Programación de alto rendimiento en el procesador Cell: Aplicación a simulación de fluídos

Carlos H. González, Basilio B. Fraguela, Diego Andrade, [1] José A. García [2], Manuel J. Castro [3]

Resumen— **En el diseño de los procesadores modernos es cada vez más común incorporar varios núcleos de procesamiento. En algunas arquitecturas estos núcleos no son homogéneos, sino que están especializados en la realización de funciones específicas. Uno de los procesadores de este tipo más destacables es el Cell, creado por Sony, Toshiba e IBM, que incorpora en un solo chip un núcleo PowerPC común y hasta 8 núcleos aceleradores, llamados Synergistic Procesing Elements, que alcanzan una gran rendimiento en operaciones en punto flotante mediante procesamiento vectorial. En este trabajo estudiaremos este procesador y su rendimiento con aplicaciones científicas. Optimizaremos para el procesador Cell una simulación de un sistema de aguas someras con transporte de contaminantes utilizando un método de volúmenes finitos. Ésta es una aplicación áltamente paralelizable, al tiempo que utiliza operaciones sobre matrices y vectores de dimensión 4, con lo que se adapta a la arquitectura de este procesador.**

Palabras clave— **Computación de altas prestaciones, Método de volúmenes finitos, Vectorización, Paralelismo, Arquitecturas heterogéneas**

I. Introducción

La evolución en el proceso de diseño de nuevos microprocesadores ha llegado en los últimos años a un punto en el que es difícil seguir aumentando la frecuencia de reloj de un núcleo individual. Por ello, la tendencia actual es la de aprovechar el mayor nivel de integración para añadir más núcleos dentro de un mismo chip. Hay dos clases principales de procesadores multinúcleo: los homogéneos y lo heterogéneos. En los primeros todos los núcleos tienen idénticas características, por lo que pueden ejecutar los mismos programas y acceder a la memoria y los dispositivos de entrada y salida. Ejemplo de este tipo son los procesadores Core2 Duo o Quad de Intel. Por otro lado, los heterogéneos cuentan con núcleos diseñados para propósitos diferentes, con su propia arquitectura y juego de instrucciones. Normalmente, un núcleo de propósito general se combinará con varios núcleos optimizados para tareas específicas. El procesador más representativo de este tipo es el Cell.

El procesador Cell [1] –diseñado por el consorcio STI formado por Sony, Toshiba e IBM–, como procesador multinúcleo heterogéneo, contiene un núcleo principal de propósito general (PowerPC Processing Element, PPE) acompañado de una serie de núcleos que realizan la labor de unidades aceleradoras (Synergistic Processing Elements, SPEs), dotadas de una pequeña memoria local sobre la cual trabajan, y que se comporta como una memoria caché que debe ser operada explícitamente por el software. La programación sobre esta plataforma entraña una gran complejidad tanto a nivel de utilización óptima del juego de instrucciones como de gestión de las memorias locales, pero al mismo tiempo permite obtener un alto rendimiento. En este trabajo hemos estudiado las características de este procesador, haciendo hincapié en los aspectos que lo diferencian de los procesadores de propósito general, y lo hemos programado para resolver un problema científico que se adapta muy bien a la arquitectura de este procesador, como es el de la simulación de un sistema de aguas someras con transporte de contaminantes mediante un método de volúmenes finitos.

El método de volúmenes finitos es altamente paralelizable, puesto que presenta pocas dependencias de datos que limiten la distribución de los mismos entre distintas unidades de procesamiento. Además, se basa en gran cantidad de operaciones entre matrices y vectores pequeños. Por ello también existen implementaciones del método adaptadas para entornos clúster, juego de instrucciones SSE de Intel [2] y más recientemente sobre GPUs [3].

