Capítulo 7

Pruebas y Análisis

7.1. Descripción de las pruebas

Para evaluar el rendimiento de las trasformadas de Fourier implementadas se han ejecutado diversas pruebas que siguen el mismo esquema:

1. Reservar espacio para un cubo de las dimensiones deseadas.
2. Invocar a la función que inicializa los *threads* de los SPEs y el array que contiene los *twiddle factors*.
3. Inicializar los arrays de valores complejos que forman el cubo.
4. Ejecutar la FFT-3D implementada sobre el cubo.
5. Liberar el espacio de memoria que contenía el cubo.

Las diferentes pruebas difieren entre sí según una serie de parámetros, los cuales varían dependiendo de la versión de FFT-3D que estemos ejecutando. A partir de ahora nos referiremos a la versión de FFT-3D según se implemente en ella una de las dos versiones del algoritmo de transposición, **A** o **B**. En el Cuadro 7.1 podemos ver cuáles son estos parámetros.

versión / parámetro	**A**	**B**
Tamaño cubo	64^3, 128^3, 256^3	
Tamaño bloque	32, 64	32, 64, 128
Núm. SPEs	1, 2, 4, 8	2, 4, 8

Cuadro 7.1: Parámetros de las pruebas de cada una de las dos versiones de FFT-3D.

Las pruebas se han ejecutado un total de 10 veces para cada una de las posibles combinaciones de parámetros.

La métrica que se utilizará para evaluar el rendimiento es el tiempo, que normalmente expresaremos en milisegundos.

Las secciones de código cuyo tiempo medimos se encuentran a varios niveles, por lo que también se han utilizado distintas maneras de obtener el tiempo consumido:

- El tiempo total de cálculo de la FFT-3D se mide en el programa de prueba, invocando a la llamada a sistema *gettimeofday* antes y después de ejecutar la FFT, para saber cuántos segundos y milisegundos han pasado.

- En el código que se ejecuta en el PPE se calcula el tiempo de cada una de las partes de la FFT-3D. Sabremos cuánto tiempo se invierte en cada FFT-1D y transposición. Para esto utilizamos también la llamada a sistema *gettimeofday*.

- Podemos medir secciones del código que se ejecuta en el SPE mediante *decrementers*.

En todas estas mediciones también se ha distinguido entre el tiempo que se se invierte en transferencias de datos y el resto de cálculos.

Los resultados que se obtienen no incluyen la fase de inicialización de *threads* y *twiddle factors*, ya que esta FFT-3D está pensada para ser invocada numerosas veces después de una sola inicialización. De todas maneras, el tiempo invertido en esta inicialización está alrededor de 1.5 milisegundos, casi despreciable, según el resto de cálculos, que veremos a continuación.

7.2. Análisis de resultados

En el Apéndice A podemos encontrar tablas con los resultados de las diferentes pruebas, calculados como la media de cada ejecución. Nos remitiremos a ellos cuando sea necesario, ya que principalmente vamos a realizar el análisis del rendimiento basándonos en los gráficos de las Figuras 7.2, 7.3, 7.4 y 7.5.

Las Figuras 7.2 y 7.3 reúnen tres gráficas cada una que expresan las partes de comunicación (*communication*) y no comunicación (*non communication*) de una FFT-3D. La no comunicación se extrae de la resta del tiempo total y del de comunicación. La parte de no comunicación no se corresponde exactamente con las porciones de código de cálculo, ya que por ejemplo en el cálculo se puede solapar mediante doble *buffering* con la parte representada en comunicación.

Por otra parte, las figuras 7.4 y 7.5 muestran cómo se invierte el tiempo en cada una de las partes que componen una FFT-3D. Normalmente tendremos las partes de primera FFT-1D (*1st FFT*), primera transposición (*1st transp*), segunda FFT-1D (*2nd FFT*), segunda transposición (*2nd transp*) y tercera FFT-1D (*3rd FFT*). Sin embargo, recordamos el caso en que la FFT-3D se puede hacer en dos pasos, visto en la sección 6.2.4, por ello en algunas

barras de las gráficas sólo habrá representación de las partes identificadas como *FFT2D* y *transp+fft*.

En cualquiera de las gráficas el eje de las x estará formado por agrupaciones de barras según el número de SPEs que se hayan utilizado. La etiqueta vertical que identifica cada barra tiene la forma [*Tamaño de bloque*] - [*Versión*]. Existe un caso especial, que no está contemplado en el Cuadro 7.1, y es el que está representado como 128-A. Tanto este como el 64-A corresponden a la FFT-3D que puede hacerse en dos pasos y que, aunque esté etiquetado como A, e internamente esté implementado dentro de la función de la versión A, no utiliza la versión de transposición que se usa en A.

También queremos indicar que en las Figuras 7.2 y 7.4 el eje y corresponde al tiempo en segundos, mientras que en las Figuras 7.3 y 7.5 representa el tanto por cien invertido en cada sección de código.

Seguidamente vamos a extraer varios puntos que se desprenden del análisis de los resultados obtenidos, agrupados según parámetros variables.

Según el tamaño del cubo

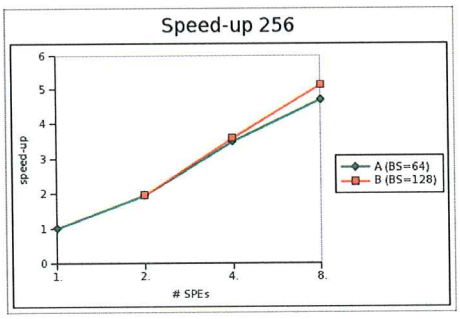
En primer lugar, la diferencia de tiempo entre cada uno de los tres tamaños de cubo es de un orden de magnitud, hay una relación aproximada de $tam \times 2 \longleftrightarrow tiempo \times 10$.

Si nos fijamos en la descomposición de la FFT-3D en cinco partes, vemos que para cubos de tamaño 64^3 y 128^3 en la operación que mayor tiempo se invierte es en cualquiera de las tres FFT-1D. Sin embargo, cuando aumentamos el tamaño a 256^3, el tiempo de la segunda transposición es mayor que el de cualquier otra operación (7.5c). Recordemos que en esta transposición hay que hacer transferencias desde memoria principal a LS de los SPEs de filas no consecutivas. Esto puede crear que, cuando hay que realizar un número importante de transferencias, se provoque contención en el EIB. Este argumento viene reforzado al observar que en las gráficas de la Figura 7.2, para el caso de 256^3 el tiempo dedicado en la comunicación aumenta considerablemente.

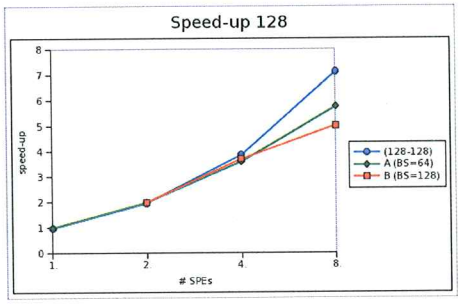
Según el número de SPEs

Los **speed-up** conseguidos al variar el número de SPEs para cada versión del programa se pueden ver en las Figuras 7.1a, 7.1b y 7.1c, correspondientes a tamaños de cubo 256^3, 128^3 y 64^3, respectivamente. Como no existen datos de la versión B con un SPE, se ha supuesto que se parte del mismo speed-up que A a partir de dos SPEs.

