izquierda e inferior derecha tienen las coordenadas 0,0 y 7x7 respectivamente. También se pueden numerar secuencialmente desde 0 hasta el 63, de izquierda a derecha y de arriba abajo empezando en la teja con coordenada 0,0.

0,0	1,0	2,0	3,0	4,0	5,0	6,0	7,0
0,1	1,1	2,1	3,1	4,1	5,1	6,1	7,1
0,2	1,2	2,2	3,2	4,2	5,2	6,2	7,2
0,3	1,3	2,3	3,3	4,3	5,3	6,3	7,3
0,4	1,4	2,4	3,4	4,4	5,4	6,4	7,4
0,5	1,5	2,5	3,5	4,5	5,5	6,5	7,5
0,6	1,6	2,6	3,6	4,6	5,6	6,6	7,6
0,7	1,7	2,7	3,7	4,7	5,7	6,7	7,7

Figura 6.6. Coordenadas de las distintas tejas. En gris se representan las cuatro tejas reservadas por la tarjeta para su gestión interna. Por tanto, existen 60 tejas libres del total de 64 existentes en el microprocesador Tile64.

Antes de cada lanzamiento de una aplicación, se puede indicar qué tejas reservar de varias maneras:

- Tejas sueltas: como por ejemplo, `shepherd-tiles 5`
- Secuencia de tejas: por ejemplo, `shepherd-tiles 0,1,2,3,4`
- Una submatriz: como por ejemplo, `shepherd-tiles 2x2` (equivale a las tejas 0, 1, 8 y 9).

- Una submatriz y una coordenada: como por ejemplo, `shepherd-tiles 7x2@1,2` que reserva 14 tejas (siete columnas y dos filas) en total a partir de la teja con coordenada 1,2 (indicadas en azul en la figura 6.7).

0,0	1,0	2,0	3,0	4,0	5,0	6,0	7,0
0,1	1,1	2,1	3,1	4,1	5,1	6,1	7,1
0,2	1,2	2,2	3,2	4,2	5,2	6,2	7,2
0,3	1,3	2,3	3,3	4,3	5,3	6,3	7,3
0,4	1,4	2,4	3,4	4,4	5,4	6,4	7,4
0,5	1,5	2,5	3,5	4,5	5,5	6,5	7,5
0,6	1,6	2,6	3,6	4,6	5,6	6,6	7,6
0,7	1,7	2,7	3,7	4,7	5,7	6,7	7,7

Figura 6.7. En azul, tejas reservadas con el comando `tiles 7x2@1,2`.

Es importante tener en cuenta la geometría que se utiliza a la hora de reservar las tejas para optimizar su uso, evitando, por ejemplo, que sean reservadas pero no usadas. Esto es así porque el comando `tile-monitor` siempre intenta agrupar las tejas bajo un cuadrado o rectángulo que las contenga a todas. Por ejemplo, sea la siguiente reserva de tejas:

```
Command: shepherd-tiles 0,1,2,3,4,5,6,7,8
```

La figura 6.8 muestra en azul las tejas que se desean reservar y en rojo las tejas adicionales necesarias para construir un rectángulo que incluya a las tejas 0 a 8. Por tanto, en este caso hay siete tejas reservadas más de las solicitadas y dependiendo de la aplicación a ejecutar en ellas, pueden quedar desaprovechadas.

0,0	1,0	2,0	3,0	4,0	5,0	6,0	7,0
0,1	1,1	2,1	3,1	4,1	5,1	6,1	7,1
0,2	1,2	2,2	3,2	4,2	5,2	6,2	7,2
0,3	1,3	2,3	3,3	4,3	5,3	6,3	7,3
0,4	1,4	2,4	3,4	4,4	5,4	6,4	7,4
0,5	1,5	2,5	3,5	4,5	5,5	6,5	7,5
0,6	1,6	2,6	3,6	4,6	5,6	6,6	7,6
0,7	1,7	2,7	3,7	4,7	5,7	6,7	7,7

Figura 6.8. En azul se indican las tejas teóricas reservadas. En rojo se indican también las tejas reservadas pero posiblemente no utilizadas.

Además, con el ejemplo anterior se puede producir otro efecto no deseado como puede ser un conflicto de reserva con otro comando `shepherd-tiles`. Efectivamente, si después de la reserva anterior, se pretende realizar otra reserva como por ejemplo `shepherd-tiles 12 13 14 15` (que se corresponde con las coordenadas 4,1 - 5,1 - 6,1 y 7,1), se producirá un error al lanzar la aplicación, pues dichas tejas han sido reservadas previamente (en rojo), aunque no de forma explícita.

Teniendo estos aspectos en cuenta, se ha optado por utilizar el formato de reserva de tejas basado en submatriz con coordenada (formato *NxM@x,y*) porque evita la reserva implícita de tejas minimizando los posibles conflictos de reservas entre lanzamientos.

Con todas estas consideraciones, finalmente se ha desarrollado una ***script* Perl** que gestiona todo el proceso de lanzamiento y ejecución de trabajos a través del comando `tile-monitor`. La funcionalidad de esta *script* se resume en los siguientes puntos:

- Ejecución y control del comando `tile-monitor`, gestionando su canal de entrada, salida y error estándar para comunicarse con él, con objeto de poder enviarle nuevos comandos y leyendo posibles mensajes de error y la salida que produce. Se hace uso de la potencia del lenguaje Perl en expresiones regulares y la lectura no bloqueante de los mencionados canales de entrada/salida para permitir el lanzamiento de múltiples alineamientos de forma simultánea.

- Carga en la memoria de la tarjeta Tilera de los ejecutables *controller* (controlador) y *worker* (trabajador), correspondientes a la implementación paralela del algoritmo Smith-Waterman junto con la matriz de puntuación que se desea utilizar en el alineamiento.

- Lectura y carga en la tarjeta de los ficheros FASTA de la secuencia interrogante y cada una de las secuencias de la base de datos con las que realizar el alineamiento. La *script* Perl recibirá varios parámetros, entre los que se encuentra el valor de umbral que determina si ejecutar o no la fase 2, evitando, en su caso, el desbordamiento de la memoria disponible. Otros parámetros a controlar son los esquemas de penalización por huecos, la matriz de puntuación comentada en el punto anterior y el directorio de salida con los resultados (ficheros de alineamiento e informe de resultados de la búsqueda).

- Ejecución y control de los trabajos de alineamiento con cada una de las secuencias de la base de datos, controlando si es necesario lanzar la fase 2, en función de la puntuación obtenida del alineamiento durante la fase 1.

- Gestión de la cola de trabajos en curso, terminados y pendientes del comando `tile-monitor`. Asimismo, la ejecución de trabajos en paralelo, en función de la estrategia de asignación de tejas que se defina y controlando que no se desborde el uso de la memoria máxima disponible en la tarjeta, al tiempo que se gestiona la disponibilidad de tejas libres para próximos trabajos.

- Obtención de los alineamientos (ficheros en formato FASTA) de aquellas secuencias que hayan superado el umbral indicado por el usuario.

- Consolidación de datos y generación de un listado de las secuencias que hayan superado el umbral, ordenado de mayor a menor puntuación y suministrando información sobre el porcentaje de similitud, número de huecos introducidos y posiciones de inicio y fin del mejor alineamiento con cada secuencia (lo cual determina su longitud). Asimismo, cada uno de los alineamientos locales también estarán disponibles para su visualización.

El esquema de la aplicación se resume en la figura 6.9.

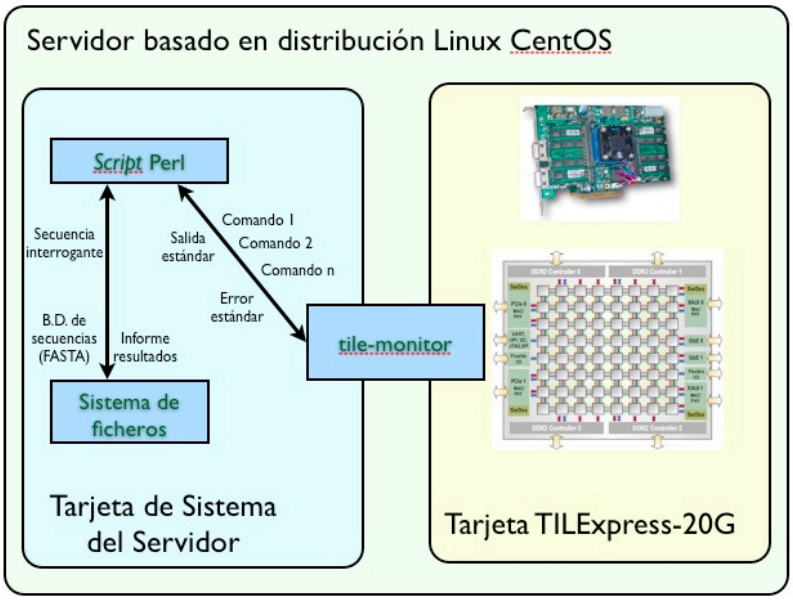


Figura 6.9. Esquema básico de la aplicación desarrollada.

Estrategia de reparto de recursos

Se han realizado mediciones de tiempos de forma empírica (que se comentan a continuación) para determinar la mejor geometría y reparto de tejas por fases para resolver el alineamiento planteado. Una vez elegida la estrategia, se implementa el algoritmo y se comprueba que los resultados son los esperados.

Tiempos de fase 1 y fase 2 y su ratio en secuencias muy similares (250 kb)

La tabla 6.2 muestra la medición del tiempo de ejecución en segundos de una fase 1 con dos secuencias muy similares de 250 kb cada una, utilizando para ello todas las tejas disponibles.

Tabla 6.2: tiempos de fase 1, fase 2 (en segundos) y ratio entre ambas.

Lanzamiento	Fase 1	Fase 2	Total	Ratio
1	117	42	159	2,8

La fase 1 tarda 117 segundos (casi dos minutos) frente a los 42 segundos de la fase 2. Conviene destacar que al ser las secuencias muy similares, la fase 2 es más costosa por lo que se obtiene una primera ratio de 2,8 para casos extremos. Es decir, una fase 1 tarda alrededor

2,8 veces lo que tarda una fase 2 o, dicho de otra forma, en el tiempo durante el que se realiza una fase 2, se pueden realizar 2,8 fases 1.

Varias fases 1 sin borrar ficheros temporales

Esta es una prueba de estrés sobre la capacidad de memoria de la tarjeta TILExpress-20G. Se trata de ejecutar 10 fases 1 sobre dos secuencias de 250 kb muy similares y sin borrar los ficheros temporales generados (necesarios para la fase 2). Después de estas 10 ejecuciones, se está próximo al límite de capacidad de memoria de la tarjeta, pues equivale aproximadamente a un alineamiento de dos secuencias de un Mb (realmente sería el equivalente a dividir la secuencia horizontal y vertical en 3 partes, 1.000 / 3 = 333 kb, es decir, 9 cuadrantes de 333 x 333 kb cada uno), según se ilustra en la figura 6.10.

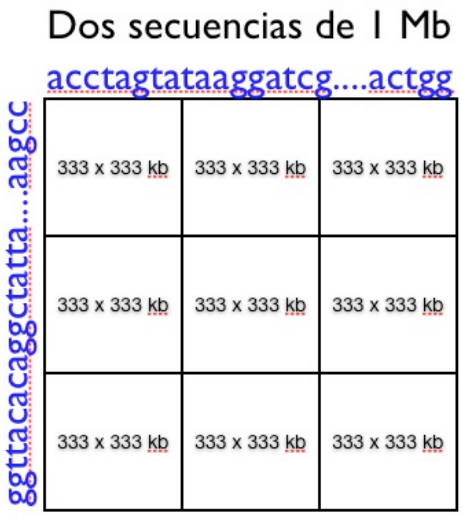


Figura 6.10. División de un alineamiento de 1 Mb x 1 Mb en nueve partes para aproximar el equivalente en varios alineamientos cercanos a los 250 kb.

La tabla 6.3 muestra que los tiempos entre ejecuciones se mantienen constantes y no influye el incremento de almacenamiento interno ocupado.

Tabla 6.3. Tiempos constantes en 10 fases 1 (en segundos), sin borrar ficheros temporales. Prueba de capacidad de memoria de la tarjeta TILExpress-20G.

Lanzamiento	Tiempo de fase 1 (segundos)
1	117
2	117
3	117
4	116
5	117
6	117
7	117
8	116
9	116
10	117

Variación de tiempos de fase 1 según la geometría de tejas utilizada (alineamiento de secuencias de 250 kb)

Con esta prueba se miden los tiempos de fase 1 al alinear dos secuencias muy parecidas de 250 kb cada una, variando la geometría de tejas entre ejecuciones y, por tanto, el número de tejas utilizadas. También se realizan ejecuciones de forma paralela de varias fases 1 y se comprueba que los tiempos se mantienen constantes.