El procesador Cell permite aprovechar el paralelismo inherente al problema, gracias a sus múltiples SPEs basados totalmente en instrucciones vectoriales. Para realizar el desarrollo hemos seguido una serie de pasos: análisis del algoritmo, reestructuración de algunas operaciones para su vectorización, diseño de la distribución de datos y tareas, implementación sobre las SPUs y optimización de transferencias de memoria. El empleo óptimo de los recursos del procesador Cell de una PlayStation3 nos ha permitido acelerar el rendimiento en un factor de 32 respecto a la implementación de referencia del método de volúmenes finitos ejecutada en un Xeon. La mayor parte de la computación se realiza en los SPEs, dado que gracias a su diseño específico cada una de ellas ejecuta el algoritmo 5 veces más rápido que el PPE.

El resto del trabajo se organiza de la siguiente manera: en la sección II explicamos la arquitectura del procesador Cell. En la sección III describimos el método de volúmenes finitos, mientras que en la sección IV explicamos cómo se ha realizado la implementación del mismo y en la sección V mostramos

[1]Grupo de Arquitectura de Computadores, Dpto. de Electrónica y Sistemas, Univ. da Coruña, e-mail: {cgonzalezv,basilio,dcanosa}@udc.es.

[2]Área de Matemática Aplicada, Dpto. de Matemáticas, Univ. da Coruña, e-mail: jagrodiguez@udc.es

[3]Dpto. de Análisis Matemático, Univ. de Málaga, e-mail: castro@anamat.cie.uma.es

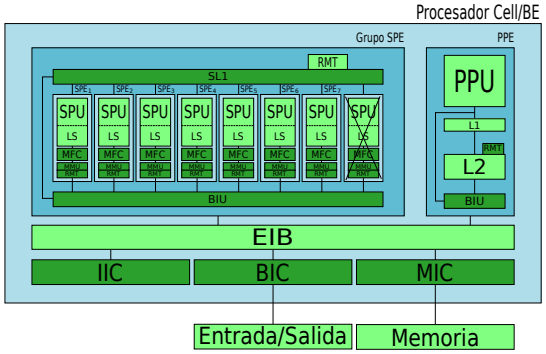

Fig. 1. Procesador Cell de la PlayStation3

los resultados obtenidos. Por último, en la sección VI detallamos las conclusiones y el trabajo futuro.

II. Arquitectura del procesador Cell

La arquitectura del procesador Cell está orientada a la computación de altas prestaciones. Así, está formado por cuatro unidades funcionales fundamentales, entre las que se incluyen un núcleo de propósito general encargado principalmente de gestión de tareas y un conjunto de núcleos aceleradores para el cómputo intensivo, además de por una serie de unidades que las complementan. Estas unidades se pueden ver en la figura 1.

- ✓ El PowerPC Processing Element (PPE), que contiene un núcleo PowerPC [4][5], llamado PowerPC Processing Unit (PPU), compatible con el juego de instrucciones PowerPC Instruction Set Architecture y con una unidad Altivec [6], que proporciona el juego de instrucciones vectorial y los registros vectoriales asociados. Dispone de 512KB de memoria caché L2 compartida para datos e instrucciones, así como dos cachés L1 separadas de datos e instrucciones de 32KB cada una.
- ✓ El Synergistic Processing Element (SPE), formado por un núcleo llamado Synergistic Processing Unit o (SPU) y 256KB de memoria local, que funciona como una caché gestionada por software. Esta unidad está totalmente basada en instrucciones SIMD, y en las implementaciones iniciales del Cell existen 8 en cada procesador.
- ✓ El Memory Flow Controler (MFC), encargado de gestionar las transferencias de DMA.
- ✓ El Internal Interrupt Controller (IIC), que maneja las interrupciones procedentes de los elementos internos del procesador.

El resto de componentes presentes en la figura 1 son:

- ✓ El Memory Interface Controller (MIC), que sirve de interfaz con la memoria RAM del sistema.
- ✓ El Bus Interface Control (BIC), que sirve de interfaz con los dispositivos de entrada y salida.
- ✓ Los Bus Interface Units (BIU), que controlan los accesos al bus.
- ✓ La memoria SL1, que cachea los datos de las operaciones DMA.
- ✓ El Element Interconnection Bus (EIB), un bus con estructura de anillo y de gran ancho de banda que conecta los SPEs y PPE entre ellos, así como con el exterior mediante el MIC y el BIC.