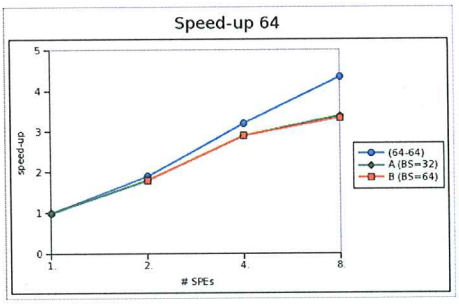
Como vemos, el *speed-up* para las versiones A y B nunca es lineal, que sería el caso ideal al incrementar el número de unidades computacionales. Sin embargo, para el caso que en las gráficas ha sido representado como *(128-128)* y *(64-64)*, el de la FFT-3D en dos pasos, podemos observar que sí se acerca bastante al ideal. Todo esto nos hace pensar en el tiempo



(a) Tamaño = 256^3



(b) Tamaño = 128^3



(c) Tamaño = 64^3

Figura 7.1: Comparación de *speed-ups* de las diferentes versiones de FFT-3D implementadas según el tamaño del cubo.

de comunicación, ya que la proporción de tiempo empleado en transferencias es menor para la versión FFT-3D en dos pasos.

Sin embargo, sea cual sea la versión, conforme aumenta el número de SPEs, mayor es la proporción de tiempo empleada en la transferencia de datos. Esto se debe a que se forma contención en el acceso a memoria principal al tener que realizarse cada vez más accesos concurrentes. Por esto, aunque el tiempo de cálculo se ve reducido considerablemente al aumentar el paralelismo, no pasa así con el tiempo de comunicación, que acaba siendo el talón de Aquiles de la aplicación.

Otro hecho que se desprende del análisis de resultados es que, para una cantidad de SPEs concreta, el tiempo de cálculo suele ser constante independientemente de la versión de transposición implementada. Esto nos hace pensar que el impacto en el rendimiento viene dado por el tratamiento que se haga de los datos en las transferencias.

Por otra parte, se puede apreciar en la Figura 7.5 que, al aumentar el número de SPEs, se reparte la proporción de tiempo invertida en cada función de las que está compuesta la FFT-3D, al reducirse el tiempo de cálculo de las FFT-1D en mayor proporción que en las transposiciones. Debido a que el cálculo de FFTs es una tarea computacionalmente intensiva, esto nos puede indicar la potencia de los SPEs como aceleradores numéricos.

Según el tamaño de bloque

Para cualquier versión del algoritmo de transposición se repite un comportamiento que enlaza con lo comentado en el punto anterior: mientras que el tiempo de cálculo se mantiene constante, el de comunicación va variando conforme varía el tamaño del bloque utilizado para el algoritmo de transposición. Así, cuanto menor sea el tamaño de bloque, mayor será el tiempo invertido en la comunicación. Esto se puede apreciar claramente en las Figuras del grupo 7.5, en las que se aprecia que la función cuyo tiempo más se incrementa conforme disminuye el tamaño de bloque es la que realiza la segunda transposición, en la que las filas de los planos no se encuentran consecutivas en memoria.

En [7] podemos encontrar una respuesta a este fenómeno. Según los autores, en las transferencias de elementos DMA influye el tamaño de los datos transmitidos. Hasta que no se llega a los 1024 Bytes no obtendremos el mejor pico de rendimiento en la comunicación. Por eso se recomienda el uso de listas DMA [18] para datos que ocupen menos de 1024 Bytes, cuyo rendimiento para transferencias de más de 1024 Bytes es menor, pero se mantiene constante en la eficiencia con transferencias de pocos Bytes.

En nuestro caso, el bloque más pequeño se tendrá que transferir por filas de 32 elementos complejos, que ocupan un total de 256 Bytes, desaprovechando de esta manera el ancho de banda del bus EIB. Por eso, al haber implementado estas transferencias mediante elementos DMA y no mediante listas, se puede estar provocando un impacto negativo en el rendimiento de la comunicación.

Para acabar, si nos fijamos en las proporciones del tiempo de comunicación en la que transferimos bloques de 128 elementos, es decir, 1024 Bytes, este tiempo de transferecia es menor, ya que se ha alcanzado el pico anteriormente mencionado.

Comparación entre versiones

La versión que realiza la FFT-3D en dos pasos es, con diferencia, la que mejor rendimiento obtiene para los dos tamaños de cubo posibles (128,64), en contraste con las otras dos versiones, A y B. Esto se debe a la reducción de comunicaciones entre memoria principal y memoria local del SPE.

Cuando realizamos la FFT-3D en cinco pasos, para un mismo tamaño de bloque suele ser mejor la versión A que la B. En el único caso en que la versión B supera a la A es con el tamaño de bloque = 128 y el cubo de 256^3 elementos, al aprovecharse más eficientemente el bus en las transferencias. Sin embargo, para la mayoría de los casos, recordemos que en la versión A teníamos la ventaja de poder utilizar el double *buffering* al encontrarse los dos bloques en la misma SPE. Por esto, podemos solapar la transferencia de un bloque con el cálculo de otro, ganando así en eficiencia.

Estimación de FFT-3D completa

Para el mejor resultado de cada versión de la FFT-3D parcial que hemos desarrollado, se ha hecho una estimación de lo que implicaría realizar los dos últimos pasos de la FFT-3D (dos transposiciones extra). De este modo obtendríamos la orientación del cubo original, pudiendo extender el algoritmo a otros ámbitos no tan específicos como el cálculo de la correlación cruzada o la convolución.

En el Cuadro 7.2 podemos ver los tiempos para 8 SPEs, según el tamaño del cubo.

Tamaño cubo	**FFT-3D parcial**	**FFT-3D completa**
256^3	0.0869	0.1309
128^3	0.00514	0.00814
64^3	0.00111	0.001828

Cuadro 7.2: Estimación de tiempos (en segundos) para una FFT-3D completa.

Según podemos observar, evitando las dos últimas transposiciones, tenemos una ganancia del tiempo de 1.51x (veces más rápido) para el tamaño 256^3, 1.58x para el tamaño 128^3 y 1.64x para el tamaño 64^3, respecto a la hipotética versión de FFT-3D completa.

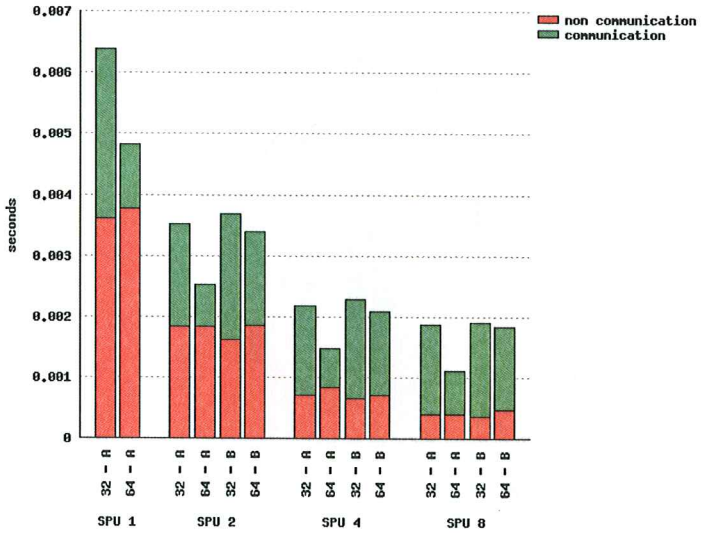
7.3. Integración en la aplicación FTDock

A continuación presentamos resultados para la adaptación de la aplicación FTDock con el nuevo código de FFT-3D. Además de este cambio, se ha realizado *function offloading* en la función de discretización de moléculas y a la multiplicación de complejos. También se ha aplicado esta técnica al filtro de *scoring*, incluyendo una estrategia de búsqueda vectorizada. Los detalles de implementación de esta parte se encuentran en [12], por lo que aquí no nos extenderemos explicándolos.