Así, por ejemplo, se tarda lo mismo en lanzar una fase 1 con una geometría 7x2 que cuatro fases 1 con la misma geometría 7x2. Esto demuestra la independencia de las tejas, cuyo rendimiento no se ve afectado por tareas de comunicación entre ellas (los tiempos de comunicación no afectan al rendimiento del microprocesador). Por tanto, la tabla 6.4 muestra una aproximación de los tiempos que se tardaría en procesar una base de datos de 50, 300, 1.000 y 7.000 entradas respectivamente (por poner varios ejemplos). En rojo y sombreados se indican los valores mínimos de tiempos encontrados y las geometrías que permiten obtenerlos.

Tabla 6.4. Tiempos de ejecución de fase 1 (en segundos) al alinear secuencias de 250 kb usando distintas geometrías.

Geometría	Número de tejas	Fase 1	Concurrencia[1]	50		300		1000		7000	
				n[2]	t Total[3]	n	t Total	n	t Total	n	t Total
7x8	56	117	1	50	5.850	300	35.100	1.000	117.000	7.000	819.000
7x7	49	132	1	50	6.600	300	39.600	1.000	132.000	7.000	924.000
6x8	48	146	1	50	7.300	300	43.800	1.000	146.000	7.000	1.022.000
7x6	42	151	1	50	7.550	300	45.300	1.000	151.000	7.000	1.057.000
5x8	40	154	1	50	7.700	300	46.200	1.000	154.000	7.000	1.078.000
7x5	35	183	1	50	9.150	300	54.900	1.000	183.000	7.000	1.281.000
4x8	32	188	1	50	9.400	300	56.400	1.000	188.000	7.000	1.316.000
7x4	28	220	2	25	***5.500***	150	33.000	500	110.000	3.500	770.000
3x8	24	255	2	25	6.375	150	38.250	500	127.500	3.500	892.500
7x3	21	290	2	25	7.250	150	43.500	500	145.000	3.500	1.015.000
2x8	16	364	3	17	6.188	100	36.400	334	121.576	2.334	849.576
7x2	14	433	4	13	5.629	75	***32.475***	250	***108.250***	1.750	757.750
1x8	8	757	7	8	6.056	43	32.551	143	108.251	1.000	***757.000***
7x1	7	900	8	7	6.300	38	34.200	125	112.500	875	787.500
				Mínimo	**5.500**		**32.475**		**108.250**		**757.000**

(1) Número de alineamientos simultáneos que se pueden realizar con la geometría seleccionada en la misma fila.

(2) Número total de alineamientos no concurrentes que hay que realizar.

(3) Tiempo total (en segundos) necesario para realizar el alineamiento de la secuencia interrogante con todas y cada una de las secuencias de una base de datos con el número de entradas indicadas en su misma columna.

En color rojo y sombreado se resaltan los mejores tiempos obtenidos para cada combinación de base de datos y geometría.

Se puede observar que para una base de datos de tamaño pequeño (50 entradas), la geometría que permite procesarla en el menor tiempo es la 7x4, mientras que para tamaños de bases de datos de 300 ó 1.000 entradas, la geometría más eficiente es la 7x2. Conforme aumenta el número de entradas de la base de datos analizada, el óptimo se aproxima a una geometría 7x2, derivando hacia geometría 1x8 para tamaños de base de datos grandes.

Hay que hacer notar que en determinados casos se desperdician tejas sueltas que no encajan en una agrupación. Así, por ejemplo, con una geometría 1x8 se pueden realizar hasta

siete trabajos simultáneos (sobran cuatro tejas que sumadas a las cuatro reservadas, suman un total de (7 x 8) + 4 sobrantes + 4 reservadas = 64 tejas en total). Sin embargo, con la geometría 2x8 se pueden realizar tres ejecuciones paralelas, pero se desaprovecha una columna de ocho tejas (como puede verse en la última columna de la figura 6.11).

0,0	1,0	2,0	3,0	4,0	5,0	6,0	7,0
0,1	1,1	2,1	3,1	4,1	5,1	6,1	7,1
0,2	1,2	2,2	3,2	4,2	5,2	6,2	7,2
0,3	1,3	2,3	3,3	4,3	5,3	6,3	7,3
0,4	1,4	2,4	3,4	4,4	5,4	6,4	7,4
0,5	1,5	2,5	3,5	4,5	5,5	6,5	7,5
0,6	1,6	2,6	3,6	4,6	5,6	6,6	7,6
0,7	1,7	2,7	3,7	4,7	5,7	6,7	7,7

Figura 6.11. Se indica en tonos verdes, cada una de las ejecuciones paralelas de geometrías 2x8. Nótese que la última columna de la derecha representa tejas que se desaprovechan.

Variación de tiempos de fase 1 según la geometría de tejas utilizada (alineamiento de secuencias de 125 kb x 109 kb)

En este caso se repite la prueba anterior, pero con dos secuencias más distantes (mayor número de diferencias en sus residuos) y más cortas de longitud. Los datos de la tabla 6.6 corresponde a alineamientos de secuencias del virus del herpes humano (125 kb de longitud) y bovino (109 kb de longitud). Este es un caso más realista de los tiempos que se pueden obtener.

Tabla 6.5. Tiempos de ejecución de fase 1 (en segundos) al alinear secuencias de 125 kb x 109 kb usando distintas geometrías.

Geometría	Tiles	Fase 1	Concurrencia	26		300		1000		7000	
				n	t Total	n	t Total	n	t Total	n	t Total
7x8	56	34	1	26	884	300	10.200	1.000	34.000	7.000	238.000
7x7	49	34	1	26	884	300	10.200	1.000	34.000	7.000	238.000
6x8	48	36	1	26	936	300	10.800	1.000	36.000	7.000	252.000
7x6	42	37	1	26	962	300	11.100	1.000	37.000	7.000	259.000
5x8	40	37	1	26	962	300	11.100	1.000	37.000	7.000	259.000
7x5	35	47	1	26	1.222	300	14.100	1.000	47.000	7.000	329.000
4x8	32	48	1	26	1.248	300	14.400	1.000	48.000	7.000	336.000
7x4	28	50	2	13	650	150	7.500	500	25.000	3.500	175.000
3x8	24	61	2	13	793	150	9.150	500	30.500	3.500	213.500
7x3	21	64	2	13	832	150	9.600	500	32.000	3.500	224.000
2x8	16	89	3	9	801	100	8.900	334	29.726	2.334	207.726
7x2	14	92	4	7	***644***	75	***6.900***	250	***23.000***	1.750	***161.000***
1x8	8	174	7	4	696	43	7.482	143	24.882	1.000	174.000
7x1	7	195	8	4	780	38	7.410	125	24.375	875	170.625
			Mínimo		**644,0**		**6.900,0**		**23.000,0**		**161.000,0**
Alrededor de un 32% de mejora de tiempo respecto a la ejecución secuencial											

Nótese como para una base de datos de 1.000 entradas, la geometría 7x2 tarda 23.000 segundos (6’39 h) en alinear dos secuencias dadas, mientras que con una ejecución secuencial (geometría 7x8) se tarda 34.000 segundos (9’44 h). Esto supone un incremento del

rendimiento de aproximadamente el 32% al procesar la base de datos con hasta cuatro geometrías paralelas de 7x2 frente a realizarlo de forma secuencial con todas las tejas disponibles.

Variación de tiempos de fase 2 según esquemas de penalización (alineamiento de secuencias de 125 kb x 109 kb)

La tabla 6.6 muestra los tiempos de fase 2 en función de los esquemas de puntuación utilizados, que tienen el formato “A,B,C,D” donde A es el coste por inserción de hueco, B el coste por deleciones, C el coste por emparejamiento y sustitución (del inglés, “match_replace”) y D el coste por extensión de hueco.

Las secuencias son las del virus del herpes humano y bovino (125 kb x 109 kb). Se observa que los tiempos de fase 2 pueden llegar a ser despreciables con determinados esquemas de puntuación (menores de un segundo).

Tabla 6.6. Tiempos de ejecución de fase 2 (en segundos) en el alineamiento de secuencias de 125 kb x 109 kb, variando esquema de penalización.

Esquema	Tiempo fase 2 (segundos)	Longitud alineamiento
4,4,1,1	16,67	148.576
4,4,2,2	16,13	138.321
4,4,3,3	16,17	130.349
4,4,4,4	15,97	123.760
4,4,5,5	12,68	95.752
4,4,6,6	0,65	3.077
4,4,7,7	0,58	2.668
4,4,8,8	0,35	993
8,8,1,1	16,46	142.517
8,8,2,2	16,11	131.902
8,8,3,3	12,77	98.263
8,8,4,4	0,63	3.063
10,10,1,1	16,17	138.369
10,10,2,2	14,95	118.140
10,10,3,3	0,65	3.085
10,10,4,4	0,53	2.364
20,20,1,1	0,61	2.759
20,20,2,2	0,54	2.272
20,20,3,3	0,37	808
20,20,4,4	0,38	808

Propuesta de reparto de tejas

Con todos estos datos, ya se puede calcular una ratio de fase1/fase2 más realista para el modelo que se pretende construir, consistente en varias fases simultáneas (1 y 2), repartidas por distintas configuraciones de tejas. Nótese que se pretende encontrar una estrategia de asignación de tejas a dos procesos con distintos tiempos de ejecución.

Este es un típico problema de organización industrial como puede verse en “Essays on the Theory of Constraints” (Goldratt [44]). Así, se tiene un proceso P1 que es más lento (fase 1 de Smith-Waterman) que otro proceso P2 (fase 2 de Smith-Waterman) y en el que se trata de dimensionar los recursos (tejas) a asignar a ambos procesos, con objeto de minimizar los tiempos de procesamiento para la obtención de la puntuación (del inglés, “score”) y el alineamiento. Para adaptar los tiempos entre ambos procesos, se suele incluir un amortiguador de carga del proceso P1 al P2 (en este caso, el almacenamiento interno de la tarjeta TILExpress-20G) que no se debe desbordar. Hay que garantizar que la suma del número de fases 1 y 2 simultáneas no supere 10 en el caso de secuencias extremas de 300 kb, pues ya se comentó anteriormente que se aproxima a un alineamiento equivalente de 1 Mb x 1 Mb. Llegado este extremo, la tarjeta se puede bloquear, con lo que los procesos se pararían.

La solución a este problema consiste en igualar los ritmos de los flujos de ambos procesos, en la medida de lo posible. Ello puede conseguirse mediante la asignación de más recursos al proceso lento (fase 1), de forma que se aproxime a los tiempos del proceso rápido (fase 2), utilizando si es necesario un amortiguador o acumulador (utilizando terminología informática, un “buffer”) para compensar las posibles desviaciones puntuales que se puedan producir. De esta forma se minimizan los tiempos en los que hay recursos (tejas) sin utilizar.

Dado que el tiempo de fase 1 con geometría 7x2 es de 92 segundos (secuencias de 125 kb x 109 kb), y que el tiempo de fase 2 es de cerca de 17 segundos, en el caso de utilizar un esquema de puntuación que alargue la fase 2, se obtiene la siguiente ratio:

Ratio: Fase 1/Fase 2 = 92/17 = 5,4

Esto significa que durante el tiempo que tarda en ejecutarse una fase 1, se pueden ejecutar hasta 5,4 fases 2. O visto al revés, que se pueden tener 5,4 fases 1 simultáneas por cada fase 2, sin peligro de desbordamiento del buffer intermedio.

En el caso del presente trabajo, se dispone de 60 tejas libres (descontadas ya las cuatro tejas reservadas por la propia tarjeta) y la mejor geometría obtenida es la 7x2. Por tanto, se pueden ejecutar hasta cuatro fases 1 simultáneas, quedando cuatro tejas libres que se reparten en dos fases 2 (cada fase 2 consume dos tejas) y se podría tener otras cuatro fases 1 pendientes de ser procesadas por fase 2 (4 + 2 + 4 = 10). Dado que se ha partido de un escenario pesimista, es prácticamente imposible que el buffer se llene, pues habrá incluso fases 2 que no se ejecutarán cuando la puntuación obtenida durante la fase 1 no supere el umbral establecido por el usuario. Se ha podido comprobar que el llenado del buffer (que implica tiempos de espera de fases 1 hasta que existan recursos libres de fase 2) se da en bases de datos de secuencias cortas (2 a 6 kb), para las cuales la aplicación desarrollada es funcionalmente operativa pero no recomendada en términos de rendimiento.