El entorno de desarrollo utilizado para este trabajo está centrado en una PlayStation3. Este sistema dispone de un procesador Cell a 3.2GHz, con 6 SPUs disponibles para el programador, reservándose una para el sistema operativo y estando otra desactivada en fábrica. Además, tiene 256MB de memoria RAM y un disco duro de 40GB.

Para programar este hardware se pueden barajar distintas alternativas, según nuestros requisitos y disponibilidad de recursos, algunos de ellos orientados exclusivamente al desarrollo de videojuegos. Para este trabajo hemos empleado el CellSDK 3.0 de IBM. Se trata de un entorno de desarrollo basado en Linux, que incluye herramientas estándar de este sistema como GCC, GDB o Eclipse, y se puede instalar sobre una PlayStation3 común. Hemos adoptado esta solución porque permite trabajar sobre el procesador Cell de forma muy económica, y la no accesibilidad al hardware gráfico de la consola no es relevante para los experimentos que vamos a realizar.

Con el CellSDK 3.0 se distribuyen una serie de librerías que realizan labores de alto nivel, como gestión de tareas u operaciones matemáticas. Sin embargo, para obtener el máximo rendimiento posible del procesador y estudiar su funcionamiento hemos utilizado los componentes fundamentales del SDK, implementando nosotros todas las funciones computacionales y de gestión de comunicaciones y tareas sobre las rutinas básicas del sistema. Así, la creación de procesos y posterior asignación a las SPUs se realiza mediante la librería pthreads estándar en UNIX, y la librería libspe2 incorporada en el CellSDK, que implementa funciones básicas de control de tareas. A la hora de realizar la vectorización de los cálculos, hemos empleado las intrínsecas vectoriales que proporciona el compilador [7].

La comunicación y transferencias de datos se pueden realizar básicamente de dos formas. Mediante comandos de DMA podemos mover datos entre la memoria RAM y las memorias locales de los SPEs, así como realizar transferencias entre dos memorias locales. Los comandos deben referirse siempre a direcciones de memoria alineadas a 16 bytes. Además, es recomendable que los accesos a la RAM se alineen a 128 bytes. La otra forma implica el uso de los mailboxes, registros disponibles en las SPUs que son accesibles por la PPU y organizados como colas FIFO de distinta profundidad. Desde la SPU, una de las colas es de escritura (la PPU lee) y tiene profundidad 1 y la otra es de lectura (la PPU escribe) y tiene profundidad 4. Mientras que los accesos por parte de la PPU pueden ser bloqueantes o no bloqueantes, los accesos desde la SPU sí son bloqueantes, lo cual puede utilizarse para la sincronización de procesos.

III. Modelo matemático y método de volúmenes finitos

Para simular el transporte inerte de un contaminante en un fluido, tomaremos como modelo el que resulta de acoplar las ecuaciones de aguas someras y una ecuación de transporte:

$$\begin{cases} \frac{\partial h}{\partial t} + \frac{\partial q_x}{\partial x} + \frac{\partial q_y}{\partial x} = 0, \\ \frac{\partial q_x}{\partial t} + \frac{\partial}{\partial x}\left(\frac{q_x^2}{h} + \frac{1}{2}gh^2\right) + \frac{\partial}{\partial y}\left(\frac{q_x q_y}{h}\right) = gh\frac{\partial H}{\partial x}, \\ \frac{\partial q_y}{\partial t} + \frac{\partial}{\partial x}\left(\frac{q_x q_y}{h}\right) + \frac{\partial}{\partial y}\left(\frac{q_y^2}{h} + \frac{1}{2}gh^2\right) = gh\frac{\partial H}{\partial y}, \\ \frac{\partial hC}{\partial t} + \frac{\partial q_x C}{\partial x} + \frac{\partial q_y C}{\partial y} = 0, \end{cases} \quad (1)$$

donde las incógnicas del problema son la altura de la columna del agua $h(\boldsymbol{x}, t)$, el caudal $q(\boldsymbol{x}, t) = (q_x(\boldsymbol{x}, t), q_y(\boldsymbol{x}, t))$ y la concentración de contaminante $C(\boldsymbol{x}, t)$. El caudal es el producto de $h(\boldsymbol{x}, t)$ por la velocidad del fluido,

$$q(\boldsymbol{x}, t) = h(\boldsymbol{x}, t)\boldsymbol{u}(\boldsymbol{x}, t) = h(\boldsymbol{x}, t)(u_x(\boldsymbol{x}, t), u_y(\boldsymbol{x}, t))$$