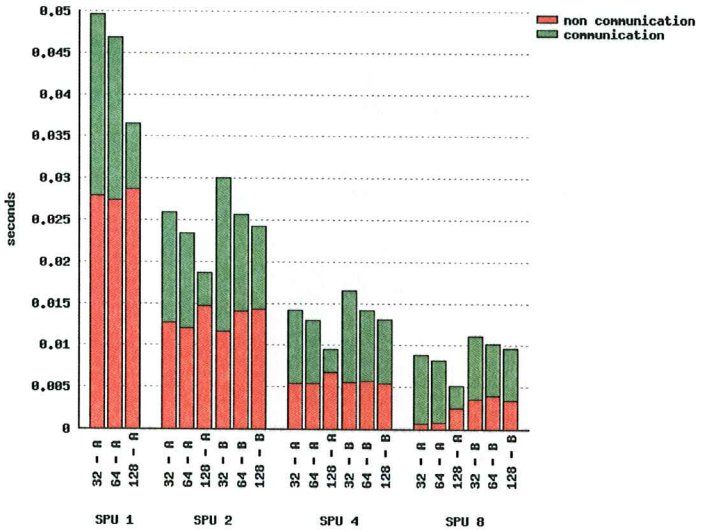
Las pruebas se han realizado con la molécula identificada como 2PKA como objetivo y la molécula 1BPI como ligando, que están en el conjunto de pruebas del FTDock original [14]. Además, se ha desactivado la función de *scoring* electrostática y se ha realizado un menor número de rotaciones (1000 de las más de 9000 del programa original), ya que no eran necesarias para la caracterización de la aplicación. El tamaño del cubo con el que se han realizado los tests es de 128^3 y 256^3. Se han ido variando el número de SPEs en las diferentes pruebas, ejecutándose con 1, 2, 4 y 8 SPEs.

La versión de la FFT-3D que se ha probado es la A, teniendo en cuenta que para el tamaño 128^3 se realiza la FFT-3D en dos pasos.

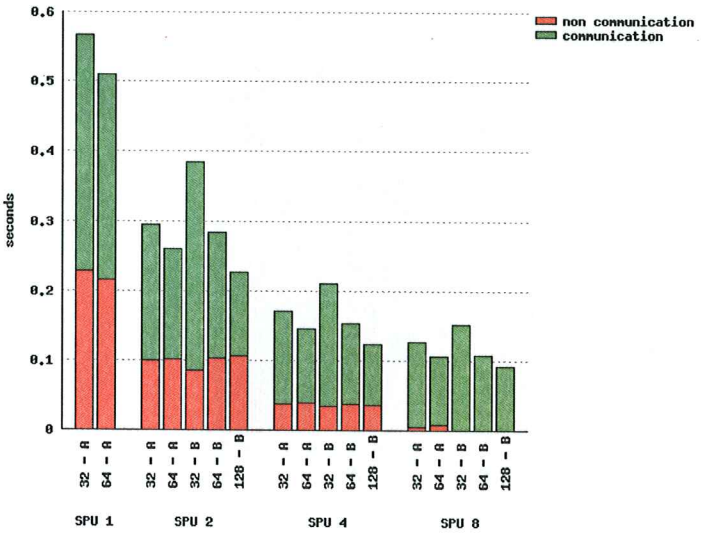
En el Cuadro 7.3 podemos ver los diferentes tiempos obtenidos por la aplicación. Según el número de SPEs utilizadas, estos tiempos escalan tal y como se observa en la Figura 7.6.



