Laboratorio de Arquitecturas Avanzadas con Cell y PlayStation 3

Fernando Pardo y Jose A. Boluda
Departamento de Informática
Universitat de València
Avda. Vte. Andrés Estellés s/n
46.100 Burjassot - Valencia
Fernando.Pardo@uv.es

Resumen

El procesador Cell incluye en un único chip la mayoría de las estructuras arquitectónicas utilizadas hoy en día en computación, como la arquitectura SIMD, multicomputadores en chip, o la tradicional SISD superescalar. Constituye por tanto una buena plataforma para la enseñanza de arquitecturas avanzadas de computadores. Por otro lado, el procesador Cell es el corazón de las populares consolas PlayStation 3, por lo que es sencillo conseguir equipos de altas prestaciones con Cell a menor coste que un ordenador convencional. Presentamos aquí el clúster de 13 PlayStation 3, realizado en el Departamento de Informática de la Universitat de València, para la docencia de Arquitecturas Avanzadas tanto en grado como en Máster.

1. Arquitectura del procesador Cell

IBM, junto con Sony y Toshiba, crearon la arquitectura Cell con el reto de hacer frente al procesamiento de alto rendimiento requerido por un rango amplio de aplicaciones, incluyendo las consolas de videojuegos; todo ello con un reducido consumo y bajo coste [1]. El Cell, o Cell Broadband Engine Architecture (CBEA), fue creado a partir del análisis de numerosas aplicaciones científicas y de cálculo numérico matricial.

El Cell es un multicomputador heterogéneo compuesto por un procesador anfitrión tradicional (IBM 64-bit Power Architecture) y 8 procesadores independientes, especializados y basados en una arquitectura SIMD. Estos procesadores reciben el nombre de Synergistic Processor Element (SPE). En el sistema se combina la flexibilidad de un procesador tradicional con la potencia optimizada de los procesadores SIMD, ofreciendo además la capacidad de cálculo paralelo.

La Figura 1 muestra la arquitectura del procesador Cell. Los 8 SPE, junto con el PowePC processing Element (PPE), se conectan entre sí mediante el Element Interconnect Bus (EIB) en el propio chip. Este bus se conecta a la memoria y periféricos mediante dos interfaces externas.

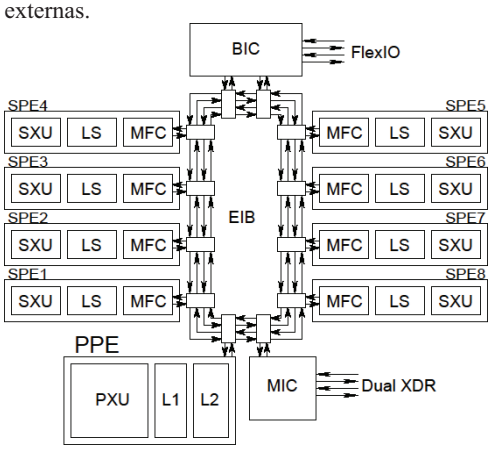

Figura 1. Arquitectura del procesador Cell

1.1. PowerPC Processing Element (PPE)

El procesador anfitrión del Cell lo forma un núcleo estándar basado en la arquitectura Power 64-bit de IBM a la que se le han añadido extensiones de instrucciones multimedia y SIMD. Tiene dos niveles de caché con 64 y 512 KB y reloj a 3,2 GHz. La implementación concreta de este núcleo se ha optimizado para mayor velocidad, menor área y menor consumo.

El PPE se encarga del programa principal de la aplicación, y como tal, es el encargado de repartir las tareas entre los 8 SPEs del sistema. Dado que no comparten memoria, el PPE debe descargar sobre los SPE tanto el propio código a ejecutar como los datos, aunque estos últimos también los puede cargar el SPE una vez empieza a ejecutar su propio código.