, $H(\boldsymbol{x})$ es la batimetría del fondo.

El sistema (1) puede escribirse como un sistema de leyes de conservación con términos fuente:

$$\frac{\partial W}{\partial t} + \frac{\partial F_1}{\partial x}(W) + \frac{\partial F_2}{\partial y}(W) = S_1(W)\frac{\partial H}{\partial x} + S_2(W)\frac{\partial H}{\partial y} \quad (2)$$

donde

$$F_1(W) = \left[q_x,\ \frac{q_x^2}{h} + \frac{1}{2}gh^2,\ \frac{q_x q_y}{h},\ q_x,\ C\right],$$
$$F_2(W) = \left[q_y,\ \frac{q_x q_y}{h},\ \frac{q_y^2}{h} + \frac{1}{2}gh^2,\ q_y,\ C\right],$$
$$S_1(W) = [0,\ gh,\ 0,\ 0]\,, S_2(W) = [0,\ 0,\ gh,\ 0]\,.$$

Para discretizar el sistema (2), descomponemos el dominio computacional en celdas o volúmenes de control, $V_i \subset \mathbb{R}^2,\ i = 1, \ldots, L$. Se usa la siguiente notación: dado un volumen finito V_i, $\mathcal{N}_i$ es el conjunto de índices j tales que V_j es vecino de V_i, E_{ij} es la arista común de dos celdas vecinas V_i y V_j, y $|E_{ij}|$ su longitud, $\boldsymbol{\eta}_{ij} = (\eta_{ij,x}, \eta_{ij,y})$ es el vector unitario normal al la arista E_{ij} y que apunta hacia la celda V_j.

La discretización del sistema (2) se lleva a cabo mediante un esquema de volúmenes finitos [8][9]. Si $W(\boldsymbol{x}, t)$ es la solución exacta denotaremos por ${W_i}^n$ una aproximación del promedio de la solución en el volumen V_i en t^n,

$${W_i}^n \simeq \frac{1}{|V_i|}\int_{V_i} W(\boldsymbol{x}, t^n)d\boldsymbol{x}. \quad (3)$$

donde $|V_i|$ es el área de la celda y $t^n = t^{n-1} + \Delta t$ es el instante de tiempo, siendo Δt el paso de tiempo.

Conocida la aproximación en el tiempo t^n, W_i^n, para avanzar en tiempo, se considera una familia de problemas de Riemann unidimensionales proyectados en la dirección normal a cada arista E_{ij}. Estos problemas de Riemann se linealizan mediante un esquema de Roe conservativo por caminos [10]. Finalmente las soluciones aproximadas de estos problemas de Riemann lineales se promedia en las celdas para obtener nuevas aproximaciones constantes a trozos de la solución. El esquema numérico queda como sigue:

$$W_i^{n+1} = W_i^n - \frac{\Delta t}{|V_i|}\sum_{j\in\mathcal{N}_\rangle} |E_{ij}|F_{ij}^-, \quad (4)$$

donde

$$F_{ij}^- = P_{ij}^-(A_{ij}(W_j^n - W_i^n) - S_{ij}(H_j - H_i)), \quad (5)$$

y donde $H_\alpha = H(N_\alpha), \alpha = i, j$, y A_{ij} y S_{ij} son las evaluaciones de

$$A(W, \boldsymbol{\eta}) = \frac{\partial F_1}{\partial W}(W)\eta_x + \frac{\partial F_2}{\partial W}(W)\eta_y,$$

y

$$S(W, \boldsymbol{\eta}) = S_1(W)\eta_x + S_2(W)\eta_y,$$

en $(W, \boldsymbol{\eta}) = (W_{ij}, \boldsymbol{\eta}_{ij})$, siendo W_{ij} el "estado intermedio" de Roe entre W_i^n y W_j^n.