La figura 6.12 muestra, mediante un esquema de colores, una posible ejecución simultánea de varias fases 1 y 2 con todas las tejas trabajando. Se muestran dos fases 2 con geometría 1x2 en amarillo y en bloques de dos tejas. En distintos tonos verdes, se muestran cuatro fases 1 con geometría 7x2.

0,0	1,0	2,0	3,0	4,0	5,0	6,0	7,0
0,1	1,1	2,1	3,1	4,1	5,1	6,1	7,1
0,2	1,2	2,2	3,2	4,2	5,2	6,2	7,2
0,3	1,3	2,3	3,3	4,3	5,3	6,3	7,3
0,4	1,4	2,4	3,4	4,4	5,4	6,4	7,4
0,5	1,5	2,5	3,5	4,5	5,5	6,5	7,5
0,6	1,6	2,6	3,6	4,6	5,6	6,6	7,6
0,7	1,7	2,7	3,7	4,7	5,7	6,7	7,7

Figura 6.12. Se indica en tonos verdes cada una de las ejecuciones paralelas de geometrías 2x8. En amarillo, dos fases 2 simultáneas.

Algoritmo principal

La aplicación desarrollada se ha denominado MC64-S3W [*MultiCore64-(Sequence Search Smith) Waterman*]. La figura 6.13 muestra el algoritmo principal de la aplicación en pseudocódigo. Se ha conseguido obtener un código relativamente sencillo gracias a la definición de un par de clases en Perl que encapsulan mucha de la funcionalidad necesaria para conseguir los objetivos propuestos. Una breve descripción de dichas clases se muestra a continuación:

- **`Tilera::TileMonitor`** es la clase principal y la que gestiona toda la interfaz de comunicación con el comando `tile-monitor`. Entre sus funciones están la gestión de los canales de entrada, salida y error estándar del comando, la gestión de puestos de trabajo (del inglés, “slots”) libres (conjunto de tejas bajo una misma geometría) y la gestión de trabajos en curso y acabados.

- **`Tilera::TileApp`** es la clase que encapsula toda la información de un trabajo o aplicación, que será ejecutada y gestionada posteriormente por un objeto de la clase `TileMonitor`. En este trabajo, cada fase del algoritmo Smith-Waterman (la fase de avance o fase 1 y la fase de retroceso o fase 2) es una aplicación independiente (una instancia distinta de la clase `TileApp`). Esta clase gestiona el número y orden de los parámetros que necesita la aplicación, obteniendo el mejor tamaño de *k* (parámetro definido por el algoritmo FastLSA) en función de los tamaños de las secuencias a procesar.

Con estos módulos en Perl, se facilita el camino para la reutilización de código y su aplicación a otros algoritmos bioinformáticos basados en la tarjeta TILExpress-20G, como puede ser el caso de ClustalW (que permite el alineamiento rápido de múltiples secuencias).

```
# Algoritmo principal aplicación MC64-S3W

controlar argumentos de la aplicación;

instanciar objeto tile-monitor y configurar slots;

calcular tamaño base de datos;

abrir base de datos;

iniciar control de tiempo;

mientras (1) {
   mientras (haya slots libres para fase 1) {
      si (hay más secuencias por procesar) {
         leer próxima secuencia;
         generar fichero fasta;
         subir fichero a tarjeta;
         preparar trabajo de fase 1;
         lanzar trabajo fase 1;
      } si no {
         si (hay trabajos en curso) {
            imprimir información de trabajos en curso;
         } si no {
            salir del bucle principal;
         }
      }
   }

   leer canal de salida de tile-monitor;

   mientras (haya trabajos de fase 2 esperando) {
      descargar ficheros en directorio de salida: alineamiento e información;
      borrar ficheros de la tarjeta;
      guardar información para informe;
   }

   si (hay trabajos de fase 1 esperando) {
      si (score >= threshold) {
         si (hay slots libres para fase 2) {
            preparar trabajo de fase 2;
            lanzar trabajo fase 2;
         } si no {
            devolver trabajo a la lista de trabajos pendientes de fase 1;
         }
      } si no {
         borrar ficheros temporales y secuencia subject;
      }
   }
}

generar cabecera informe;

procesar secuencias que pasan el umbral según score y añadir información a informe;

salir;
```

Figura 6.13. Resumen del algoritmo principal de la aplicación en pseudocódigo.

Detalles de la implementación

Algunos aspectos a destacar de la implementación realizada son:

- Utilización de la biblioteca BioPerl para procesar ficheros FASTA multientrada y generación de ficheros de secuencia individual (figura 6.14).

```
# Gets size of query sequence for k estimation
my $db_temp = Bio::SeqIO->new(-file => $query_seq, -format => 'Fasta');
my $seq = $db_temp->next_seq();
my $query_seq_length = $seq->length;
my $query_seq_header = $seq->id() . " " . $seq->desc();
print "Query sequence file  : $query_seq\n";
print "Query sequence header: $query_seq_header\n";
print "Query sequence length: $query_seq_length\n";
```

Figura 6.14. Uso de la biblioteca BioPerl.

- Generación de módulos Perl para encapsular la funcionalidad del algoritmo y facilitar la reutilización de código en otros proyectos basados en la tarjeta TILExpress-20G (figura 6.15).

```
# Creates tile_monitor object
my $tile_monitor = Tilera::TileMonitor->new();
$tile_monitor->start('tile-monitor --pci --bootrom-file '.MC64_S3W_PATH.'/boot/vmlinux-
pci-Geometry4.bootrom');
$tile_monitor->mkdir('s3w');
$tile_monitor->upload(MC64_S3W_PATH.'/env', 's3w');
$tile_monitor->cd('s3w');
$query_seq_new = "000000.fa";
$tile_monitor->upload($query_seq, $query_seq_new);
```

Figura 6.15. Generación de módulos Perl.

- Lanzamiento en paralelo de varios trabajos mediante segmentación de tejas, utilizando el concepto de geometría del chip Tile64 (figura 6.16).

```
# Setup tile_monitor slots
$tile_monitor->put_slot(STAGE_1, "7x2\@1,0");
$tile_monitor->put_slot(STAGE_1, "7x2\@1,2");
$tile_monitor->put_slot(STAGE_1, "7x2\@1,4");
$tile_monitor->put_slot(STAGE_1, "7x2\@1,6");
$tile_monitor->put_slot(STAGE_2, "1x2\@0,0");
$tile_monitor->put_slot(STAGE_2, "1x2\@0,6");
```

Figura 6.16. Uso del concepto de geometría para el lanzamiento de varios trabajos en paralelo.

- Lectura no bloqueante de los canales de entrada, salida y error estándar de un proceso Unix desde Perl. Si no hay datos para leer al transcurrir un plazo determinado (del inglés, “timeout”) el algoritmo sigue su curso (figuras 6.17 y 6.18).

```
# Run command for reading, writint and error handling
$_child_pid = open3(\*WRITER, \*READER, \*ERROR, $self->cmd_line()) or die
"[TileMonitor->start] - Cannot run command\n";
# Create SELECT object with command file handlers
$_select = new IO::Select();
$_select->add(\*READER);
$_select->add(\*WRITER);
$_select->add(\*ERROR);
...
```

Figura 6.17. Creación de los manejadores (del inglés, "handlers") asociados al proceso tile-monitor.

```
...
# Get handles if data ready for reading
while (my @data_ready = $_select->can_read($timeout)) {
  foreach $fh (@data_ready) {
    $bytes_read = sysread($fh, $data, READ_SIZE);

    if (fileno($fh) == fileno(ERROR)) {
      $ret{'ERROR'} = $data;
    } elsif (fileno($fh) == fileno(WRITER)) {
        $ret{'WRITER'} = $data;
    } elsif (fileno($fh) == fileno(READER)) {
        $ret{'READER'} = $data;
    }
  }
}
...
```

Figura 6.18. Lectura no bloqueante de los canales de entrada, salida y error estándar del proceso tile-monitor.

- Utilización de expresiones regulares para la interpretación de mensajes de salida estándar y error estándar del proceso `tile-monitor` (figura 6.19).

```
if ($line =~ /.*DONE1 (\d+) (\S+)$/) {
  # "DONE1" output detected, so update state of job (type stage 1)
  debug(2, "_parse_data", ">>> DONE1 $1 $2 ".$_jobs{$2}->geometry()." score: ".$1);
  $_jobs{$2}->score($1);
  $_jobs{$2}->state("waiting");
  $self->free_slot($_jobs{$2});
  $_jobs{$2}->geometry("");

  $line =~ s/DONE1 \d+ \S+//g;
}
```

Figura 6.19. Utilización de expresiones regulares.

- Estructuras avanzadas de datos para el control de puestos (“slots”) y trabajos (figura 6.20).

```
# Use case: get_job("waiting", STAGE_1), get_job("running", STAGE_2)
sub get_job {
  my $self = shift;
  my $state = shift;
  my $type = shift;

  # Finds and returns the firts job with such a state and type, if any
  foreach my $job (values %_jobs) {
    if ($job->state() eq $state) {
      if (defined $type) {
        if ($job->type eq $type) {
          delete $_jobs{$job->id()};
          return $job;
        }
      } else {
        delete $_jobs{$job->id()};
        return $job;
      }
    }
  }
  return 0;
}
```

Figura 6.20. Uso de estructuras avanzadas de datos.

Se ha diseñado un servicio Web para la utilización de la herramienta de búsqueda desarrollada en este proyecto desde Internet, disponible en <http://www.sicuma.uma.es/manycore/index.jsp?seccion=m64sssw/run>

La figura 6.21 muestra el formulario de búsqueda.

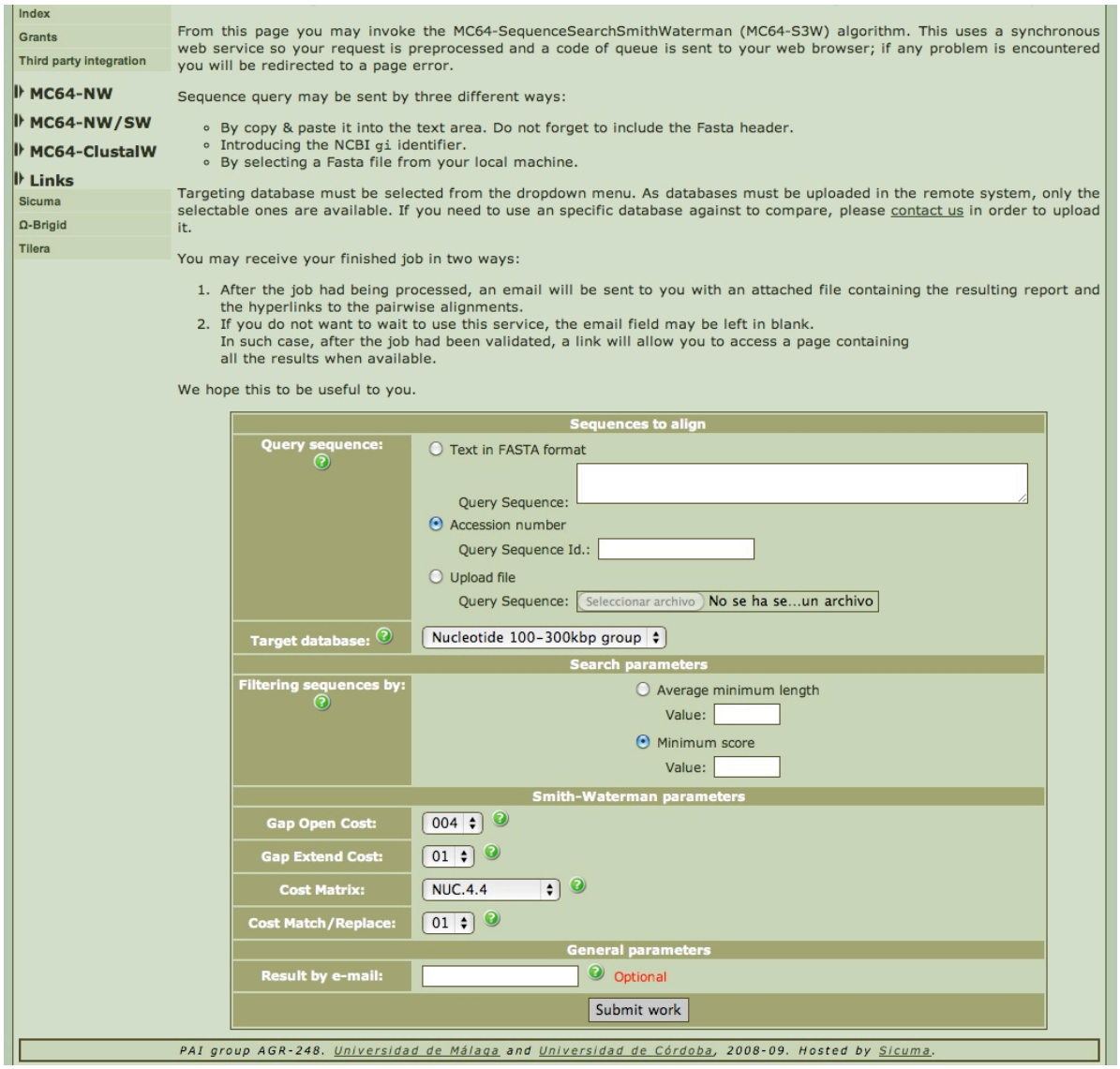


Figura 6.21. Formulario de búsqueda del servicio Web asociado a MC64-S3W.