1.2. Element Interconnect Bus (EIB)

Los 9 elementos procesadores que componen el procesador Cell se encuentran interconectados mediante una red en anillo. Esta red dispone de 4 anillos de 16 bytes, lo que permite varias transferencias de forma simultánea. Este tipo de red en anillo está especialmente indicada para el procesamiento encauzado de tareas, de manera que el resultado de una etapa, resuelto en un SPE, pasa al siguiente SPE donde se calcula la siguiente etapa de forma segmentada.

Esta red de interconexión puede transmitir 96 bytes/ciclo, lo que supone más de 200 Gbytes/s para una frecuencia de reloj de 3,2 GHz.

Esta red anillo cuenta con 12 nodos tal como se muestra en la Figura 1. Ocho nodos los ocupan los SPE, un nodo conecta con el PPE, otro nodo con la memoria (MIC) y por último hay dos nodos dedicados a la comunicación con los periféricos (BIC).

1.3. Synergistic Processor Element (SPE)

Los elementos de proceso SPE son los que proveen el mayor rendimiento al procesador Cell. Cada uno presenta una arquitectura SIMD de 32 bits, lo que permite la ejecución paralela de 2 operaciones con números flotantes de doble precisión, 4 con flotantes de precisión simple o enteros, y 8 con enteros cortos.

Cada SPE cuenta con una memoria local interna de 256 KB donde se almacenan el programa y los datos. La transferencia de datos entre el SPE y la memoria principal se realiza mediante transferencias DMA (Direct Memory Access). Tiene un dispositivo específico para transferencia DMA que puede llegar a 16 peticiones concurrentes. Cada transacción DMA puede contener desde 1 byte hasta 16 KB como máximo.

En la Figura 1 se muestran los componentes básicos de cada SPE: La unidad de procesamiento SXU se ocupa de la ejecución de instrucciones escalares y vectoriales. La memoria local (LS) de 256 KB almacena datos y programa y es de acceso muy rápido por lo que no precisa de caché. Por último, la interfaz con el anillo de comunicaciones (MFC) asegura la realización óptima de las transacciones entre los elementos de proceso, con la memoria, con los periféricos y con el procesador anfitrión.

1.4. Interfaz con la memoria y periféricos

El procesador cuenta con interfaces independientes para la memoria y periféricos. La memoria se conecta directamente al bus de interconexión interno del Cell mediante el módulo MIC. Los periféricos tienen todavía mayor ancho de banda al contar con dos nodos paralelos en el anillo de interconexión. El módulo BIC se ocupa de esta interconexión con los elementos periféricos.

2. Clúster con PlayStation 3

La utilización del procesador Cell por parte de Sony para las PlayStation 3 (PS3), brinda la posibilidad de montar un sistema de computación de altas prestaciones de forma sencilla y a un coste realmente bajo [2][3].

Las PS3, además del Cell, incorporan una tarjeta gráfica Nvidia RSX de alta definición con 256 MB de memoria de vídeo GDDR. La PS3 tiene un disco duro estándar, Bluetooth, Ethernet Gigabit, Wifi, DVD Blu-ray, etc.

Una limitación importante de la PS3 para supercomputación se encuentra en su escasa memoria principal (256 MB) que limita fuertemente la cantidad de datos que las aplicaciones pueden almacenar en memoria.

Otra limitación aparece en el propio Cell, pues tiene uno de sus SPE deshabilitado para ahorrar recursos, mientras que otro es utilizado en exclusiva por el firmware de Sony y no está disponible al usuario. Esto significa que las aplicaciones sólo pueden hacer uso de los 6 SPE restantes y del PPE.

En 2009 la PS3 fue remodelada pasando a llamarse PS3 Slim. Este nuevo modelo presenta un menor consumo y menor calentamiento, es más

ligero, tiene un disco duro con mayor capacidad, etc. En general se trata de un modelo con mejores características gracias en buena parte al paso del Cell a la tecnología de 45 nm. Lamentablemente, este nuevo modelo no permite la instalación de un sistema operativo huésped, por lo que no se puede usar para hacer un supercomputador.