La matriz P_{ij} se calcula como:

$$P_{ij}^- = \frac{1}{2}\mathcal{K}_{ij}\cdot(I - sgn(\mathcal{D}_{ij}))\cdot\mathcal{K}_{ij}^{-1},$$

donde I es la matriz identidad, $\mathcal{D}_{ij}$ y $\mathcal{K}_{ij}$ son las matrices de autovalores y autovectores de A_{ij}, respectivamente.

Puesto que el esquema numérico obtenido es explícito, para garantizar su estabilidad, es necesario imponer una condición de tipo CFL. En la práctica esta condición supone una restricción en el paso de tiempo; para avanzar de la iteración t^n a la t^{n+1} el paso temporal viene dado por:

$$\Delta t^n = \min_{i=1,\ldots,L}\left\{\frac{\sum_{j\in\mathcal{N}_i}|E_{ij}|\,\|D_{ij}\|_\infty}{2\gamma|V_i|}\right\},$$

siendo $\|D_{ij}\|_\infty$ es la norma infinito de la matriz $\mathcal{D}_{ij}$, es decir el máximo de los autovalores de la matriz A_{ij}.

IV. Implementación

La adaptación del programa original [2] optimizado para procesadores Intel se ha realizado siguiendo una serie de etapas diferenciadas: análisis del algoritmo, vectorización de las operaciones, diseño de la distribución de datos entre las SPUs y optimización.

A. Análisis del algoritmo

La implementación del método de volúmenes finitos con la que empezamos a trabajar sigue el diagrama de flujo de la figura 2. En la etapa de descomposición del dominio {1} se divide el conjunto total de volúmenes en subconjuntos disjuntos, que se distribuyen entre los diferentes procesos que realizarán el cálculo en paralelo. Posteriormente, un bucle itera en tiempo durante el periodo que se quiere simular. Para cada paso de tiempo se realizan tres procesos:

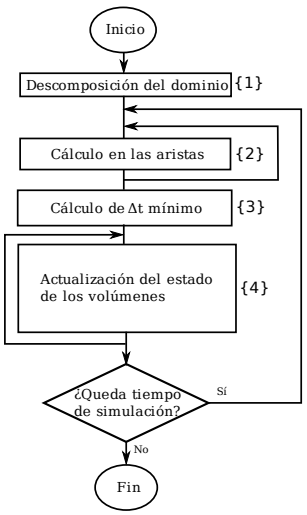

Fig. 2. Diagrama de flujo del algoritmo

en un bucle se calcula para cada arista el flujo que se produce a través de ella entre los dos volúmenes que comunica {2}. Este cálculo asocia a cada volumen un valor de paso de tiempo. Entre estos valores se buscará el mínimo {3}. Finalmente, con este valor mínimo de paso de tiempo se actualizarán los valores de las variables en todos los volúmenes {4}, para llegar al siguiente estado.

Para realizar este procesamiento los datos se almacenan en una estructura tipo grafo compleja, que contiene los datos correspondientes a cada volumen y arista, así como las relaciones entre ellos: qué dos volúmenes comunica cada arista, y qué aristas delimitan cada volumen. Esta estructura permite trabajar con mallas no estructuradas, con conectividades y volúmenes arbitrarios, aunque la implementación original de la que partimos sólo emplea mallas con volúmenes rectangulares.