(a) Tamaño = 64^3

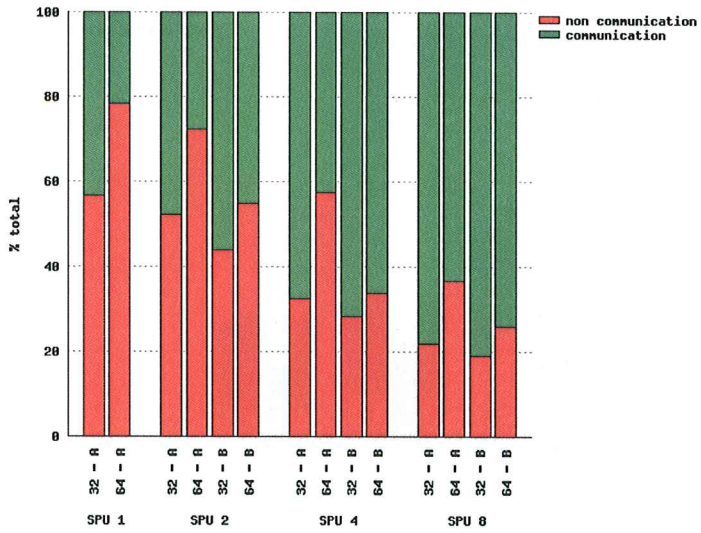


(b) Tamaño = 128^3

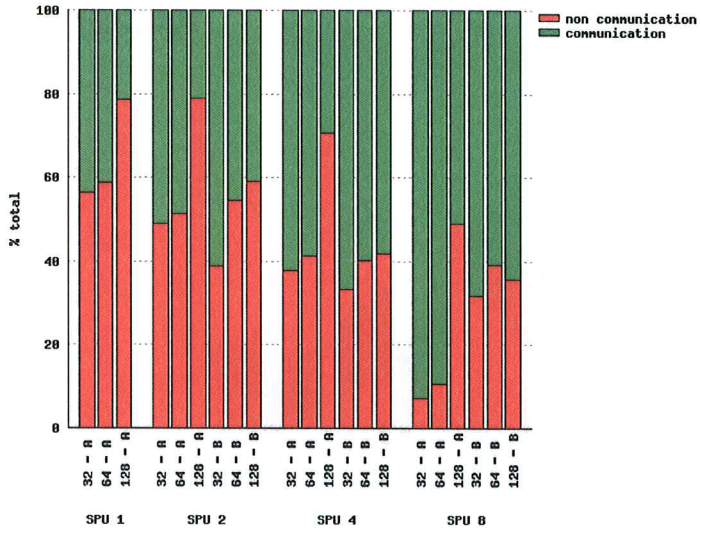


(c) Tamaño = 256^3

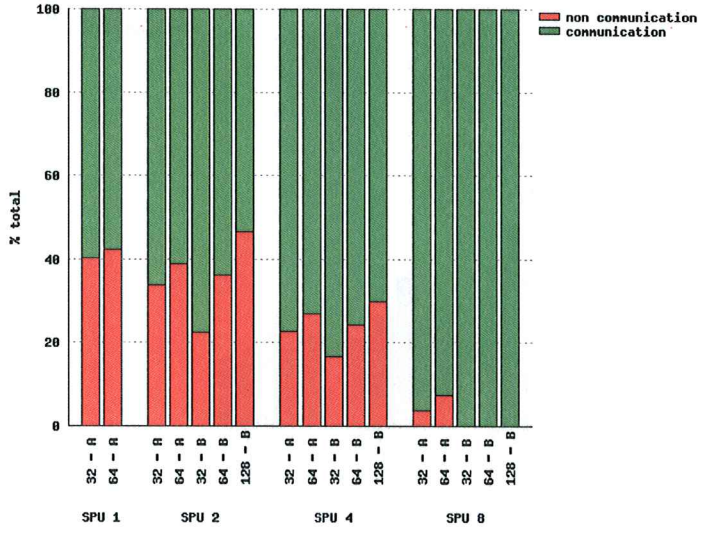
Figura 7.2: Gráficas comparativas de los tiempos totales de comunicación y no comunicación según versión de la FFT-3D, tamaño de bloque, número de SPEs para cada tamaño de cubo.



(a) Tamaño = 64^3

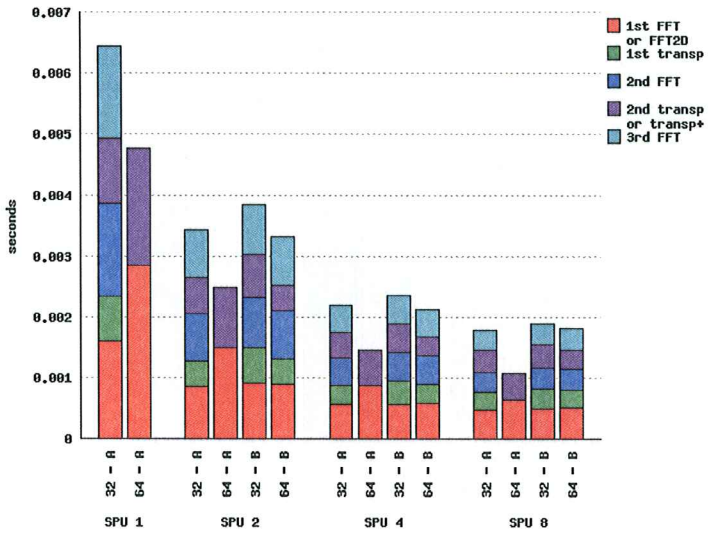


(b) Tamaño = 128^3

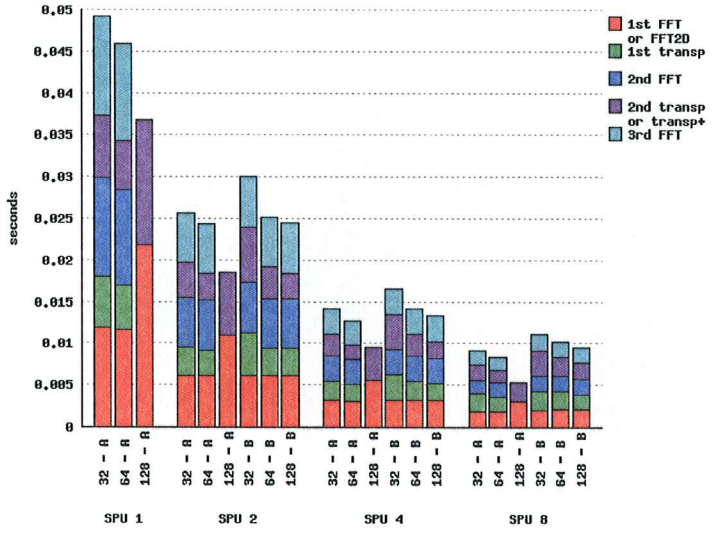


(c) Tamaño = 256^3

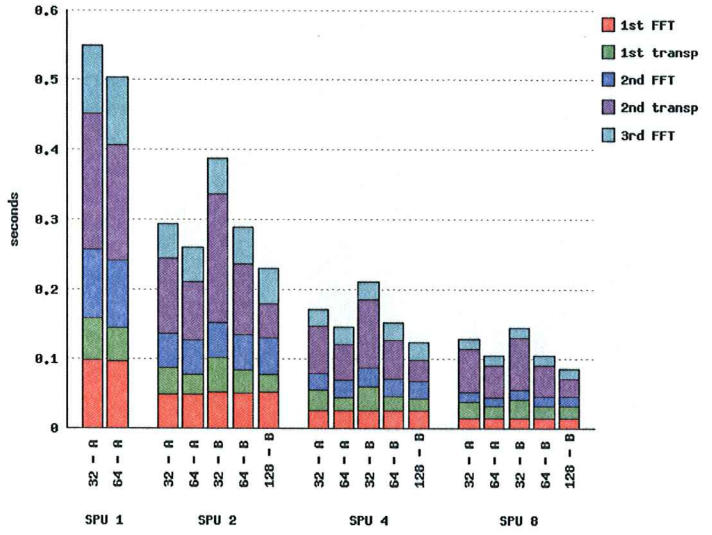
Figura 7.3: Gráficas comparativas de la proporción de los tiempos de comunicación y no comunicación según versión de la FFT-3D, tamaño de bloque, número de SPEs para cada tamaño de cubo.