Como se puede apreciar en la figura 6.22, los resultados que se muestran una vez finalizada la búsqueda constan de una cabecera y un listado de secuencias. La cabecera muestra datos generales de la búsqueda como son:

- Identificación y longitud de la secuencia interrogante.
- Base de datos donde se ha llevado a cabo la búsqueda, con indicación de su tamaño y el número total de entradas que contiene.
- Valor del umbral de corte.
- Matriz de pesos utilizada.
- Esquema de penalización por apertura y extensión de huecos.
- Tiempo total de la búsqueda y obtención de alineamientos.

- Total de secuencias de la base de datos que han superado el valor de corte (umbral).

Figura 6.22. Ejemplo de resultado de una búsqueda con el algoritmo MC64-S3W.

Tras la información de cabecera, se muestra un listado con información de las secuencias que han superado el umbral, en orden descendente por el valor de puntuación obtenida. Se detalla la siguiente información para cada secuencia:

- Identificación y longitud de la secuencia.
- Longitud del alineamiento óptimo para dicha secuencia.
- Puntuación máxima obtenida.
- Número y porcentaje de identidades dentro del alineamiento óptimo.
- Número y porcentaje de huecos dentro del alineamiento óptimo.

- Posiciones de inicio y fin del alineamiento en la secuencia interrogante.
- Posiciones de inicio y fin del alineamiento en la secuencia encontrada.
- Valores de *k* utilizados (necesarios para el algoritmo FastLSA).
- Tiempo de la fase 1 y tiempo total (ambos en segundos) para la obtención del alineamiento (fase 1 + fase 2).

A continuación se incluyen algunas capturas de pantalla de la salida que produce la aplicación MC64-S3W. La figura 6.23 muestra el inicio de la ejecución de la aplicación, en la que se incluye un resumen de los parámetros más importantes de la aplicación (rutas principales de la aplicación y de los ficheros de base de datos, umbral, esquema de puntuación, base de datos con indicación de su tamaño y número de entradas, así como secuencia interrogante con indicación de su tamaño).

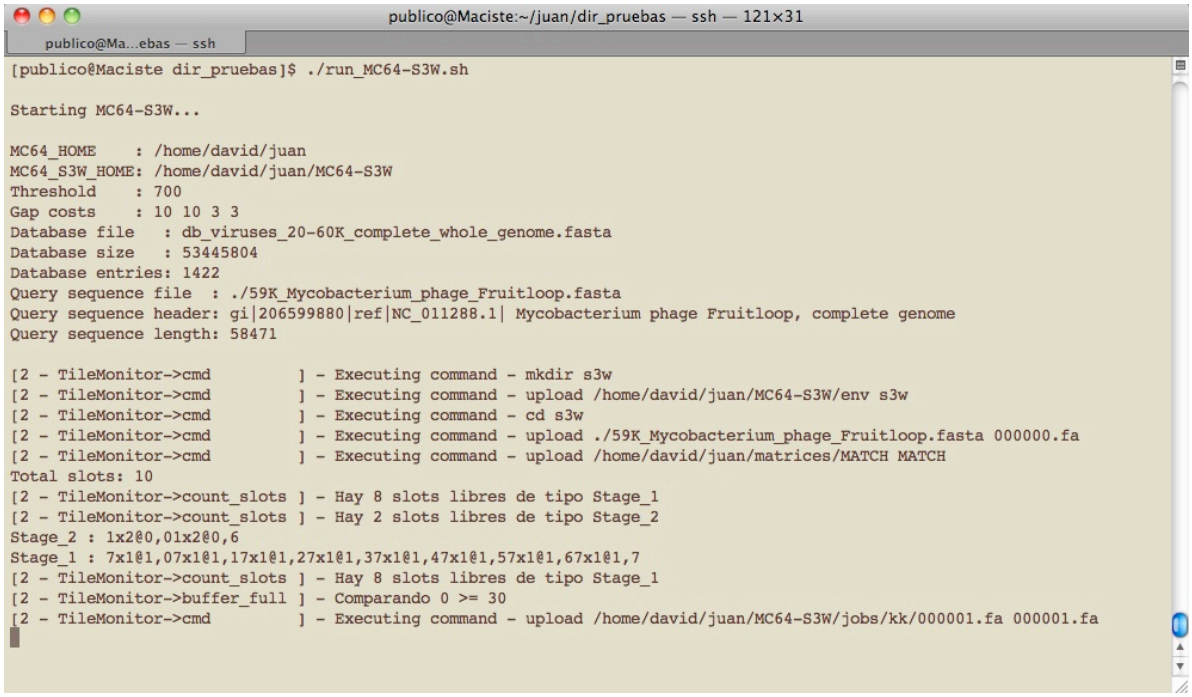


Figura 6.23. Salida por consola producida por el algoritmo MC64-S3W al inicio de su ejecución.

La figura 6.24 muestra otro ejemplo de la salida producida por el algoritmo MC64-S3W, en la que se puede apreciar un momento del algoritmo principal en el que finalizan varias fases 1, se comprueba la puntuación obtenida (en este caso no supera el umbral) y lanzamiento de nuevas fases 1.

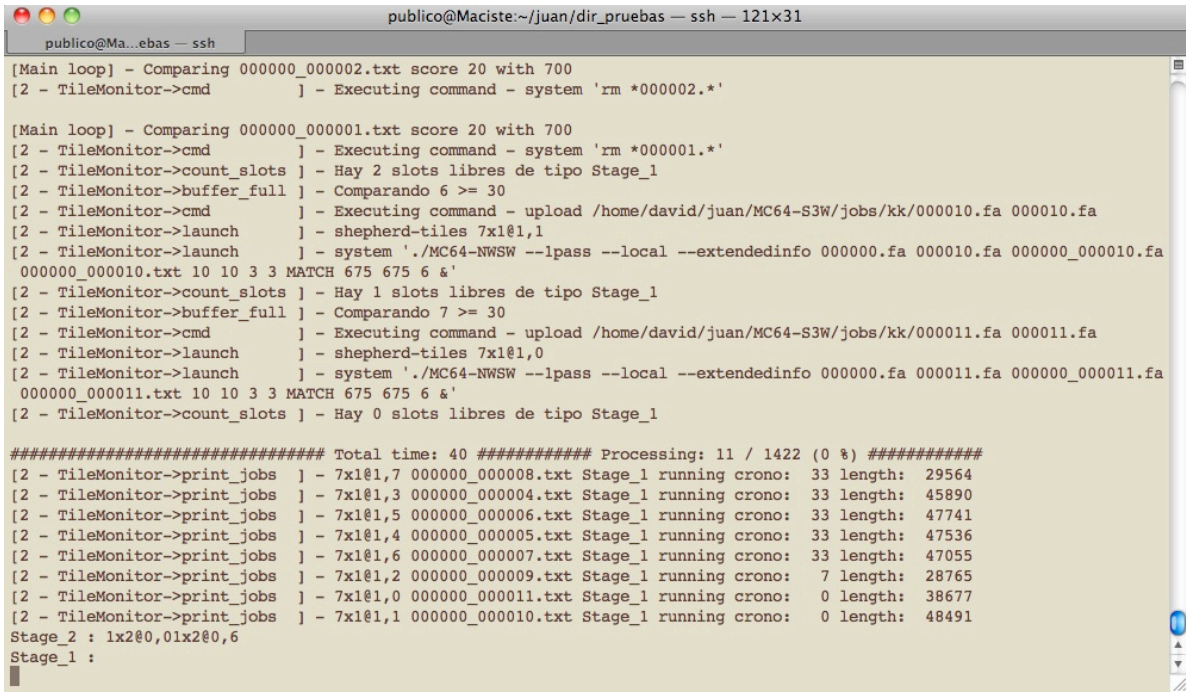


Figura 6.24. Salida por consola producida por MC64-S3W en una fase intermedia de la ejecución, con finalización y lanzamiento de nuevas fases 1.

La figura 6.25 muestra el principio del fichero de informe generado. Tras la información de cabecera, se muestra el detalle de las secuencias que han superado el umbral ordenadas en orden decreciente según la puntuación obtenida.

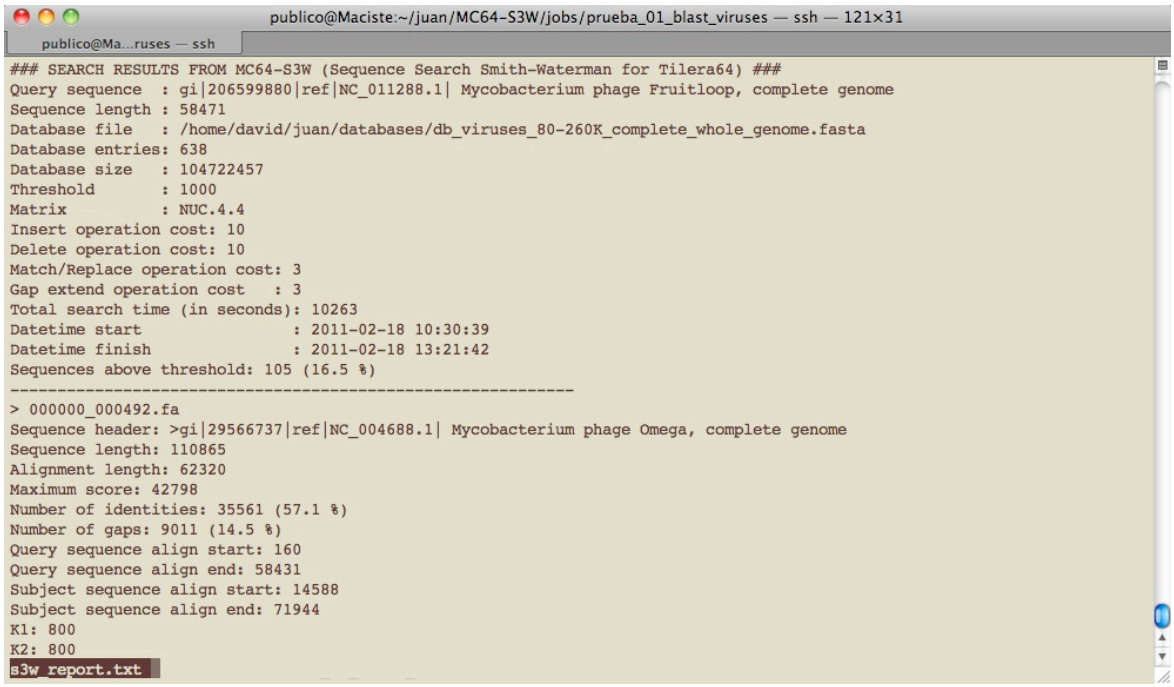


Figura 6.25. Informe generado por el algoritmo MC64-S3W.

Y finalmente, la figura 6.26 muestra un ejemplo de fichero del alineamiento obtenido.

Figura 6.26. Ejemplo de resultado de un alineamiento tras una búsqueda con el algoritmo MC64-S3W.

Bases de datos de ácidos nucleicos usadas

Se han usado secuencias de gran tamaño de ácidos nucleicos procedentes del NCBI, empleando la herramienta "ebot" también del NCBI, disponible en Internet <http://www.ncbi.nlm.nih.gov/Class/PowerTools/eutils/ebot/ebot.cgi>.

Se trata de una herramienta interactiva (un asistente Web) que genera una script Perl que permite ejecutar una consulta contra las bases de datos del NCBI y almacenar el resultado de la consulta en un fichero FASTA multientrada en el sistema de ficheros (del inglés, "filesystem") local del equipo desde el que es invocada la script. De esta forma, se han obtenido las siguientes colecciones de secuencias indicadas en la tabla 6.7:

Tabla 6.7. Bases de datos generadas a partir de "ebot" y "NCBI Nucleotide Database".