Desde el firmware 3.21 tampoco las PS3 originales pueden instalar un Sistema Operativo huésped. Al parecer, se habría encontrado una vulnerabilidad que permitiría el acceso privilegiado a la consola desde un Sistema Operativo huésped. Si no se actualiza el firmware, no se puede acceder a la red de Sony ni a otros servicios, pero esto no es un problema si se utiliza la consola para supercomputación.

El clúster implementado en el Departamento de Informática de la Universitat de València cuenta con 13 PS3 interconectadas mediante un encaminador Ethernet común. La Figura 2 muestra una parte del clúster. En principio este clúster se realizó con el objetivo de servir en las prácticas de laboratorio de la asignatura de Arquitecturas Avanzadas de la Ingeniería Informática, aunque actualmente también se utiliza en las prácticas del máster de Computación Avanzada y Sistemas Inteligentes impartido en el Departamento de Informática.

Figura 2. Detalle del Clúster de PS3 de la Universitat de València

2.1. Sistema operativo

Las PS3 originales, no las actuales Slim ni cualquiera con firmware>3.15, permiten la instalación sencilla de un sistema operativo huésped. Se han realizado diversas adaptaciones de Linux para PS3 siendo la distribución Yellow Dog [4] una de las primeras y la más popular para este sistema. Otras distribuciones como Fedora, Ubuntu y OpenSuse también se han portado con éxito, cada una con sus limitaciones particulares.

La distribución utilizada en el clúster ha sido Yellow Dog pues se instala muy fácilmante en estos sistemas y no requiere de mayores ajustes.

El Sistema Operativo huésped se ejecuta siempre bajo la supervisión del software propietario de la consola bajo una máquina virtual (por eso hay una SPE reservada). Esto implica que ciertas características del sistema no están disponibles para el huésped. Por ejemplo, no hay acceso completo a la potencia de procesamiento de vídeo ofrecido por la tarjeta gráfica incorporada. Sin embargo, y a pesar de tratarse de una virtualización, la capacidad de procesamiento del Cell permanece intacta, que es lo que interesa para el sistema de supercomputación.

3. Aumento de rendimiento

Un clúster de PS3 como el mostrado anteriormente es interesante para la enseñanza de arquitecturas de altas prestaciones pues conjuga diferentes paradigmas de computación en un mismo sistema:

- Arquitectura vectorial SIMD: gracias a que cada SPE posee una potente unidad vectorial SIMD se pueden realizar pruebas de rendimiento con cálculos vectoriales. Además, aunque los SPE son independientes, se les puede programar para realizar la misma operación tal como se haría en un procesador matricial, pudiendo explotar la arquitectura SIMD a varios niveles.
- Multicomputador SIMD: tal como se comentaba en el caso anterior, los SPE, aunque independientes, se les puede poner el mismo código de manera que se comporten como un procesador matricial de 6 elementos (2 SPEs están reservados). Se puede aplicar el paralelismo inherente del cálculo vectorial pero a nivel de los SPE.
- Segmentación gruesa. Dada la disposición de los SPE en anillo, se puede implementar un cauce segmentado de manera que cada SPE realiza una etapa del algoritmo.

- Multicomputador OpenMP/MPI: dado que se trata de un clúster con 13 PS3, se puede utilizar la OpenMP para realizar aplicaciones paralelas como si de un sistema de memoria compartida se tratase, o se puede usar MPI para programación por paso de mensajes.

En las prácticas de laboratorio se trata de visualizar el aumento de rendimiento debido sobre todo a las características SIMD de la arquitectura del Cell. La utilización de OpenMP y MPI se realiza en otros sistemas pero se pretende incorporar estas herramientas también a las PS3.