(a) Tamaño = 64^3

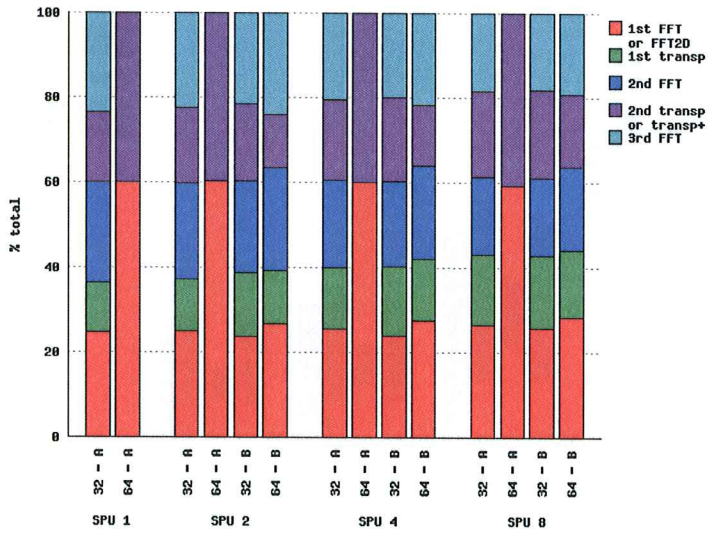


(b) Tamaño = 128^3

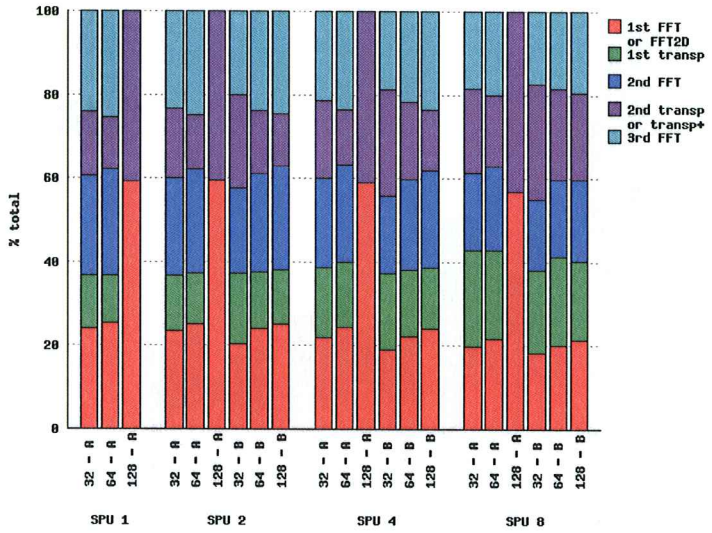


(c) Tamaño = 256^3

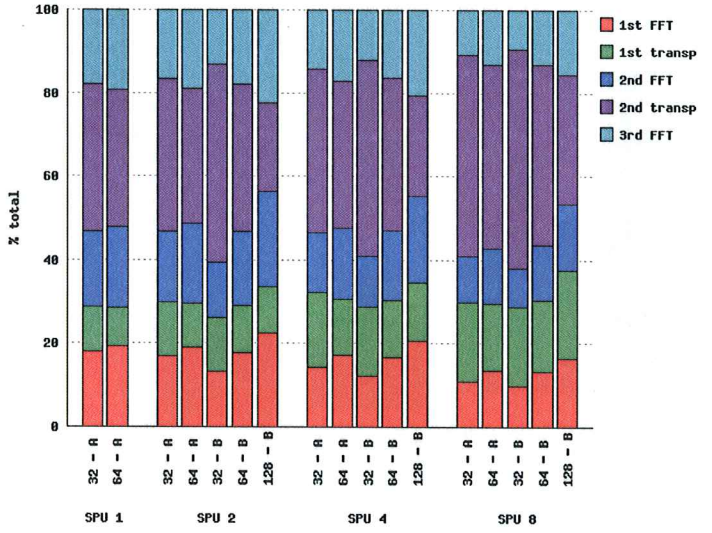
Figura 7.4: Gráficas comparativas de los tiempos totales de cada parte del algoritmo, según versión de la FFT-3D, tamaño de bloque, número de SPEs para cada tamaño de cubo.



(a) Tamaño = 64^3



(b) Tamaño = 128^3



(c) Tamaño = 256^3

Figura 7.5: Gráficas comparativas de la proporción del tiempo invertido en cada parte del algoritmo, según versión de la FFT-3D, tamaño de bloque, número de SPEs para cada tamaño de cubo.

Comparando esta gráfica con las de las Figuras 7.1a y 7.1b, vemos que el *speed-up* de la aplicación FTDock está muy relacionado con el *speed-up* de las funciones de FFT-3D, que son la carga mayor de trabajo de la aplicación.

Núm. SPEs \ Tam. cubo	128^3	256^3
1	120,32	1363,01
2	61,54	713,91
4	32,45	398,57
8	18,68	264,26

Cuadro 7.3: Tiempo de diversas ejecuciones de FTDock en Cell BE (en segundos).

El tiempo invertido en una de esta pruebas con FTDock estará repartido entre:

- $1000 + 1$ discretizaciones, una por la molécula estática al principio y otras 1000 por el ligando, después de cada nueva rotación.
- $1 + 1000 \times 2$ FFTs. Al principio se hace la FFT de la molécula estática. En cada una de las 1000 iteraciones se realiza la FFT de la molécula que cambia su orientación y la FFT inversa del resultado de la multiplicación de complejos.
- 1000 multiplicaciones de complejos.
- 1000 filtros de *scoring*.

Con los datos que figuran en el Apéndice A, podemos ver que el tiempo en realizar una FFT-3D para la versión en dos pasos, con tamaño del cubo 128^3 y 8 SPEs, vemos que tarda 0.005146 segundos. Con este dato y el tiempo total que figura en el Cuadro 7.3 podemos separar el tiempo que se invierte en las FFT y el resto de cálculos:

$$TiempoFFT_{A-128} = 0{,}005146 \times (1000 \times 2 + 1) = 10{,}2971$$

segundos

$$TiempoResto_{A-128} = 18{,}68 - 10{,}2971 = 8{,}38285$$

segundos

Estos cálculos nos sirven para la estimación de qué pasaría si se hubiera implementado el FTDock con la versión B de FFT-3D, con bloques de tamaño 128, que según la Figura 7.2 es mejor que operar con bloques de 32 y 64 elementos. Ahora el tiempo de una FFT-3D con 8 SPEs ascenderá a 0.009623 segundos.

$$TiempoFFT_{B-128} = 0{,}009623 \times (1000 \times 2 + 1) = 19{,}24$$

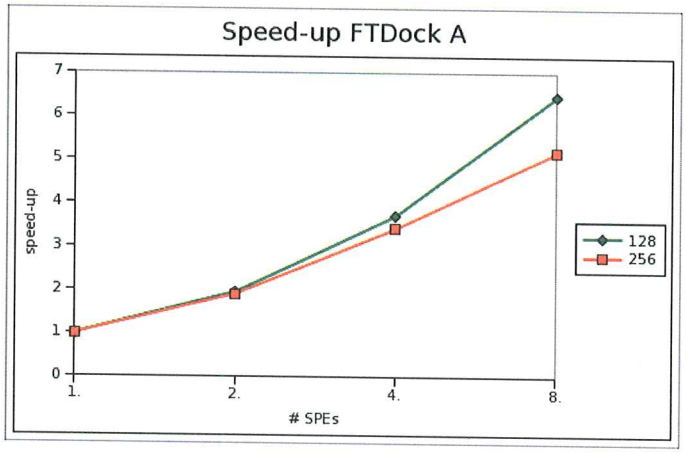


Figura 7.6: Speed-up del programa FTDock con la versión A de FFT-3D desarrollada.

segundos

Si a este resultado le sumamos lo correspondiente a la parte que depende de otros cálculos no FFT, que tomaremos como constante, tenemos que:

$$TiempoTotal_{B-128} = 19{,}24 + 8{,}38 = 27{,}64$$

segundos

Como vemos, la diferencia entre ambas versiones de FFT-3D es considerable. Concretamente, FTDock con la versión A de FFTs se estima 1.47x más rápida que la versión B.

Capítulo 8

Gestión del proyecto

8.1. Planificación

A continuación mostramos un calendario de planificación con las diferentes tareas realizadas para completar el proyecto. En él se observa el tiempo dedicado a cada tareas a lo largo de los más de cinco meses de desarrollo.