Base de datos	Número secuencias
db_dsdna_80-260K_complete_whole_genome.fasta	295
db_ecoli_20-60K_complete_whole_genome.fasta	5.252
db_ecoli_80-260K_complete_whole_genome.fasta	2.053
db_mus_musculus_80-260K_complete_whole_genome.fasta	4.618
db_saccharomyces_80-260K_complete_whole_genome.fasta	234
db_ssdna_80-260K_complete_whole_genome.fasta	352

Base de datos	Número secuencias
db_viridiplantae_80-260K_complete_whole_genome.fasta	3.896
db_viruses_20-60K_complete_whole_genome.fasta	1.422
db_viruses_80-260K_complete_whole_genome.fasta	638

Las bases de datos de tamaño 20 a 60 kb han sido generadas para las pruebas de comparación con CUDASW++ y su justificación se verá en el siguiente apartado.

Comparación con CUDASW++

El entorno utilizado para las pruebas y comparación con CUDASW++ ha sido el siguiente:

- CUDASW++ (versión 2.0.5).
- Servidor equipado con hasta cuatro tarjetas con soporte CUDA: 2 tarjetas GeForce 9800 GX2 (compatibles con versiones 1.1 de CUDA) y dos tarjetas Tesla S2050 (compatibles con versiones 2.0 de CUDA). Sólo se ha utilizado las tarjetas Tesla por su compatibilidad con la versión 2.0 de CUDASW++, pues la versión de CUDA utilizada sólo puede trabajar a partir de la versión 1.2 o superior.

El algoritmo CUDASW++ tiene algunas limitaciones de partida indicadas en su fichero de instrucciones (fichero “ReadMe”), como son que la secuencia problema no puede ser mayor de 59 kb y que el límite de las longitudes de las secuencias sujeto (del inglés, “subject”) de la base de datos es modificable mediante la macroinstrucción (macro) denominada **MAX_SEQ_LENGTH**, que por defecto está asignada a 256 kb.

Otra limitación adicional es la imposibilidad de trabajar con secuencias que produzcan una puntuación por encima de **65.535 (2^{16}-1)**. Ello es debido a que el algoritmo utiliza enteros de **16 bits** para almacenar el valor máximo de la puntuación. En tales casos, se ha comprobado que el algoritmo da error (no se ha encontrado referencias en la documentación sobre esta restricción).

Esto ha obligado a dos actuaciones: i) utilizar forzosamente un esquema de puntuación de tipo MATCH, que puntúa positiva y negativamente las identidades y diferencias, respectivamente (del inglés, “match +1, mismatch −1”); y ii) buscar secuencias de menor tamaño, tanto para la secuencia interrogante como para las de la base de datos de búsqueda (secuencias sujeto). Por tanto, las bases de datos iniciales con secuencias de longitudes comprendidas entre 80 y 260 kb fueron sustituidas por otras entre 20 y 60 kb. Se ha

descartado incluir secuencias menores de 20 kb por requerir menos recursos de computación (no tiene sentido utilizar la arquitectura ofrecida por Tilera en estos casos).

Se han realizado pruebas con dos bases de datos con distinta distribución de longitudes de secuencias. Las tablas 6.8 y 6.9 muestran información de dichas bases de datos junto con sus distribuciones y el total de secuencias que contienen:

Tabla 6.8. Distribución de la base de datos 1 (virus).

BD 1: db_viruses_20-60K_complete_whole_genome.fasta		
Longitud de las secuencias (bases)	**Número de secuencias**	**Porcentaje (%)**
20.000 - 25.000	26	1,9
25.001 - 30.000	387	27,2
30.001 - 35.000	220	15,5
35.001 - 40.000	323	22,7
40.001 - 45.000	260	18,3
45.001 - 50.000	97	6,8
50.001 - 55.000	52	3,7
55.001 - 60.000	57	4
Total secuencias	1.422	

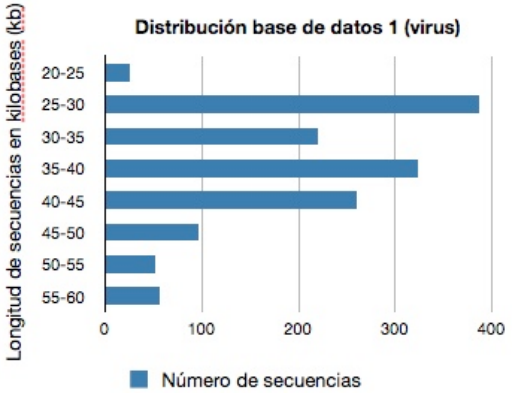
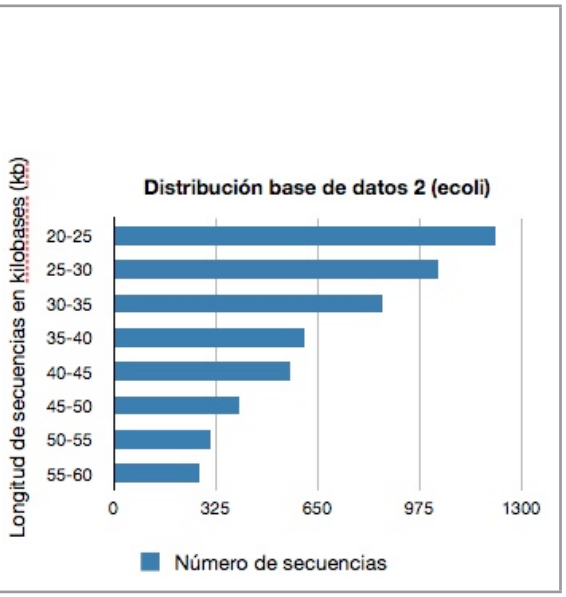


Tabla 6.9. Distribución de la base de datos 2 (*Escherichia Coli*).

BD 1: db_ecoli_20-60K_complete_whole_genome.fasta		
Longitud de las secuencias (bases)	**Número de secuencias**	**Porcentaje (%)**
20.000 - 25.000	1.216	23,2
25.001 - 30.000	1.033	19,7
30.001 - 35.000	856	16,3
35.001 - 40.000	609	11,6
40.001 - 45.000	559	10,6
45.001 - 50.000	397	7,6
50.001 - 55.000	308	5,9
55.001 - 60.000	274	5,2
Total secuencias	5.252	



Para la comparación con CUDASW++ se han utilizado geometrías de tejas de 7x1 con la aplicación de MC64-S3W en Tile64, pues mediciones previas han demostrado que es la configuración más adecuada teniendo en cuenta las distribuciones de secuencias de ambas

bases de datos. Esto permite paralelizar hasta ocho fases 1 simultáneamente, junto con dos fases 2. Al ser secuencias de menor tamaño del previsto inicialmente, se ha incrementado igualmente el buffer interno hasta 30. No obstante, se ha podido comprobar que con la distribución de tejas comentada y los tiempos analizados, se puede trabajar con tamaños de buffer bastante menores (en cualquier caso, ello no afecta al rendimiento obtenido, siendo sólo una precaución para evitar desbordamientos de memoria). La tabla 6.10 muestra un resumen de los tiempos obtenidos:

Tabla 6.10. Resultados comparación CUDASW++ con MC64-S3W. Tiempos de ejecución medidos en segundos y relación entre tarjeta Tilera TILExpress-20G y NVidia Tesla.

Longitudes de secuencias	CUDASW++ 1 x Tesla S2050	CUDASW++ 2 x Tesla S2050	MC64-S3W 1 x TILExpress-20G	Ratio TILExpress / Tesla
Virus (20 a 60 kb)	1.014	524	6.167	6,08
Bacterias (20 a 60 kb)	3.420	1.727	20.970	6,13

Se puede comprobar como CUDASW++ es capaz de aprovechar tarjetas adicionales para aumentar su rendimiento (prácticamente al doble con una segunda tarjeta adicional). Dado que los recursos hardware de una tarjeta Tesla supera a los de una tarjeta TILExpress-20G, se va a centrar la comparativa teniendo en cuenta los tiempos obtenidos con una única tarjeta Tesla. En estas circunstancias, se puede comprobar que el algoritmo CUDASW++ es hasta **seis veces** más rápido que MC64-S3W pero con las siguientes salvedades:

1) Los recursos de computación de una tarjeta Tesla S2050 superan varias veces el suministrado por el microprocesador Tile64. Una tarjeta Tesla S2050 cuenta con 14 microprocesadores con 32 núcleos cada uno, haciendo un total de 448 núcleos frente a los 64 núcleos de microprocesador Tile64; es decir, **siete veces** más el número total de núcleos. Además, la frecuencia de reloj a la que funcionan los núcleos de la tarjeta Tesla es de 1,15 GHz, frente a los 866 MHz de Tile64.

2) Como ya se ha mencionado, sólo se ha podido utilizar el algoritmo CUDASW++ con tamaños de secuencias que no superen una puntuación de 65.535, sin posibilidad de utilizar otro tipo de matrices de puntuación como por ejemplo NUC.4.4., dada la limitación que supone usar 16 bits para almacenar el valor de la puntuación.

3) CUDASW++ **genera la puntuación, pero no los alineamientos**.

En cuanto a la calidad de los resultados, como ambas aplicaciones se basan en el algoritmo Smith-Waterman (y la matriz de puntuación MATCH), generan las mismas puntuaciones máximas. La figura 6.27 muestra las 10 mejores secuencias con sus puntuaciones.

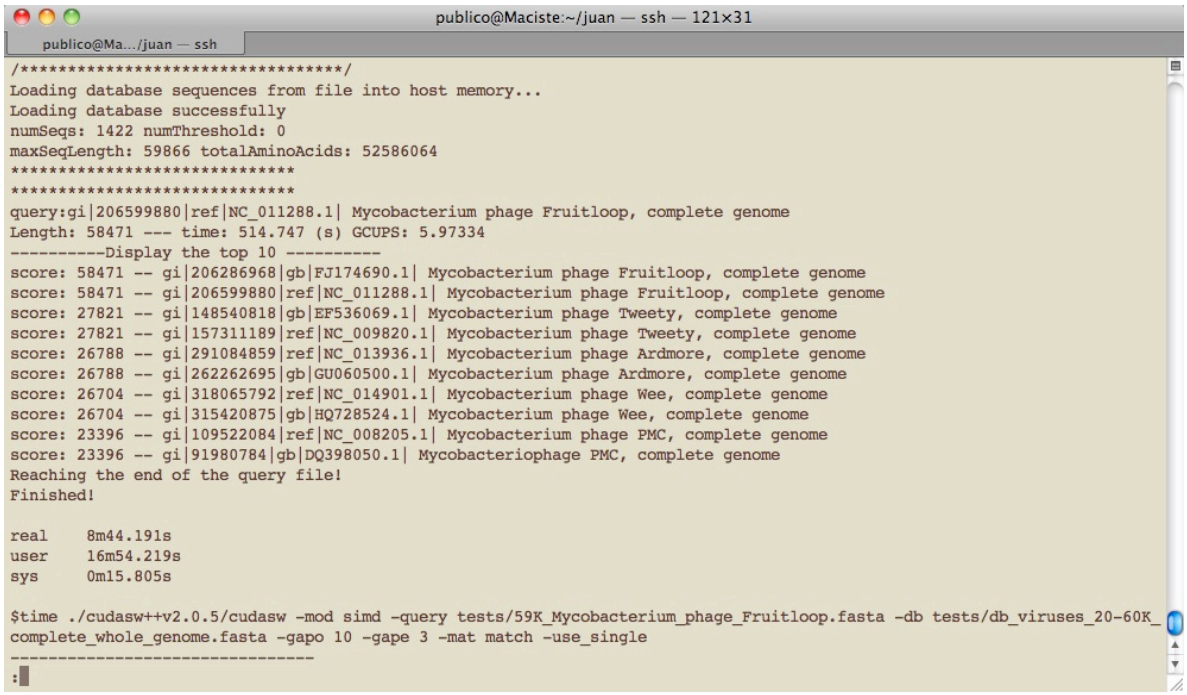


Figura 6.27. Algunos resultados de CUDASW++ para la base de datos db_viruses_20-60K.

Comparación con BLAST

El entorno utilizado para las pruebas y comparación con BLAST ha sido el siguiente:

- BLAST (versión ncbi-blast-2.2.24+).
- Quad Core Intel Xeon 2.0 GHz con 8 GB de memoria RAM DDR2 y con sistema operativo CentOS 5 y versión de kernel 2.6.18-128.el5. Este servidor tiene instalada una tarjeta TILExpress-20G.