4. Programación de las PS3

Cualquier aplicación para ser ejecutada en el Cell debe especificar tanto el programa correspondiente al procesador anfitrión (PPE) como los programas de cada coprocesador (SPE). Los códigos de cada uno de estos programas se realizan y compilan inicialmente por separado. El PPE es el encargado de copiar en la memoria de cada SPE el programa que tienen que ejecutar y es el encargado igualmente de lanzar la ejecución de cada SPE. Esto implica que los ficheros objeto de todos los programas se deben unir en uno solo, que será el que ejecute el PPE y que contendrá el código que el propio PPE le pasará a los SPE.

Los programas para la arquitectura SPE se compilan con una versión específica del **`gcc`** llamada **`spu-gcc`**. Con esto se genera el objeto del programa a ejecutar en el SPE. Este programa hay que prepararlo para enlazarlo junto con el programa que ejecuta el procesador principal PPE. Esta preparación se realiza mediante la herramienta **`ppu-embedspu`** especificando un nombre para el código para que el programa del PPE pueda referenciarlo. El resultado es un fichero objeto que se compila con el **`gcc`** normal junto con el fuente del programa del PPE.

Los detalles específicos de la programación tanto del PPE como del SPE están fuera del alcance de este artículo. Existe numerosa documentación al respecto y no es complicado a partir de un ejemplo [3]. Como resumen, el programa del PPE se encarga de crear los contextos de ejecución de cada SPE a los que les traslada el programa a ejecutar. Posteriormente crea un hilo por cada SPE y cada hilo se encarga de lanzar el programa del SPE. El PPE puede realizar mientras tanto otras tareas o esperar a que los hilos acaben cuando termina la ejecución del programa de los SPE.

Los elementos de proceso no comporten la memoria principal a la que solamente el PPE tiene acceso por dirección. Los SPE sólo pueden direccionar su memoria local por lo que cualquier otro dato que necesiten debe ser escrito previamente en su memoria. Existen diferentes mecanismos para llevar y traer datos entre la memoria principal y la local, y también entre las memorias locales de los propios SPE. El mecanismo más simple, utilizado en las sesiones de laboratorio, es el que utiliza transacciones DMA para leer/escribir de la memoria principal a las memorias locales. Estas transacciones las puede realizar tanto el PPE como el SPE. En nuestro caso particular son los SPE los que acceden a la memoria principal mediante DMA para leer/escribir los datos, de esta manera se aprovecha mejor el ancho de banda y capacidad paralela del anillo de comunicaciones que interconecta todos los elementos del Cell.

Existen numerosos mecanismos de comunicación en el Cell, como los *mailboxes* por ejemplo. Estos mecanismos permiten hacer transferencias entre SPE específicos para implementar algoritmos segmentados donde cada SPE realiza una etapa del algoritmo. Dado que la red de interconexión es un anillo, cada SPE de una etapa puede comunicarse con la etapa siguiente en paralelo con el resto de transacciones en el anillo. Este tipo de arquitectura segmentada resulta igualmente interesante para ser incorporada a futuras sesiones de laboratorio.

4.1. Programación SIMD

Cada SPE tiene una unidad vectorial SIMD con su propio juego de instrucciones vectorial. Estas operaciones actúan sobre un tipo especial de datos llamado "**`vector`**". De hecho, las operaciones comunes como multiplicación (*), suma (+), etc., están sobrecargadas para el tipo "**`vector`**" de manera que son compiladas directamente a instrucciones SIMD aunque no se indique explícitamente.

Las variables se pueden declarar directamente como tipo vector, pero hay un uso todavía más interesante de este tipo, pues se puede referenciar como "**`vector`**" al típico puntero a escalares de

C (usado normalmente para especificar un vector en un procesador escalar).

Por ejemplo, se puede definir un vector para un procesador escalar como `float a[8];` y trabajar con él de forma escalar. Si ahora se quiere utilizar esos mismos 8 elementos como vectores en operaciones SIMD, sin copiarlos, definiríamos una nueva variable como `vector float *va=(vector float *)a;` dado que un vector de tipo `float` tiene 4 elementos escalares, el puntero vectorial `va` anterior apuntaría a dos vectores de 4 elementos cada uno. En estos ejemplos se están obviando las necesidades de alineamiento de la memoria local de los SPE (los vectores deben estar alineados a 16 bytes en memoria).