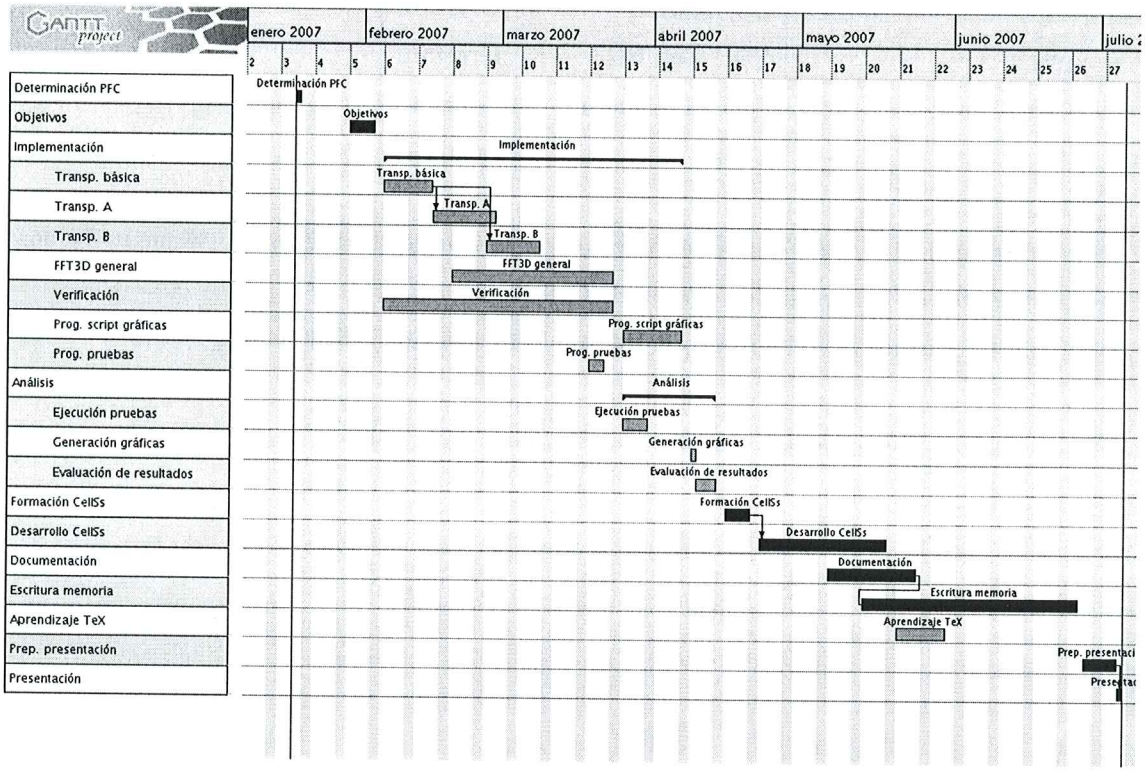


Figura 8.1: Calendario de la planificación del proyecto.

8.2. Costes

Para calcular los costes del proyecto hemos agrupado las tareas que se han visto en el diagrama de la Figura 8.1 en diferentes ámbitos, en los que se habrán invertido un número de horas concreto.

Ámbito	**Horas**
Formación	40
Implementación	440
Pruebas y análisis	120
Documentación	240
Total	840

Si suponemos que el proyecto ha sido desarrollado por un becario contratado por medio de un convenio Universidad-Empresa, el cual cobra 6,5 euros la hora, el coste de personal será de 5460 euros.

Por otra parte, el coste de los recursos software y hardware son los siguientes:

Recurso	**Euros**
Procesador Cell BE	600
SO Linux	Gratuito
SDK Cell BE	Gratuito
GCC Toolchain	Gratuito
Total	600

Para este cálculo hemos supuesto el precio de una consola PlayStation 3, ya que no hemos encontrado datos de los precios de *Blades* de Cell BE.

Por tanto, tenemos que el coste total del proyecto asciende a 6060 euros.

Capítulo 9

Conclusiones

Este proyecto final de carrera forma parte de un trabajo de investigación orientado al estudio de aplicaciones bioinformáticas, que ha sido desarrollado por un grupo del departamento de Arquitectura de Computadores. Esta investigación será presentada en un poster en la conferencia Parallel Architectures and Compilation Techniques (PACT), 2007. El estudio aparece en los *proceedings* de dicha conferencia.

En el proyecto que hemos presentado aquí, se ha analizado el rendimiento de las Fast Fourier Transform en 3D (FFT-3D), en el contexto de una aplicacion bioinformática de *protein-docking* (FTDock), para una máquina *heterogeneous multicore*, el Cell Broadband Engine (BE).

La aplicación FTDock se utiliza para el diseño de nuevos medicamentos. Para ello busca la mejor orientación en la unión de dos moléculas mediante correlación cruzada. Para este cálculo utiliza las FFT-3D.

Con el objetivo de analizar el comportamiento de las FFT-3D en Cell BE se han realizado las siguientes tareas:

- Análisis de la arquitectura del procesador multicore Cell BE, formado por un PowerPC Processor Element (PPE) y ocho Synergistic Processor Elements (SPE). Cada SPE tiene una memoria local (LS), de tamaño limitado. También puede acceder a la memoria principal del sistema, mediante transferencias DMA.
- Estudio de diferentes modelos de programación específicos para este procesador.
- Análisis de la aplicación FTDock.
- Análisis de diferentes implementaciones propias de FFT-3D, en las que juega un papel relevante la operación de transposición de matrices.

Como resultado del trabajo realizado, hemos llegado a ciertas conclusiones.

En primer lugar, siempre que sea posible, para aprovechar el rendimiento de Cell BE debemos paralelizar el código en los SPEs. Esta tarea se podrá realizar gracias al modelo de

programación *function offloading*. En este modelo, el programa que se ejecuta en el PPE será utilizado para la gestión de los *threads* de los SPEs, y para invocar a las funciones que son delegadas en el código de un SPE.

La aplicación desarrollada que calcula la FFT-3D es una adaptación muy específica para el programa de *docking* en el que será utilizada. Se ha intentado buscar el máximo rendimiento, de manera que se ha evitado en la medida de lo posible realizar transferencias de DMA y cálculos innecesarios. La aplicación FTDock realiza la correlación cruzada mediante transformadas de Fourier. Hemos visto que podemos evitar dos transposiciones (de cuatro), de modo que se puede conseguir una ganancia de más del 50 % del tiempo total de ejecución. No obstante, si se quisiera adaptar la FFT-3D desarrollada para el caso general sería fácilmente asumible.

Por otra parte, es importante la paralelización en los SPEs mediante *function offloading* debido a que la potencia de cálculo de los SPEs es mejor que la del PPE, en especial si el código es vectorizable, ya que se mejora el rendimiento de las FFT-3D y el FTDock. Además los accesos a memoria desde los SPEs son más rápidos que los que realiza el PPE, gracias a las transferencias DMA. El PPE tiene que pasar por la jerarquía de memoria, donde no es capaz de conseguir el ancho de banda máximo ofrecido por la memoria principal.

Otro punto a destacar respecto a los SPEs es su escalabilidad, la cual es buena pero no es lineal, debido a que el cálculo intensivo sí escala correctamente, pero en las transferencias con memoria principal nos encontramos un cuello de botella. Por esto, si deseamos alcanzar un *speed-up* lineal, deberemos reducir el número de transferencias siempre que podamos, rentabilizando cada DMA *put/get* intentando transferir a la vez el mayor número de datos. Otra manera de rentabilizarlas es reusar los datos, tal y como hacemos en las FFT-3D en que juntamos pasos. También se podría proponer un cambio en la arquitectura; si el LS del SPE fuera de mayor tamaño, podríamos reducir el número de transferecias a realizar.

Otro aspecto que mejora la eficiencia de los programas en Cell BE es el doble *buffering*, por lo que en los momentos en que sea posible debemos solapar la transferencia de datos con el cálculo.