Un primer intento de ejecutar BLAST bajo entorno Windows generó problemas al intentar indexar una de las base de datos (en concreto la de virus). Los tiempos de indexación de las bases de datos se muestran en la tabla 6.11:

Tabla 6.11. Tiempos de indexación de BLAST para las bases de datos analizadas.

Base de datos	Tiempo indexación (segundos)
db_viruses_80-260K_complete_whole_genome.fasta	2,6
db_ecoli_80-260K_complete_whole_genome.fasta	8,7

En este caso, y a diferencia de CUDASW++, no se han encontrado limitaciones para trabajar con los tamaños de secuencias entre 80 y 260 kb.

Como era de esperar, los tiempos de ejecución de BLAST son muy inferiores que los obtenidos con CUDASW++ (teniendo en cuenta además que en este caso se trabaja con tamaños de secuencias que CUDASW++ no puede procesar) y, por tanto, inferiores también a los obtenidos con la tarjeta Tilera, como se puede comprobar en la tabla 6.12.

Tabla 6.12. Tiempos de ejecución de BLAST para las bases de datos analizadas. El valor e (del inglés, "e-value") permite valorar el significado estadístico de los resultados obtenidos. Cuanto menor sea este valor, más significativo es un alineamiento.

Búsqueda	Búsqueda	Tiempo búsqueda y resultados BLAST	Tiempo búsqueda y resultados MC64-S3W
1	59K_Mycobacterium_phage_Fruitloop.fasta contra db_viruses_80-260K_complete_whole_genome.fasta	0,17 segundos 4 resultados (valor e = 1e-120)	11.030 segundos 7 resultados (umbral = 7.620)
2	59K_Mycobacterium_phage_Fruitloop.fasta contra db_viruses_80-260K_complete_whole_genome.fasta	0,19 segundos 16 resultados (valor e = 0.001)	10.263 segundos 105 resultados (umbral = 1.000)
3	100K_monkeypox_virus_strain.fasta contra db_viruses_80-260K_complete_whole_genome.fasta	14,7 segundos 120 resultados (valor e = undef)	35.534 segundos 250 resultados (umbral = 5.000)
4	50K_Escherichia_coli_RN587.fasta contra db_ecoli_80-260K_complete_whole_genome.fasta	3,1 segundos 1.338 resultados (valor e = undef)	32.862 segundos 37 resultados (umbral = 120.000)

Hay que indicar que el comando BLAST ha sido ejecutado con los parámetros por defecto. A continuación se muestra un ejemplo de ejecución:

```
BLAST: blastn -query 59K_Mycobacterium_phage_Fruitloop.fasta -db
db_viruses_80-260K_complete_whole_genome.fasta -evalue 1e-120 -
num_threads 4 -out sal.txt
```

El número de hilos (threads) indicado es cuatro, con objeto de aprovechar el paralelismo de los núcleos del servidor (Quad Core Intel Xeon).

En cuanto a los resultados obtenidos, se debe mencionar que se han encontrado diferencias entre ambas aplicaciones. Así, por ejemplo, para la búsqueda 1 definida según la tabla 6.2, ambos algoritmos sólo coinciden en la secuencia con mayor puntuación, siendo diferentes las siguientes tres secuencias obtenidas por cada algoritmo según orden de puntuación decreciente (véase la tabla 6.13).

Tabla 6.13. Diferencias de resultados entre BLAST y MC64-S3W ordenadas de mayor a menor (de arriba abajo) puntuación por cada algoritmo tras la búsqueda 1.

BLAST	MC64-S3W
NC_004688.1| Mycobacterium phage Omega	NC_004688.1| Mycobacterium phage Omega
GQ303262.1| Mycobacterium phage LRRHood	NC_011273.1| Mycobacterium phage Myrna
NC_011269.1| Mycobacterium phage ScottMcG	EU826466.1| Mycobacterium phage Myrna
EU826469.1| Mycobacterium phage ScottMcG	AY261359.1| Bovine herpesvirus 5 strain SV507/99

Además, hay que indicar que BLAST no genera un alineamiento óptimo completo (aunque ambos algoritmos obtienen la misma secuencia con mayor puntuación), mostrando distintas porciones del alineamiento entre las cadenas directa o positiva y reversa o negativa (del inglés, strand=plus, strand=minus), como así se ha podido comprobar en la salida generada de las distintas búsquedas.

La búsqueda 2 es similar a la 1, pero relajando los límites del valor e (del inglés, "e-value") y umbral, para favorecer la generación de más resultados.

Analizando la búsqueda 3, igualmente se pueden apreciar diferencias en los resultados obtenidos por BLAST y MC64-S3W (véase la tabla 6.14). Las 2 primeras secuencias coinciden en ambos algoritmos (verde), la tercera y cuarta (naranja) están intercambiadas con las posiciones quinta y sexta en MC64-S3W. De forma análoga ocurre con las secuencias en las posiciones quinta y sexta (azul):

Tabla 6.14. Diferencias de resultados entre BLAST y MC64-S3W ordenadas de mayor a menor (de arriba abajo) puntuación por cada algoritmo tras la búsqueda 3.

BLAST	MC64-S3W
HQ857563.1| Monkeypox virus strain D14L knockou...	HQ857563.1| Monkeypox virus strain D14L knockout
HQ857562.1| Monkeypox virus strain V79-I-005	HQ857562.1| Monkeypox virus strain V79-I-005
DQ011155.1| Monkeypox virus strain Zaire_1979-00...	NC_003310.1| Monkeypox virus Zaire-96-I-16
DQ011154.1| Monkeypox virus strain Congo_2003_35...	AF380138.1| Monkeypox virus strain Zaire-96-I-16
NC_003310.1| Monkeypox virus Zaire-96-I-16	DQ011155.1| Monkeypox virus strain Zaire_1979-005
AF380138.1| Monkeypox virus strain Zaire-96-I-16...	DQ011154.1| Monkeypox virus strain Congo_2003_35...

Estos resultados son lógicos, ya que BLAST es un heurístico [10], como se apuntó en la introducción de este trabajo.

Trabajos futuros

7

Dado que CUDASW++ no genera alineamientos, resulta interesante un proyecto de trabajo conjunto entre ambas aplicaciones (CUDASW++ y MC64-S3W), centrándose la primera en obtener las mejores puntuaciones, y la segunda en producir los alineamientos a partir de dichas puntuaciones, para secuencias de longitud menor o igual a 60 kb. Además de esto, el presente trabajo podría tener continuidad en los siguientes aspectos:

- Optimización del modelo de paralelismo. Clásicamente, cada fase 1 utiliza una teja como proceso controlador y el resto de tejas asignadas como trabajadores (del inglés, "workers"). Un paso más consiste en el diseño de un controlador inteligente que reparta de forma dinámica el trabajo de varios alineamientos distintos en implementaciones de trabajadores más genéricos.

- Cálculo y utilización de geometrías de tejas adaptativas en función de los tamaños de la secuencia interrogante y de la secuencia sujeto. Añade la complejidad de intentar evitar que se desaprovechen tejas, fruto de geometrías que dejan huecos (tejas sin utilizar), así como también gestionar de forma dinámica el tamaño del buffer entre la fase 1 y 2, con objeto de evitar problemas de desbordamiento de memoria.

- Se he podido comprobar de forma empírica que variando los valores de *k* se obtienen algunas mejoras de rendimiento en determinados casos, como así se indica en la tabla 7.1.

Tabla 7.1. Pruebas con distintos factores de *k* y tiempo obtenido para búsquedas en una base de datos de reducidas entradas. Nótese cómo con *k**0,8 se consigue un tiempo menor que *k*.

Factor *k*	Tiempo de ejecución (segundos)
k	665
k * 1,5	700
k * 1,25	681
k * 0,8	**658**
k * 0,7	685

- Adaptación del algoritmo para poder trabajar con varias tarjetas TILExpress-20G, estén o no en el mismo servidor, con objeto de aprovechar los recursos de computación disponibles.

- Aplicación a alineamientos múltiples, como es el caso de la primera fase del algoritmo ClustalW.

- Dada la calidad de los resultados obtenidos en cuanto a rendimiento y capacidad de proporcionar el alineamiento local completo, se está preparando un artículo con la idea de presentarlo en uno de los próximos congresos internacionales sobre Bioinformática. Concretamente, la intención es presentar una comunicación en el "11th Workshop on Algorithms in Bioinformatics" (WABI'11), a celebrar los días 5 a 7 de septiembre en Saarbrücken (Alemania). Dicho congreso tiene clasificación B según Citeseer <http://citeseer.ist.psu.edu/cs>.

Conclusiones 8

Los objetivos planteados en este proyecto abarcan la extensión de la programación paralela del algoritmo Smith-Waterman al caso de búsquedas en bases de datos. Los principales resultados obtenidos durante este trabajo son los siguientes:

1. Desarrollo del código fuente en lenguaje Perl del algoritmo MC64-S3W para búsquedas en bases de datos de secuencias de ácidos nucleicos, utilizando una implementación paralela del algoritmo Smith-Waterman.

2. Reforzamiento del concepto de paralelismo y aprendizaje de arquitectura de muchos núcleos, demostrando su utilidad para el análisis de secuencias en bioinformática. Tal es el caso del microprocesador Tile64, que forma parte de la tarjeta TILExpress-20G fabricada por Tilera.

3. Profundización en diversos algoritmos y técnicas de alineamiento de secuencias mediante revisión bibliográfica y análisis de multitud de artículos y publicaciones científicas. En esta misma línea, análisis de los fundamentos biológicos para la comparación de secuencias de gran tamaño.

4. El desarrollo realizado en el presente proyecto tiene especial utilidad en aquellos estudios en los que se necesita encontrar los alineamientos óptimos de una secuencia problema contra una base de datos, en el caso particular de secuencias de gran tamaño (entre 80 y 260 kb). Dadas estas dimensiones, tiene más sentido su aplicación en el ámbito de secuencias de ácidos nucleicos que de péptidos (proteínas).

5. Las aplicaciones basadas en el algoritmo de programación dinámica, como es el caso de CUDASW++, suelen tener dificultadas para trabajar con secuencias muy largas (como las analizadas en este trabajo entre 80 y 260 kb). Sin embargo, CUDASW++ sí es muy eficiente en la obtención de las mejores puntuaciones en secuencias de menos de 60 kb. Con este trabajo se ha demostrado la posibilidad, no solo de conseguir las mejores puntuaciones, sino también de obtener el alineamiento local óptimo en búsquedas de bases de datos usando programación dinámica.

6. El rendimiento conseguido para la búsqueda de secuencias de gran tamaño (entre 80 y 260 kb) con esta implementación del algoritmo Smith-Waterman en un ordenador personal ha sido comparativamente satisfactorio. Con un número de núcleos siete veces inferior al de una tarjeta Tesla S2050 se ha conseguido una búsqueda en base de datos sólo seis veces más lenta que con el algoritmo CUDASW++.

7. Al igual que MC64-S3W, BLAST puede emplearse con secuencias de gran tamaño, generando resultados rápidamente. Sin embargo, al estar basado en un algoritmo heurístico, los resultados obtenidos pueden no ser óptimos, como así se ha podido comprobar en las

comparativas realizadas entre ambos algoritmos. La tendencia actual es usar BLAST para realizar búsquedas en bases de datos con objeto de acotar las secuencias candidatas, para luego refinar el resultado mediante la aplicación de algoritmos como el caso de Smith-Waterman.

Agradecimientos

Agradezco en primer lugar a la Fundación IAVANTE su apoyo, tanto económico como de tiempo, puesto que ello ha supuesto un sólido respaldo para la realización de este Máster.

Igualmente, agradezco a Tilera Corporation <http://www.tilera.com> el haber proporcionado tarjetas TILExpress-20G y software MDE, que han permitido desarrollar este trabajo. Asimismo, a David Díaz González y demás miembros del Grupo PAIDI AGR-248 (Biotecnología Agroalimentaria) de la Universidad de Córdoba, al haberme facilitado el acceso a dicho hardware y software, y prestado su colaboración durante el desarrollo (todo ello en el marco del proyecto nacional AGL2006-12550-C02-01, así como 041/C/2007, 75/C/2009 y 56/C/2010 de la Consejería de Agricultura y Pesca de la Junta de Andalucía).