Las instrucciones SIMD de los SPE trabajan siempre con vectores cuyo tamaño es fijo e igual a 16 bytes (128 bits). Esto quiere decir que el número de elementos en el vector va a depender del tipo de datos que contenga. Así por ejemplo, los vectores de byte tienen 16 elementos, los de enteros cortos 8 elementos, los de enteros y flotantes 4 elementos, y los de flotantes dobles o enteros largos dos elementos. Las operaciones SIMD siempre operan con un vector, de manera que si sumamos dos vectores de flotantes se realizarán 4 sumas en paralelo, mientras que si sumamos reales dobles sólo se realizan dos sumas de forma simultánea.

En la realización de los programas se les solicita a los estudiantes que pongan explícitamente las instrucciones SIMD en lugar de dejar esta tarea al compilador. El resultado es el mismo pero de esta manera los estudiantes aprenden a distinguir mejor las instrucciones vectoriales de las escalares. Para la suma vectorial utilizan la instrucción `spu_add(a,b)`, para la multiplicación `spu_mul(a,b)` y para la suma combinada con la multiplicación `spu_madd(a,b,c)` (a*b+c). Dado que no hay instrucciones para multiplicar escalar por vector, hay que pasar el escalar a vector con `spu_splats(s)`.

5. Casos de estudio

En las prácticas de laboratorio se pretende que el estudiante aprenda las implicaciones del cálculo vectorial en la programación utilizando arquitecturas SIMD y de varios elementos de proceso con memoria local. Se implementan dos casos típicos de cálculo vectorial realizando primero una implementación escalar, repartiendo la tarea entre varios procesadores y posteriormente se hace uso de las instrucciones vectoriales SIMD para acelerar todavía más la ejecución de la aplicación.

En todos los casos se deben realizar estudios del aumento del rendimiento con la utilización de más o menos elementos de proceso (hasta un máximo de 6 SPE) y con la utilización de instrucciones SIMD.

5.1. Operación simple sobre un vector

Es de los casos más simples de ejecución paralela que se pueden plantear y resulta adecuado para que los estudiantes se familiaricen con la programación paralela de las PS3, que es el objetivo de esta primera sesión. En este caso concreto se parte de un vector de miles de elementos y la operación a realizar consiste en sumar 1 a cada elemento. Primero reparten el vector entre los elementos de proceso a utilizar (hasta un máximo de 6) y posteriormente convierten ese vector en un tipo de datos vectorial sobre el que aplican las instrucciones vectoriales. Con todo esto se estudia el aumento de rendimiento y se sacan conclusiones sobre la escalabilidad de la arquitectura utilizada.

5.2. Multiplicación SIMD de matrices

La multiplicación de matrices es una de las operaciones más utilizadas en cálculo vectorial. Al mismo tiempo ofrece múltiples posibilidades de paralelización dependiendo de la arquitectura, número de procesadores, etc.

En nuestro caso los estudiantes deben realizar una implementación que explote sobre todo las instrucciones SIMD vectoriales de la arquitectura Cell. Para que este aprovechamiento sea el mayor posible, todas las instrucciones que intervienen en la multiplicación deben ser vectorizables.

La implementación típica de un bucle que recorre las filas y columnas de las matrices multiplicándolas entre sí para luego sumar sus elementos, no es vectorizable de forma directa: mientras que la multiplicación de fila por columna es una operación típicamente vectorial, la suma de

los elementos del vector resultado no siempre es vectorizable pues cada elemento del vector se encuentra en un elemento de procesado diferente y no suele ser posible sumar todos de una vez. Por otro lado, el acceso a filas se realiza de forma consecutiva en memoria (si se ha ordenado por filas la matriz en memoria) sin embargo, el acceso a una columna implica acceder a elementos separados entre sí, y dependiendo del tipo de memoria, separación, etc., puede ocurrir que se produzca una disminución del rendimiento.