El primer paso que hemos dado para transformar este programa de forma que utilice óptimamente el procesador Cell ha sido la vectorización de las operaciones fundamentales. Estas operaciones matemáticas entre vectores y matrices se encuentran encapsuladas en [2] en una serie de clases C++, que hacen uso de las librerías Integrated Performance Primitives de Intel. Para poder ejecutar el programa en un sistema PowerPC como es la PPU del Cell, hemos sustituido las llamadas a esta librería por implementaciones equivalentes que emplean las instrucciones vectoriales Altivec. En el sistema que nos ocupa se trabaja con cuatro variables –profundidad, concentración de contaminante y flujo en los ejes x e y–, lo que resulta en matrices cuadradas y vectores de dimensión 4, el mismo número de valores en punto flotante de simple precisión que se pueden acomodar en los registros vectoriales de 128 bits de la PPU y de las SPUs.

B. Distribución de datos

Como se ha mencionado antes, la estructura de datos que almacena la información de la malla de volúmenes finitos es compleja, derivada de la necesidad de soportar mallas no estructuradas. Sin embargo, en la práctica es posible emplear mallas estructuradas en una gran parte de los problemas, en las que los volúmenes rectangulares se ordenan en una matriz de dos dimensiones de forma regular. De esta forma, no es necesario almacenar información sobre las aristas, y a partir del índice de un volumen dentro de la matriz es trivial calcular las posiciones de sus vecinos. El ahorro de memoria es considerable, logrando una reducción de hasta el 75 % en los requisitos de memoria.

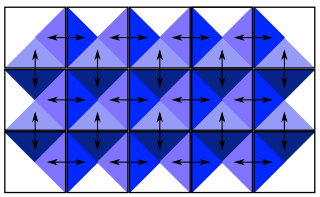

Fig. 3. Malla estructurada

La figura 3 ilustra la disposición de los volúmenes en una malla estructurada. Las flechas indican los flujos entre volúmenes que se han de calcular. Con este ordenamiento cambia la forma de operar en cada iteración del bucle en tiempo: en lugar de iterar sobre las aristas, se itera sobre los volúmenes, y para cada uno de ellos se calcula el flujo entre él y sus vecinos derecho e inferior. Además, esta estructura también facilita la división en bloques de la malla, para su distribución entre las distintas SPUs, y permite utilizar transferencias de DMA sobre conjuntos grandes de datos consecutivos, más eficientes que los movimientos de pequeñas cantidades de datos que corresponderían a cada volúmen individual sobre el que se trabajaría en una malla no estructurada.

La distribución de los volúmenes tiene lugar en dos niveles, desde el punto de vista de las SPUs: externo e interno. En el nivel externo se distribuyen las columnas de la malla por bloques entre las diferentes SPUs. Los volúmenes en la primera y última columnas asignadas a una SPU presentan el problema de no disponer de los datos de sus vecinos izquierdo y derecho, respectivamente. Para solucionarlo, optamos por utilizar regiones de solapamiento entre los bloques asignados a las SPU: cada una obtiene dos columnas de datos extra, una perteneciente al bloque anterior y otra perteneciente al siguiente. Debido a esto se calculan los datos de los extremos por duplicado, una vez en cada SPU participante en ese margen, pero esta duplicidad tiene un impacto muy reducido puesto que sólo representa el 0,21 % del total de operaciones.

El nivel interno se refiere a la división en bloques necesaria para procesar todos los datos en cada SPU. La memoria local de éstas es muy limitada, disponiendo solo de 256KB, lo que impide almacenar la malla completa. La solución que adoptamos fue la de dividir el bloque de columnas que se le asigna a cada SPU en subbloques de columnas, de las que se leerán un grupo de filas de cada vez. El número de columnas de estos subbloques es tal que cada fila se pueda obtener con una transferencia de DMA, que puede ser