Respecto a la programación en CellSs, podemos concluir varios aspectos. El modelo simplifica mucho la programación en la arquitectura, ya que la gestión de *threads* y la paralelización de tareas se realiza de manera transparente. La obtención de datos desde los SPEs también se puede simplificar siempre que los datos se encuentren en memoria contigua. Sin embargo, en nuestro caso, tenemos que acceder a posiciones de memoria no contigua, por lo que continuamos necesitando funciones que nos proporcionen control sobre las transferencias DMA. Además, se ha tenido que utilizar un cubo auxiliar para poder completar las transposiciones, por lo que ya no es un algoritmo *in-place*.

Finalmente, me gustaría destacar la formación que me ha aportado desarrollar un proyecto

de esta envergadura, cuya realización no habría sido posible sin los conocimientos adquiridos en las asignaturas de la carrera. Destaco, en primer lugar, las asignaturas de Sistemas Operativos, que me dieron la base de la programación con *threads*. También son destacables asignaturas de la rama de Estructura y Arquitectura de Computadores, en las que se enseñan los conceptos base para la comprensión del funcionamiento de procesadores, jerarquías de memoria, etc. Por otro lado, la asignatura de Programación Consciente de la Arquitectura me ha formado en aspectos relacionados con la optimización y el análisis del rendimiento de las aplicaciones. Finalmente, la asignatura de Multiprocesadores también ha sido básica para la comprensión de una arquitectura con la de Cell BE, así como la programación en entornos paralelos.

Líneas de trabajo futuro

Esperamos acabar de portar la aplicación de FFT-3D al modelo de programación de Cell Superscalar, y realizar pruebas de rendimiento para analizar la eficiencia al programar con este *framework*.

Respecto a la versión con *function offloading*, está previsto realizar un cambio que en el análisis de resultados se vio que podría mejorar el rendimiento. Las transferencias de tamaño más pequeño que 1024 Bytes, que ahora mismo se realizan con DMA *get* de elementos DMA, podrían realizarse con listas DMA mediante técnicas de *gather/scatter*, tal y como se desprende del estudio hecho en [7].

Por otra parte, actualmente existe una librería de control de los SPEs más actualizada, *libspe2*. Por esto, sería deseable reescribir nuestra aplicación para esta nueva versión de la librería y ver qué ganancia se obtiene, tanto a nivel de desarrollo como a nivel del rendimiento de la aplicación.

Cuadro A.1 – viene de la página anterior

Núm. SPU		A	B
	comunicación	0.106214	0.086679
	Speed-up	3.5035	3.60432
8	1ª FFT-1D	0.013988	0.013964
	1ª transposición	0.01698	0.018047
	2ª FFT-1D	0.013702	0.013604
	2ª transposición	0.045596	0.026493
	3ª FFT-1D	0.013837	0.013378
	Total	0.106575	0.086958
	comunicación	0.098800	0.092037
	Speed-up	4.7804	5.13

Cuadro A.2: Resultados de tiempos para tamaño 128^3, para el mejor caso de cada versión. * es la versión que se realiza en dos pasos.

Núm. SPU		*	A	B
1	1ª FFT-1D	0.0218281	0.0116794	-
	1ª transposición	-	0.0051965	-
	2ª FFT-1D	-	0.0116227	-
	2ª transposición	0.014957	0.0057213	-
	3ª FFT-1D	-	0.0116229	-
	Total	0.036548	0.046765	-
	comunicación	0.007785	0.019346	-
	Speed-up	1	1	-
2	1ª FFT-1D	0.0109922	0.006127	0.006126
	1ª transposición	-	0.0029912	0.0032166
	2ª FFT-1D	-	0.00605	0.0060456
	2ª transposición	0.0074912	0.0031685	0.003046
	3ª FFT-1D	-	0.0060499	0.0060448
	Total	0.018639	0.023424	0.024173
	comunicación	0.003949	0.011381	0.009938
	Speed-up	1.9608	1.9964	1.9964
	1ª FFT-1D	0.005613	0.0030908	0.003187

Continúa en la siguiente página

Cuadro A.2 – viene de la página anterior

Núm. SPU		*	A	B
	1ª transposición	-	0.0019758	0.0019358
	2ª FFT-1D	-	0.0029886	0.0031148
	2ª transposición	0.0038879	0.0016718	0.0019394
	3ª FFT-1D	-	0.0029878	0.0031218
	Total	0.009462	0.013018	0.013054
	comunicación	0.002772	0.007633	0.007603
	Speed-up	3.8626	3.59233	3.6968
8	1ª FFT-1D	0.003028	0.001818	0.002055
	1ª transposición	-	0.001766	0.001789
	2ª FFT-1D	-	0.001691	0.001875
	2ª transposición	0.002306	0.001425	0.001987
	3ª FFT-1D	-	0.001683	0.001873
	Total	0.005146	0.008142	0.009623
	comunicación	0.002630	0.007287	0.006182
	Speed-up	7.1022	5.74367	5.0149

Cuadro A.3: Resultados de tiempos para tamaño 64^3, para el mejor caso de cada versión. * es la versión que se realiza en dos pasos.

Núm. SPU		*	A	B
1	1ª FFT-1D	0.0028511	0.0015931	-
	1ª transposición	-	0.000751	-
	2ª FFT-1D	-	0.0015203	-
	2ª transposición	0.0019067	0.0010562	-
	3ª FFT-1D	-	0.0015206	-
	Total	0.004814	0.006375	-
	comunicación	0.001033	0.002762	-
	Speed-up	1	1	-
2	1ª FFT-1D	0.0014999	0.0008533	0.0008894
	1ª transposición	-	0.0004194	0.0004154
	2ª FFT-1D	-	0.0007752	0.0008014
	2ª transposición	0.0009902	0.000605	0.0004128

Continúa en la siguiente página

Cuadro A.3 – viene de la página anterior

Núm. SPU		*	A	B
	3ª FFT-1D	-	0.000775	0.0008018
	Total	0.002528	0.003521	0.003397
	comunicación	0.000700	0.001679	0.001538
	Speed-up	1.9042	1.8105	1.8105
4	1ª FFT-1D	0.000873	0.0005585	0.0005874
	1ª transposición	-	0.0003192	0.0003104
	2ª FFT-1D	-	0.0004488	0.0004642
	2ª transposición	0.0005838	0.0004212	0.0003082
	3ª FFT-1D	-	0.0004484	0.0004648
	Total	0.001466	0.002185	0.002088
	comunicación	0.000625	0.001477	0.001382
	Speed-up	3.2837	2.9176	2.9455
8	1ª FFT-1D	0.000638	0.000473	0.000511
	1ª transposición	-	0.000297	0.000291
	2ª FFT-1D	-	0.000327	0.000351
	2ª transposición	0.00044	0.000362	0.000309
	3ª FFT-1D	-	0.000328	0.000349
	Total	0.001110	0.001881	0.001845
	comunicación	0.000704	0.001473	0.001368
	Speed-up	4.3369	3.3891	3.3334