Para evitar estos problemas hay una forma de multiplicar matrices en la que sólo intervienen operaciones vectoriales y además el acceso se realiza siempre a elementos consecutivos en memoria, lo que permite aprovechar los mecanismos de acceso segmentado que pudiera tener. Este mecanismo se explica a continuación.

Supongamos que se quiere multiplicar dos matrices A y B para obtener una matriz C. El algoritmo cogería el primer elemento de la primera fila de A y lo multiplicaría por toda la primera fila de B (operación escalar por vector). El vector resultante contendría los primeros términos de la primera fila de C. A continuación se toma el segundo elemento de la primera fila de A y se multiplica por la segunda fila de B sumando vectorialmente el resultado al anterior que se había obtenido. De esta manera se siguen sumando resultados hasta llegar al último elemento de la primera fila de A que se multiplica por la última fila de B y con cuyo resultado, sumado al que se ha acumulado hasta este momento, contiene el vector con la primera fila de C. Estas operaciones se repetirían para cada fila de A, obteniendo cada fila de C. Resumiendo, se multiplica cada elemento de una fila de A por cada una de las filas de B acumulando el resultado hasta obtener la fila correspondiente de C. Las operaciones realizadas son de escalar por vector, que es una operación vectorial, y la suma de vectores que también es una operación vectorial. Las únicas operaciones escalares son las propias de los bucles que recorren las filas. Con este algoritmo, el acceso se realiza siempre por filas, es decir, elementos consecutivos en memoria, por lo que se pueden aprovechar los mecanismos de memoria entrelazada que pueda haber implementados.

Esta forma de multiplicar matrices aprovecha el procesado vectorial de cada SPE, pero además se dispone de 6 SPEs entre los que repartir la tarea. Dado que se accede por filas, la estrategia más sencilla consiste en repartir todas las filas de la matriz entre los elementos de proceso, de manera que cada SPE se encargue de una parte de todas las filas. Para esto, se debe pasar a cada SPE (no hay memoria compartida por lo que los datos hay que enviarlos a cada SPE) las filas de A que le correspondan y toda la matriz B pues todos los elementos de una fila de A se multiplican por todas las filas de B. El resultado que devuelve será el de las filas de C que le corresponda, que justo son las mismas filas que se hayan pasado de A.

6. Resultados obtenidos

En el laboratorio los estudiantes implementan tanto una operación simple sobre un vector lineal como una multiplicación de matrices. En ambos casos deben calcular los tiempos de ejecución y realizar estudios de aumento del rendimiento y escalabilidad, tanto con el uso de varios elementos de proceso como con el de instrucciones SIMD.

En este artículo se muestran sólo los resultados obtenidos con el ejemplo de la multiplicación de matrices por ser un problema más interesante. Además, los resultados en cuanto a escalabilidad y aumento del rendimiento son muy similares en ambos casos.

Las matrices a multiplicar son cuadradas con 60x60 elementos de tipo flotante simple (**`float`**). Se ha preferido utilizar el flotante simple pues cada vector tiene 4 elementos en lugar de 2 del flotante doble, de esta manera el aumento del rendimiento es más significativo.

Para calcular el tiempo de ejecución se ha utilizado la función **`times`** de C (**`sys/times.h`**). Como tiene una resolución de décimas de segundo, muy por encima del tiempo de ejecución de la multiplicación de matrices, se ha repetido el código del SPE 20.000 veces. Se ha repetido igualmente el experimento varias veces para evitar transitorios del sistema operativo observándose que el resultado no varía.

6.1. Utilización de varios elementos de proceso

En esta parte se busca utilizar varios elementos de proceso para aumentar el rendimiento de una aplicación que en este caso es la multiplicación de matrices. No se utilizan todavía las instrucciones

SIMD para distinguir las diferentes contribuciones en el aumento del rendimiento.