También expreso mi agradecimiento a los miembros del Departamento de Arquitectura de Computadores de la Universidad de Málaga por el acceso al computador "yuca" financiado con los proyectos TIN2006-01078 y P08-TIC-3500, así como a las GPU financiadas con el proyecto PR06-TIC-2109, para realizar la comparativa con CUDASW++.

Estoy especialmente agradecido a Sergio Gálvez Rojas por el tiempo y la minuciosidad dedicados en las diversas revisiones del presente trabajo, sobre todo teniendo en cuenta el grado de ocupación profesional que ha tenido durante el desarrollo de este trabajo fin de Máster.

Mi agradecimiento también a los co-tutores de este trabajo por sus múltiples aportaciones y comentarios.

Bibliografía

Las referencias bibliográficas están indicadas en estilo IEEE.

[1] Díaz, D., Gálvez, S., Falqueras, J., Caballero, J. A., Claros, G. and Dorado, G. Intuitive bioinformatics for genomics applications: Omega-Brigid workflow framework. "*Lecture Notes in Computer Science (IWANN 2009) 5518: 1084-1091*".

[2] Huang, X. and Madan, A. CAP3: A DNA sequence assembly program. "*Genome Res*", 9, 9 (Sep 1999), 868-877.

[3] Thompson, J. D., Higgins, D. G. and Gibson, T. J. CLUSTAL W: improving the sensitivity of progressive multiple sequence alignment through sequence weighting, position-specific gap penalties and weight matrix choice. "*Nucleic Acids Res*", 22, 22 (Nov 11 1994), 4673-4680.

[4] Altschul, S. F., Gish, W., Miller, W., Myers, E. W. and Lipman, D. J. Basic local alignment search tool. "*J Mol Biol*", 215, 3 (Oct 5 1990), 403-410.

[5] Smith, T. F. and Waterman, M. S. Identification of common molecular subsequences. "*J Mol Biol*", 147, 1 (Mar 25 1981), 195-197.

[6] Zhang, Z., Schwartz, S., Wagner, L. and Miller, W. A greedy algorithm for aligning DNA sequences. "*J Comput Biol*", 7, 1-2 (Feb-Apr 2000), 203-214.

[7] Altschul, S. F., Madden, T. L., Schaffer, A. A., Zhang, J., Zhang, Z., Miller, W. and Lipman, D. J. Gapped BLAST and PSI-BLAST: a new generation of protein database search programs. "*Nucleic Acids Res*", 25, 17 (Sep 1 1997), 3389-3402.

[8] Buhler, J. D., Lancaster, J. M., Jacob, A. C. and Chamberlain, R. D. Mercury BLASTN: Faster DNA sequence comparison using a streaming hardware architecture. "*Proceedings of Reconfigurable System Summer Institute*"2007).

[9] Darling, A. E., Carey, L. and Feng, W. The Design, Implementation adn Evaluation of mpiBLAST. "*Proceedings of ClusterWorld*"2003).

[10] Shpaer, E. G., Robinson, M., Yee, D., Candlin, J. D., Mines, R. and Hunkapiller, T. Sensitivity and selectivity in protein similarity searches: a comparison of Smith-Waterman in hardware to BLAST and FASTA. "*Genomics*", 38, 2 (Dec 1 1996), 179-191.

[11] "*CLC Bioinformatics Cube. CLC Bio Inc., http://www.clcbio.com*"2010).

[12] Li, I. T., Shum, W. and Truong, K. 160-fold acceleration of the Smith-Waterman algorithm using a field programmable gate array (FPGA). "*BMC Bioinformatics*", 82007), 185.

[13] Manavski, S. A. and Valle, G. CUDA compatible GPU cards as efficient hardware accelerators for Smith-Waterman sequence alignment. "*BMC Bioinformatics*", 9 Suppl 22008), S10.

[14] Liu, Y., Maskell, D. L. and Schmidt, B. CUDASW++: optimizing Smith-Waterman sequence database searches for CUDA-enabled graphics processing units. "*BMC Res Notes*", 22009), 73.

[15] Szalkowski, A., Ledergerber, C., Krahenbuhl, P. and Dessimoz, C. SWPS3 - fast multi-threaded vectorized Smith-Waterman for IBM Cell/B.E. and x86/SSE2. "*BMC Res Notes*", 12008), 107.

[16] Farrar, M. Striped Smith-Waterman speeds database searches six times over other SIMD implementations. "*Bioinformatics*", 23, 2 (Jan 15 2007), 156-161.

[17] Wirawan, A., Kwoh, C. K., Hieu, N. T. and Schmidt, B. CBESW: sequence alignment on the Playstation 3. "*BMC Bioinformatics*", 92008), 377.

[18] Lindholm, E., Nickolls, J., Oberman, S. and Montrym, J. NVIDIA Tesla: A Unified Graphics and Computing Architecture. "*Micro IEEE*", 28, (2) 2008), 39-55.

[19] Howard, J., Dighe, S., Hoskote, Y., Vangal, S., Finan, D., Ryhl, G., Jenkins, D., Wilson, H., Borkar, N., Schrom, G., Pailet, F., Jain, S., Jacob, T., Yada, S., Marella, S., Salihundam, P., Erraguntla, V., Konow, M., Riepen, M., Droege, G., Lindemann, J., Gries, M., Apel, T., Henriss, K., Lund-Larsen, T., Steibl, S., Borkar, S., De, V., Van Der Wijngaart, R. and Mattson, T. A 48-Core IA-32 Message-Passing Processor with DVFS in 45nm CMOS. "*ISSCC*"2010), 19-21.

[20] Bell, S., Edwards, B., Amann, J., Conlin, R., Joyce, K., Leung, V., MacKay, J., Reif, M., Bao, L., Brown, J., Mattina, M., Chyi-Chang Miao, Ramey, C., Wentzlaff, D., Anderson, W., Berger, E., Fairbanks, N., Khan, D., Montenegro, F., Stickney, J. and Zook, J. TILE64 - Processor: A 64-Core SoC with Mesh Interconnect. "*ISSCC*"2008), 588-598.

[21] Galvez, S., Diaz, D., Hernandez, P., Esteban, F. J., Caballero, J. A. and Dorado, G. Next-generation bioinformatics: using many-core processor architecture to develop a web service for sequence alignment. "*Bioinformatics*", 26, 5 (Mar 1 2010), 683-686.

[22] Boston, U. d. *Scientists Discover Keys to Long Life*. http://online.wsj.com/article/SB10001424052748703571704575341034212066208.html, 2010.

[23] Ivanov, A. P., Instuli, E., McGilvery, C. M., Baldwin, G., McComb, D. W., Albrecht, T. and Edel, J. B. DNA Tunneling Detector Embedded in a Nanopore. "*Nano Lett*"(Dec 6 2010).

[24] Sonnhammer, E. L. and Durbin, R. A workbench for large-scale sequence homology analysis. "*Comput Appl Biosci*", 10, 3 (Jun 1994), 301-307.

[25] Hardison, R. C., Oeltjen, J. and Miller, W. Long human-mouse sequence alignments reveal novel regulatory elements: a reason to sequence the mouse genome. "*Genome Res*", 7, 10 (Oct 1997), 959-966.

[26] Chao, K. M., Zhang, J., Ostell, J. and Miller, W. A local alignment tool for very long DNA sequences. "*Comput Appl Biosci*", 11, 2 (Apr 1995), 147-153.

[27] Delcher, A. L., Kasif, S., Fleischmann, R. D., Peterson, J., White, O. and Salzberg, S. L. Alignment of whole genomes. "*Nucleic Acids Res*", 27, 11 (Jun 1 1999), 2369-2376.

[28] Hardison, R. and Miller, W. Use of long sequence alignments to study the evolution and regulation of mammalian globin gene clusters. "*Mol Biol Evol*", 10, 1 (Jan 1993), 73-102.

[29] Boguski, M. S., Lowe, T. M. and Tolstoshev, C. M. dbEST--database for "expressed sequence tags". "*Nat Genet*", 4, 4 (Aug 1993), 332-333.

[30] Cartwright, R. A. Logarithmic gap costs decrease alignment accuracy. "*BMC Bioinformatics*", 72006), 527.

[31] Bellman, R. and Rand Corporation. *Dynamic programming*. Princeton University Press, Princeton, 1957.

[32] Renganathan, K. *Bioinformatics: sequence alignment and markov models*. McGraw Hill, 2009.

[33] Cormen, T. H. *Introduction to algorithms*. MIT Press; McGraw Hill [distributor], Cambridge, Mass. Boston, 2001.

[34] Needleman, S. B. and Wunsch, C. D. A general method applicable to the search for similarities in the amino acid sequence of two proteins. "*J Mol Biol*", 48, 3 (Mar 1970), 443-453.

[35] Gotoh, O. An improved algorithm for matching biological sequences. "*J Mol Biol*", 162, 3 (Dec 15 1982), 705-708.

[36] Hirschberg, D. S. A linear space algorithm for computing longest common subsequences. "*Communications of the ACM*", 18, (6) 1975), 341-343.

[37] Charter, K., Schaeffer, J. and Szafron, D. Sequence alignment using FastLSA. "*International Conference on Mathematics and Engineering Techniques in Medicine and Biological Sciences*"2000), 239-245.

[38] Driga, A., Lu, P., Schaeffer, J., Szafron, D., Charter, K. and Parsons, I. FastLSA: A fast linear-space, parallel and sequential algorithm for sequence alignment. "*International Conference on Parallel Processing (ICPP'03)*"2003), 48-57.

[39] Chao, K. M., Zhang, J., Ostell, J. and Miller, W. A tool for aligning very similar DNA sequences. "*Computer applications in the biosciences CABIOS*", 13 (1)1997), 75-80.

[40] Ukkonen, E. On-line construction of suffix trees. "*Algorithmica*", 14 (3)1995), 249-260.

[41] Hohl, M., Kurtz, S. and Ohlebusch, E. Efficient multiple genome alignment. "*Bioinformatics*", 18 Suppl 12002), S312-320.

[42] Moore's law. "*http://en.wikipedia.org/wiki/Moore's_law*".

[43] Agarwal, A. and Levy, M. Going multi-core presents challenges and opportunitie. "*EE Times India*"April 2007).

[44] Goldratt, E. M. *Essays on the Theory of Constraints*. North River Press, 1998.

Anexos

Ficha microprocesador Tile64

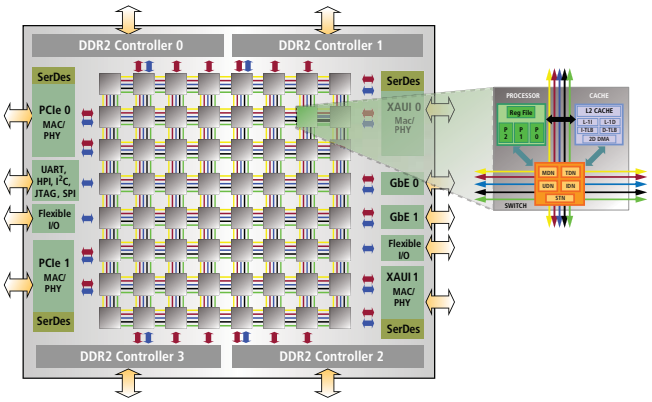
TILERA®

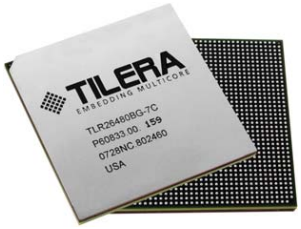
TILE64™ Processor

Product Brief

Overview

The TILE64™ family of multicore processors delivers immense computing performance to drive the latest generation of embedded applications. This revolutionary processor features 64 identical processor cores (tiles) interconnected with Tilera's iMesh™ on-chip network. Each tile is a complete full-featured processor, including integrated L1 and L2 cache and a non-blocking switch that connects the tile into the mesh. Each tile can independently run a full operating system, or a group of multiple tiles can run a multi-processing OS like SMP Linux.





The TILE64 processor family slashes board real estate and system costs by integrating a complete set of memory and I/O controllers, therefore eliminating the need for an external North Bridge or South Bridge. Tilera's architectural innovations deliver scalable performance, power efficiency and low processing latency in an extremely compact footprint.

The TILE64 Processor is programmable in ANSI standard C and C++, enabling developers to leverage their existing software and maximize their investments. Tiles can be grouped into clusters to apply the appropriate amount of horsepower to each application. Since multiple operating system instances can be run on the TILE64 simultaneously, it can replace multiple CPU subsystems for both the data plane and control plane.

Combining multiple C-programmable processor tiles with the iMesh multicore technology enables the TILE64 processor to achieve the performance of a fixed function ASIC or FPGA in a powerful software-programmable solution.