Glosario

API Application Programming Interface. Conjunto de funciones proporcionadas por una librería para proporcionar abstracción al programar, 10

atomicidad Propiedad que establece que, en un entorno de programación concurrente o paralela, cada operación ocurrirá en un instante preciso de tiempo, sin que surjan inconsistencias sobre el orden en que han sido ejecutadas, 40

blade Servidor compacto que agrupa sólo el microprocesador, la memoria y los buses, mientras que la alimentación, tarjetas de red y otros componentes son externos, con el objetivo de añadir simplicidad al reducir espacio y ahorrar energía, 11

BSC Barcelona Supercomputing Center, Centro Nacional de Supercomputación, 11

DMA Direct Memory Access, sistema de control de transferencias entre memoria y dispositivos E/S sin intervención de la CPU., 5

función intrínseca Función que será tratada de manera especial por el compilador para traducirla por un conjunto de instrucciones predefinido. Las intrínsecas de Altivec, por ejemplo, se usan para declarar en tiempo de compilación operaciones vectorizables., 12

in place Referente a un programa, que no necesita memoria adicional aparte de los datos de entrada/salida, 21

localidad espacial Principio que establece que si se accede a un dato, existen altas probabilidades de acceder a datos cercanos a él, 37

RISC Reduced Instruction Set Computer, filosofía de diseño de CPU que apuesta por un juego de instrucciones reducido y simple., 5

SIMD Single Instruction Multiple Data, técnica que consigue paralelismo a nivel de datos, 5

SMP Symmetric multiprocessing. Arquitectura de multiprocesadores en la que se conectan varios procesadores a una única memoria principal compartida, 6

SMT Simultaneous MultiThreading, técnica para aumentar el nivel de paralelismo de los *threads*. Consiste en que *threads* del mismo o diferente programa comparten la *pipeline* de ejecución, 31

speed-up Ganancia obtenida en rendimiento de una ejecución a otra. Se puede expresar como factor multiplicador (2x, el doble) o en tanto por cien., 53

stride Número de posiciones de memoria entre un elemento y otro de un array, 36

Bibliografía

[1] AutoDock. http://autodock.scripps.edu/.

[2] OpenMP. http://www.openmp.org/drupal/.

[3] Worldwide Protein Data Bank. http://www.wwpdb.org/.

[4] Apple Computer. AltiVec Instruction Cross-Reference. http://developer.apple.com/hardwaredrivers/ve/instruction_crossref.html.

[5] Arnd Bergmann. Spufs: The Cell Synergistic Processing Unit as a virtual file system. http://www.ibm.com/developerworks/power/library/pa-cell/ (25 junio 2005).

[6] Barcelona Supercomputing Center. "Cell Superscalar (CellSs) User's Manual", 1.2 edición (mayo 2007).

[7] Daniel Jiménez González, Xavier Martorell y Álex Ramírez. Performance Analysis of Cell Broadband Engine for High Bandwidth Applications. *Performance Analysis of Systems & Software, 2007. ISPASS 2007* páginas 210–219 (abril 2007).

[8] E. Katchalski-Katzir, I. Shariv, M. Eisenstein, A.A. Friesem, C. Aflalo y I.A. Vakser. Molecular surface recognition: determination of geometric fit between proteins and their ligands by correlation techniques. *Proceedings of the Natlonal Academy of Sciences of the USA* **89**, 2195–2199 (1992).

[9] Eric W. Weisstein. Convolution. MathWorld–A Wolfram Web Resource. http://mathworld.wolfram.com/Convolution.html.

[10] Freescale Semiconductor. Información de Altivec. http://www.freescale.com/webapp/sps/site/overview.jsp?nodeId=0162468rH3bTdGmKqW5Nf2.

[11] Graham R Smith y Michael JE Sternberg. Prediction of protein-protein interactions by docking methods. *Current Opinion in Structural Biology* **12**, 28–35 (2002).

[12] Harald Servat, Cecilia González, Xavier Aguilar, Daniel Cabrera y Daniel Jiménez. Drug design issues on the Cell BE. (2007).

[13] HEIKE JAGODE. Fourier Transforms for the BlueGene/L Communication Network. Proyecto Fin de Carrera, University of Edinburgh (2006).

[14] HENRY A. GABB, RICHARD M. JACKSON Y MICHAEL J. E. STERNBERG. Modelling protein docking using shape complementarity, electrostatics and biochemical information. *Molecular Biology* (272), 106–120 (1997).

[15] IBM. Cell Broadband Engine Programming Tutorial (21 octubre 2005).

[16] IBM. Cell Broadband Engine SDK Libraries. Overview and Users Guide (agosto 2005).

[17] IBM. SPU C/C++ Language Extensions (20 octubre 2005).

[18] IBM. Cell Broadband Engine Programming Handbook (24 abril 2007).

[19] IBM SYSTEMS AND TECHNOLOGY GROUP. Cell Software Programming Model (4 diciembre 2006). Course Code: L2T1H1-11.

[20] J. A. KAHLE, M. N. DAY, H. P. HOFSTEE, C. R. JOHNS, T. R. MAEURER Y D. SHIPPY. Introduction to the Cell multiprocessor. *IBM J. Res. & Dev.* **49**(4/5), 589–604 (julio 2005).

[21] JAMES W. COOLEY Y JOHN W. TUKEY. An Algorithm for the Machine Calculation of Complex Fourier Series. *Mathematics of Computation* **19**(90), 297–301 (abril 1965).

[22] J.O. EKLUNDH. A Fast Computer Method for Matrix Transposing. *Transactions on Computers* **C-21**, 801– 803 (julio 1972).

[23] M. ELEFTHERIOU, B. G. FITCH, A. RAYSHUBSKIY, T. J. C. WARD Y R. S. GERMAIN. Scalable framework for 3D FFTs on the Blue Gene/L supercomputer: Implementation and early performance measurements. *IBM Journal of Research and Development* **49**(2/3), 457–464 (mayo 2005).

[24] MARIA ELEFTHERIOU, BLAKE FITCH, ALEKSANDR RAYSHUBSKIY, T.J. CHRISTOPHER WARD Y ROBERT GERMAIN. Performance Measurements of the 3D FFT on the Blue Gene/L Supercomputer. *Lecture Notes in Computer Science* **3648/2005**, 795–803 (2005).

[25] MARIA ELEFTHERIOU, JOSÉ E. MOREIRA, BLAKE G. FITCH Y ROBERT S. GERMAIN. A Volumetric FFT for BlueGene/L. *Lecture Notes in Computer Science* **2913/2003**, 194–203 (2003).

[26] P.E. BOURNE Y H. WEISSIG. "Structural Bioinformatics". Wiley-Liss (2003).

[27] PIETER BELLENS, JOSEP M. PEREZ, ROSA M. BADIA Y JESUS LABARTA. CellSs: A Programming Model for the Cell BE Architecture. *proceedings of the ACM/IEEE SC 2006 Conference* (noviembre 2006).

[28] STEVEN W. SMITH. "The Scientist and Engineer's Guide to Digital Signal Processing". http://www.dspguide.com/ (1997).

[29] THOMAS CHEN, RAM RAGHAVAN, JASON DALE Y EIJI IWATA. Cell Broadband Engine Architecture and its first implementation. A performance view. *Developer Works* (29 noviembre 2005).

[30] WILLIAM H. PRESS, SAUL A. TEUKOLSKY, WILLIAM T. VETTERLING Y BRIAN P. FLANNERY. "Numerical Recipes in C. The Art of Scientific Computing". Cambridge University Press, second edición (1992).