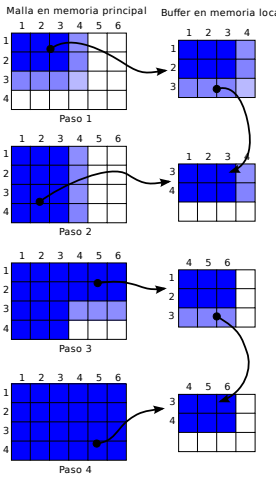

Fig. 4. Transferencias con regiones de solapamiento

como máximo de 16KB de datos. Para el cálculo en los volúmenes en los extremos del bloque es necesario nuevamente disponer del estado actual de volúmenes que pertenece a bloques vecinos, por lo que se vuelven a emplear regiones de solapamiento. En la figura 4 se ilustra cómo se realizan estas transferencias paso a paso. La obtención de cada bloque de volúmenes se hace con una fila y una columna extras, que contienen los datos de los vecinos de los volúmenes en los extremos. Cuando se almacenen en memoria principal contendrán resultados parciales del cálculo que se completarán con el cálculo en bloques siguientes. Como optimización, la fila de sombra es reaprovechada en la memoria local, colocándola al comienzo del buffer, de forma que no se almacena en la RAM para obtenerla de nuevo inmediatamente.

Con la distribución de datos propuesta no es necesario ningún tipo de comunicación explícita entre las distintas SPU, lo que redunda en un programa algo más sencillo y con menos sobrecargas por esperas de datos. La sincronización se realiza entre las SPUs y la PPU para obtener el valor de paso de tiempo mínimo: las primeras almacenan en su mailbox de salida el valor local obtenido, que será leído por la PPU y de los cuales ésta obtendrá el mínimo. Posteriormente escribirá este valor en los mailboxes de entrada de las SPUs, que lo leerán mediante una lectura bloqueante.

C. Optimización

Las transferencias de datos entre las memorias locales de las SPUs y la memoria principal se hacen siempre mediante transferencias de DMA, que es de naturaleza asíncrona. Es posible, por tanto, solapar la computación con estas transferencias para ocultar la latencia de memoria. Para ello utilizamos una técnica de multibuffering, en la que se definen una serie de buffers en la memoria local de las SPUs, de forma que el programa solicita que se traigan de memoria RAM datos que se usarán más adelante a ciertos buffers, solapándose dicha transferencia con la realización de computaciones sobre datos ya disponibles en otros buffers.

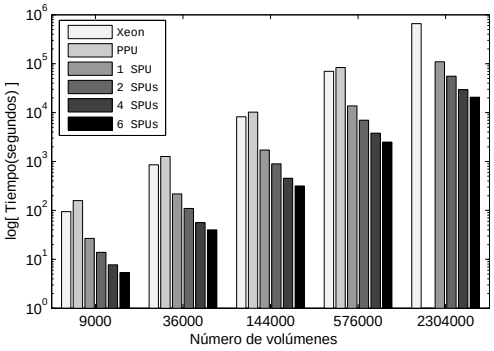

Fig. 5. Tiempos de ejecución

El número de líneas que se leen por adelantado está dimensionado de forma que no se produzcan nunca esperas por datos, y el número de buffers será el doble de este valor para almacenar también los datos calculados mientras son escritos de nuevo en memoria principal. El número de buffers en nuestro programa es 12, en cada uno de los cuales caben 256 volúmenes, lo que hace un total de 192KB. Este es el tamaño óptimo, pues oculta totalmente las latencias y aprovecha al máximo la memoria local de las SPUs dejando 64KB libres para el programa, las variables globales y la pila.

V. Resultados

Para la realización de los experimentos se ha utilizado una simulación de un sistema consistente en un canal rectangular con medidas 75×30 metros, cuyo fondo presenta una cresta dada por la función $B(x, y) = e^{-0{,}075(x-37{,}5)^2}$. Suponemos inicialmente un contaminante ocupando un círculo con centro ubicado en $(15, 15)$ y de radio $6m$, y un flujo inicial constante $q = (1, 0)$, así como una superficie libre constante de $3m$ de altura en su parte más profunda. Consideramos condiciones de contorno libres y un valor de CFL de 0,9, dejando evolucionar la simulación hasta un tiempo de 200 segundos. A fin de medir el rendimiento obtenido ante problemas de distinto tamaño y por tanto diversos grados de carga computacional, hemos realizado las medidas sobre cinco simulaciones, las cuales emplean mallas de 9000, 36000, 144000, 576000 y 2304000 volúmenes respectivamente. Como referencia hemos ejecutado el programa original en una máquina con procesador Xeon E5440 a 2,83GHz con 6MB de memoria caché L2.