La Tabla 1 muestra los tiempos de ejecución y el aumento de rendimiento (aumento de la velocidad con respecto de la ejecución con un único procesador) dependiendo del número de SPE utilizados.

Num. SPE	Tiempo (s)	Aumento Rend.
1	216,45	1,000
2	108,26	1,999
3	72,19	2,998
4	54,17	3,996
5	43,34	4,994
6	36,14	5,989

Tabla 1. Tiempos de ejecución y aumento de rendimiento sin utilizar SIMD

Estos resultamos muestran que el aumento del rendimiento coincide prácticamente con el ideal. Sólo se nota una ligera, casi imperceptible, disminución con el aumento del número de procesadores. Se puede concluir que esta aplicación en esta arquitectura tiene una escalabilidad casi perfecta, si bien es verdad que el número de procesadores considerado es realmente muy bajo y los problemas de escalabilidad cuando no se comparte memoria, como es el caso, aparecen con cientos de procesadores.

6.2. Vectorización SIMD del código

En esta ocasión se sustituyen las instrucciones escalares por vectoriales.

Los resultados obtenidos de tiempo de ejecución así como del aumento del rendimiento se muestran en la Tabla 2.

Num. SPE	Tiempo (s)	Aumento Rend.
1	53,41	1,000
2	26,75	1,997
3	17,86	2,990
4	13,41	3,983
5	10,75	4,968
6	8,97	5,954

Tabla 2. Tiempos de ejecución y aumento de rendimiento utilizando SIMD

También se observa una escalabilidad casi perfecta, aunque ligeramente peor que la del caso anterior. La Figura 3 muestra una gráfica con los aumentos de rendimiento tanto utilizando SIMD como si no.

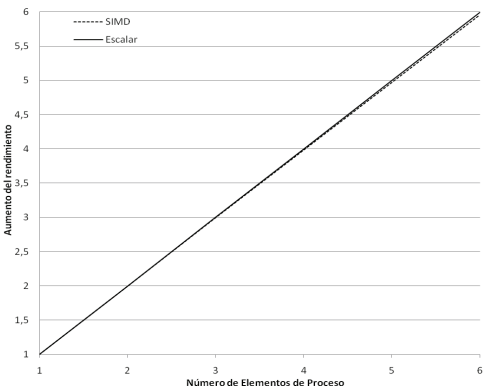

Figura 3. Aumento de rendimiento tanto SIMD como escalar

En la Figura 3 se observa que en ambos casos el aumento del rendimiento coincide casi exactamente con el ideal, sin embargo, es difícil apreciar las diferencias entre la escalabilidad del uso del escalar frente al SIMD. La gráfica en Figura 4 muestra el porcentaje de desviación del aumento de rendimiento respecto del ideal (es decir, respecto del número de procesadores).

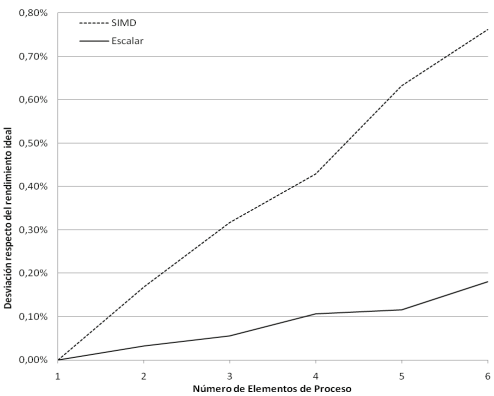

Figura 4. Desviación respecto del aumento de rendimiento idea para SIMD y escalar

En la gráfica de la Figura 4 se aprecia con mayor detalle que en ambos casos el aumento de rendimiento se aleja del ideal con el aumento del

número de procesadores. También se observa que este alejamiento es mayor en el código SIMD que en el escalar. No es raro que un sistema con procesadores más potentes escale peor que otro con procesadores más lentos, pues los costes de comunicaciones y ajenos al procesador son iguales en ambos casos pero ocupan un porcentaje mayor del tiempo total en el caso de procesadores rápidos. En cualquier caso la escalabilidad observada es muy buena pues en el peor caso, el alejamiento del rendimiento respecto del ideal está por debajo del 1%.