Product Differentiators

	Features	Enables
Massively Scalable Performance	• 8 X 8 grid of identical, general purpose processor cores (tiles) • Three-way VLIW pipeline for instruction level parallelism • 5 Mbytes of on-chip cache • Up to 443 billion operations per second (BOPS) • 31 Tbps of on-chip mesh interconnect enables linear application scaling • Up to 50 Gbps of I/O bandwidth	• 10 Gbps Snort® processing • 20 Gbps nProbe • 16 X 16 SAD at 540 MBlocks/s • H.264 HD video encode: 2+ streams of 720p30
Power Efficiency	• 700MHz - 866MHz operating frequency • 15 - 22W @ 700MHz all cores running full application • Idle Tiles can be put into low-power sleep mode • Power efficient inter-tile communications	• Highest performance per watt • Simple thermal management and power supply design • Lower operating cost
Integrated Solution	• Four DDR2 memory controllers with optional ECC • Two 10GbE XAUI configurable MAC or PHY interfaces • Two 4-lane PCIe interfaces: Root complex or endpoint mode • Two GbE MAC interfaces • Flexible I/O interface	• Reduces BOM cost – standard interfaces included on-chip • Dramatically reduced board real estate • Direct interface to leading L2-L3 switch vendors
Multicore Development Environment	• ANSI standard C / C++ compiler • Advanced profiling and debugging designed for multicore programming • Supports SMP Linux with 2.6 kernel • Tilera Multicore Components (TMC™) libraries for efficient inter-tile communication	• Run off-the-shelf C programs • Reduce debug and optimization time • Faster time to production code • Standard multicore communication mechanisms

For more information on Tilera® products, visit www.tilera.com.

TILE64™ Processor – Product Brief

Targeted Applications

The TILE64 family of processors has both the flexibility and performance to support a wide range of computing-intensive applications, including advanced networking, digital video, and telecommunication. Because it is a general-purpose MIMD multicore processor, it can run multiple operating systems and applications simultaneously. For example, it can perform 10Gbps of TCP offload (TOE) along with multiple streams of video transcoding. Virtual memory support and Tilera's Multicore Hardwall™ technology provides kernel-level protection related to both shared memory and user-level streaming and messaging.

Advanced Networking Products. The TILE64 processor is ideally suited to intelligent network services such as Unified Threat Management (UTM), L4-7 Deep Packet Inspection, or Quality of Service (QoS) provisioning. The impressive computing performance together with a complete integrated networking I/O subsystem makes the TILE64 a powerful single-chip communications processor.

Digital Multimedia Products. The TILE64 processor also excels at digital video and audio processing, easily taking the place of multiple DSPs to perform audio/video encoding, transcoding, video analytics, or other digital video manipulation. In addition, the control plane and any required networking capability can be handled by the same TILE64 device.

Since this processor is available in multiple power/performance configurations, the family can address varying price and performance needs.

Development Environment

Tilera's Multicore Development Environment™ (MDE) is a complete standards-based multicore programming solution that enables developers to take full advantage of the parallel processing potential of the Tile Processor™ architecture.

Powerful innovations allow the developer to take a Gentle Slope Programming™ approach to multicore software development. By leveraging Open Source software and the developer's existing software code base, designers can achieve impressive results in an extremely short period of time. Then, as developers become more familiar with large-scale multicore processing, they can take advantage of the enhanced tools and libraries offered in the MDE to optimize performance further.

Tilera's MDE includes:

- Standard Eclipse-based IDE
- ANSI standard C / C++ compiler
- Multi-tile cycle-accurate simulator
- Whole chip debug and performance analysis
- Complete SMP Linux support
- TMC library for efficient inter-communication
- PCIe Hardware development platform
- Linux and Windows host environments

Scalable Processing and Ease of Use for Embedded Application Developers

The TILE64 Processor addresses application developers' needs for scalable multiprocessor performance, performance per watt, and ease of development. The Tile Processor's on-chip iMesh network, the distributed cache architecture, and the industry-standard development tools combine to provide a solution uniquely suited to today's networking and multimedia applications.

Ordering Information

Part Number	Description	I/O Interfaces	Processor Frequency	DDR2 Memory Speed	Number of Tiles	Package	Operating Temperature
TLR26480-BG-7C	Standard	2 XAUI, 2 PCIe, 2 GbE	700MHz	800MHz	64	1517 BGA	0-70° Commercial
TLR26480-BG-9C	High-Performance	2 XAUI, 2 PCIe, 2 GbE	866MHz	800MHz	64	1517 BGA	0-70° Commercial

Tilera Corporation
2333 Zanker Road
San Jose, CA 95131

Phone: (408) 383-9292
Fax: (408) 383-9225
www.tilera.com

For more information on Tilera products, visit www.tilera.com.

TILERA
PB010 Rev4

Ficha tarjeta TILExpress-20G

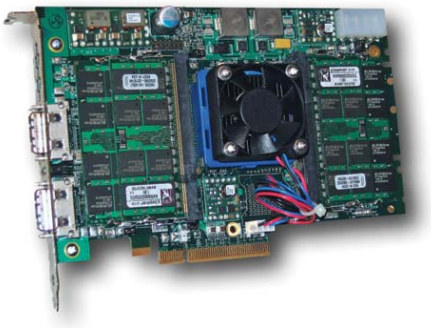
TILERA®

TILExpress-20G™ Card

Product Brief

Unprecedented Computing Power in a Half-Length PCI Express Plug-In Card

The TILExpress™ family of PCIe cards delivers the industry's highest performance processing for demanding applications in networking and digital multimedia. The board features Tilera's powerful TILE64™ processor together with multiple network interfaces, speeding time-to-market for embedded appliances.



The 64-core TILE64 processor's versatility and C/C++ programmability allows it to perform a variety of math-intensive tasks such as:

- Network traffic flow-classification
- Deep Packet Inspection (DPI)
- Quality of Service (QoS) provisioning
- Video stream processing
- Video and audio encoding / transcoding

A pair of 10-gigabit Ethernet ports (CX4) provide an impressive 20Gbps of network I/O. An optional mezzanine expansion board further increases flexibility by supporting additional I/O options, such as 10G optical interfaces or multiple 1G ports. The TILExpress-20G card is also available as a PCIe-based co-processor solution with no I/O interfaces for computation-only applications.

Target Applications

Advanced Networking: The TILExpress-20G card provides the Ethernet line interfaces as well as the entire data plane processing for intelligent network services such as:

- Unified Threat Management (UTM)
- Intrusion Prevention (IPS/IDS)
- Network security appliances
- Network monitoring and forensics
- L4-7 Deep Packet Inspection

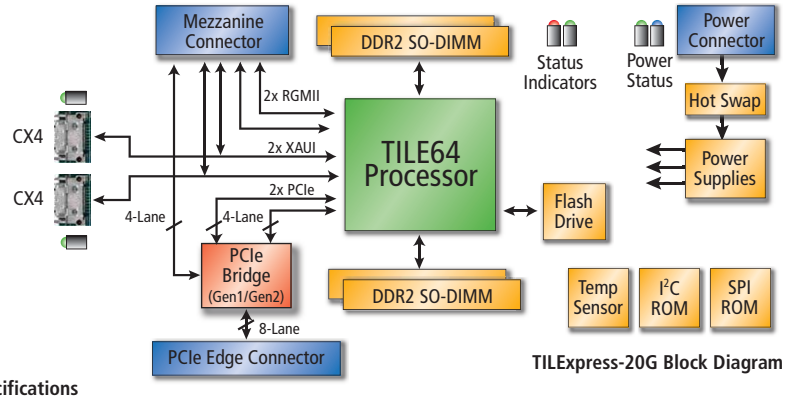
Digital Video: The TILExpress-20G card also excels at digital video processing, easily taking the place of multiple DSPs as well as the networking functions for:

- Broadcast "Head-End" media services
- Video-on-Demand (VoD) servers
- Video conferencing
- Video surveillance
- Audio and video transcoding

	Features	Enables
Massive Compute in a Small Form Factor	• TILE64 processor provides 64 general purpose processor cores for compute-intensive applications	• 10Gbps Snort® processing • 20+ Gbps nProbe • 16 X 16 SAD at 540 MBlocks/s • H.264 HD video encode: 2+ streams of 720p
Powerful Software Approach	• C/C++ programming model • Supports Linux and SMP Linux • Standard tool chain based on Eclipse and GNU	• Easy to leverage existing code • Take advantage of huge body of Open Source applications • Familiar programming environment speeds development
Rich I/O Capabilities	• Eight-lane PCIe interface (Gen1 and Gen2 support) • Two 10Gbps Ethernet ports (CX4) • Co-processor configuration with no I/O ports	• >12Gbps bandwidth between TILExpress and host processor • 20Gbps of I/O or dual 10Gbps with redundancy • Single card combines processing and network connectivity
Memory Options	• Standard 4 GBytes of socketed SO-DIMM DDR2 memory • On-board 1 GByte Solid-State Flash Drive for stand-alone boot	• Industry-standard upgradable memory modules • Flexible memory options for a variety of applications • Card can boot from host across PCIe or from Flash
PCIe Card Format	• Standard half-length PCIe card for x8 PCIe slots • Single-slot board spacing	• Plug-in solution for PCIe motherboard-based appliances • Instant hardware compatibility enables rapid time-to-market

For more information on Tilera® products, visit www.tilera.com.

TILExpress-20G™ Card – Product Brief



TILExpress-20G Block Diagram

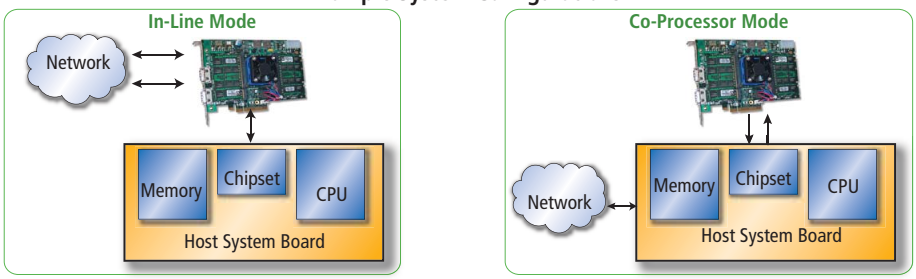
Specifications

General	• Form factor: Full-height, half-length PCI Express • RoHS-6 compliant per EU 2002/95/EC
PCIe Interface	• Eight-lane PCIe 2.0, Gen2 compliant • Fall-back to 4, 2 or 1 lane. Fall-back to Gen1 mode • Hot plug compliant
Network Interfaces	• Two 10Gbps Gigabit Ethernet (25-pin CX4) • Mezzanine card option for dual 1Gbps RJ45 • Can be used in co-processor mode with interfaces disabled
Memory	• Four SO-DIMM sockets for DDR2 SDRAM • PC2-6400 (DDR2-800), 1GB to 16GB total capacity • 1 GByte on-board Flash drive
Indicators	• Four on-board power supply status indicators • LEDs on rear plate for link status

Technical Features	• Supports INTx and MSI interrupts across PCI • On-board dual temperature sensor • JTAG test connector • Mezzanine expansion connector
Power	• Standard 4-pin Molex "ATX" connector for power • Single 12V supply from ATX connector • Typical power dissipation: 43W @ 700 MHz • Recommended airflow: 1.1 m/s (200 lfm) @55° C • Cooling options: - Active cooling fan (Standard) - Passive heatsink[1]. Required airflow: 1.1 m/s (200 lfm) @55° C
Environmental	• Operating temperature: 0° to 55°C • Storage temperature: -40° to 70°C • Humidity: 10% to 90% RH non-condensing
Physical Specifications	• Board Height: 11.1 cm (4.38 in) • Board Length: 16.8 cm (6.6 in) • Bracket Height: 2cm (4.72 in)

[1] Passive heatsink for 700MHz version only.

Example System Configurations



Ordering Information

Part Number	Processor	Memory Installed	I/O Interfaces	Processor Frequency	Operating Temperature
TLB-26400-7-PCIe-2X10-4-GC	TILE64	4 GBytes	Two 10G CX4	700 MHz	Commercial
TLB-26400-9-PCIe-2X10-8-GC	TILE64	8 GBytes	Two 10G CX4	866 MHz	Commercial

For more information on Tilera products, visit www.tilera.com.

Contact Tilera sales for other card configurations.

TILERA
P8018 Rev 3

Tilera Corporation
2333 Zanker Road
San Jose, CA 95131

Phone: (408) 383-9292
Fax: (408) 383-9225
www.tilera.com