Faltaría comprobar si por el hecho de utilizar instrucciones SIMD que operan con vectores de cuatro elementos, la velocidad se multiplica por cuatro. Para ello se ha calculado el aumento de rendimiento para cada número de procesadores utilizados, arrojando los resultados mostrados en la Tabla 3.

N	1	2	3	4	5	6
S	4,053	4,047	4,042	4,040	4,032	4,029

Tabla 3. Aumento de rendimiento vectorial SIMD

Los resultados muestran en este caso una escalabilidad perfecta pues el aumento de rendimiento está incluso ligeramente por encima de 4. No es raro que el rendimiento sea incluso un poco mayor que el ofrecido por las propias instrucciones vectoriales, pues hay otros aspectos beneficiosos de la utilización de instrucciones vectoriales, como por ejemplo la reducción en el número de iteraciones de los bucles.

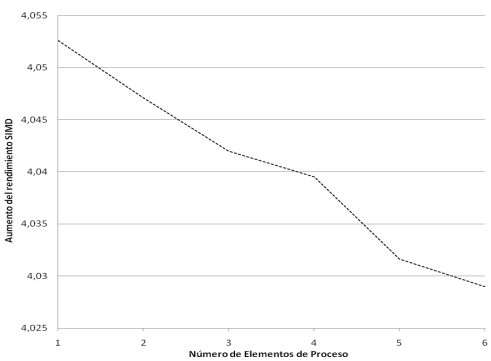

Figura 5. Aumento del rendimiento vectorial SIMD

En la Figura 5 se muestra gráficamente este aumento de rendimiento vectorial para cada número de SPE utilizados. Vemos que el aumento de rendimiento disminuye muy ligeramente con el número de procesadores, básicamente por la ligera peor escalabilidad de las instrucciones SIMD frente a las escalares tal como se ha explicado antes. En cualquier caso vemos que esta variación está entre 4,029 y 4,053 lo que supone menos del 0,6% y además siempre por encima del ideal.

7. Conclusiones

Se ha mostrado la utilidad del procesador Cell de las PS3 para la enseñanza de Arquitecturas Avanzadas de Computadores. Se han mostrado y explotado algunas de las ventajas arquitectónicas de la arquitectura Cell como son el procesamiento paralelo con memoria local y la utilización de instrucciones vectoriales SIMD. Quedan otros aspectos como la implementación de algoritmos segmentados en el Cell o la programación del propio Clúster (MP y MPI) para futuros trabajos y ampliación de las sesiones de laboratorio. Los resultados de las experiencias realizadas muestran una escalabilidad prácticamente igual a la ideal con diferencias inferiores al 1% incluyendo los costes de comunicaciones, tanto en el uso de diferentes procesadores como en el de instrucciones vectoriales. La utilización de sistemas como las videoconsolas motiva a los estudiantes y dinamiza la enseñanza de las arquitecturas de computadores.

Agradecimientos

Este trabajo ha sido posible gracias al Departamento de Informática de la Escuela Técnica Superior de Ingeniería de la Universitat de València y al estudiante Ángel Expósito Sánchez de la titulación de Ingeniería Informática de este mismo centro.

Referencias

[1] Web de IBM sobre el procesador Cell. http://www.research.ibm.com/cell
[2] Clúster realizado por Frank Mueller. http://moss.csc.ncsu.edu/~mueller/cluster/ps3
[3] Web de Fernando Pardo sobre clúster PS3 de la Universitat de València. http://tapec.uv.es/pardo/ps3
[4] Yellow Dog Linux. http://www.ydl.net/