La figura 5 muestra los tiempos de ejecución tanto del programa original en un Xeon, compilado con GCC 4.1.2 y las opciones de optimización -O3 -ffunrol-all-loops -ffast-math -ftree-vectorize y -msse2, como de nuestra implementación sobre la PPU utilizando distinto número de SPUs, con las opciones -O3 -ffunrol-all-loops y -ffast-math de GCC 4.1.1, agrupados según el tamaño de la malla. Las pocas dependencias de datos presentes en el algoritmo y la minimización de las comunicaciones en la implementación, permiten que el rendimiento aumente de forma casi lineal con el número de SPUs, con lo que tene-

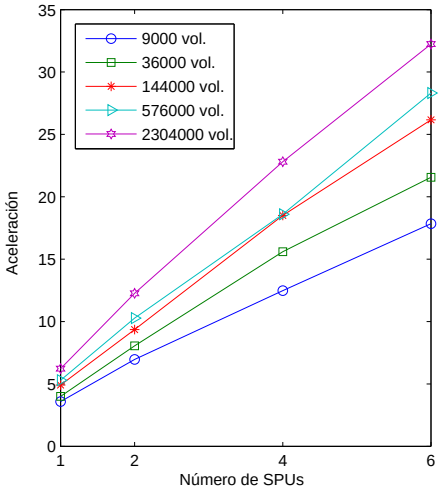

Fig. 6. Aceleraciones usando 6 SPUs frente al Xeon

mos un programa muy escalable. Nótese que la versión del programa original adaptada a la PPU no se ha podido ejecutar para la malla de 2304000 volúmenes, pues los 256Mb de memoria de la PlayStation3 no son suficientes para almacenar la información tal como estaba estructurada originalmente.

Los tiempos de ejecución en un Xeon E5440 a 2.83GHz están reflejados en la tabla I, mientras que la aceleración obtenida al usar el procesador Cell se muestran en la figura 6, donde se puede observar que nuestra implementación mejora sus aceleraciones a medida que se emplean más volúmenes. Así, se llega a una aceleración de 32x para 2304000 volúmenes utilizando las 6 SPUs de la PlayStation3.

Estos datos se refieren a la versión totalmente optimizada, que hace uso de multibuffering. El impacto de esta técnica en el programa es significativo, como se puede ver en la tabla II, dónde se aprecia que se obtiene una aceleración de 1,15 de media respecto a la implementación sin multibuffering, si bien este valor varía con el tamaño del conjunto de trabajo y el número de SPUs.

VI. Conclusiones

En este trabajo se ha partido de una implementación del método de volúmenes finitos para procesadores x86 para conseguir una adaptación al procesador Cell que consigue un rendimiento considerablemente superior. Ésto se ha conseguido mediante un estudio detallado de la arquitectura, de forma que hemos logrado unos tiempos hasta 32 veces mejores.

En cuanto a líneas de trabajo futuro, hay varios puntos que se pueden considerar, tales como considerar la paralelización entre varios procesadores Cell (bien sobre memoria compartida o distribuida), la implementación de versiones de la aplicación que operen con valores en punto flotante de doble precisión (lo cual es necesario para problemas de una mayor dimensionalidad) o el desarrollo de una implementación genérica que soporte mallas no estructuradas de forma eficiente.

TABLA I

Tiempos de referencia en el Xeon E5440@2.83GHz

Volúmenes	Tiempo (s)
9000	94.36
36000	857.20
144000	8196.80
576000	70014.18
2304000	657686.36

TABLA II

Aceleración de la versión con multibuffering sobre la versión sin multibuffering

Volúmenes \ SPUs	1	2	4	6
9000	1,12	1,16	1,19	1,23
36000	1,15	1,14	1,21	1,14
144000	1,18	1,15	1,12	1,10
576000	1,18	1,16	1,07	1,05
2304000	1,20	1,18	1,11	1,05

Agradecimientos

Este trabajo ha sido financiado por la Xunta de Galicia bajo el proyecto INCITE08PXIB105161PR y por el Ministerio de Ciencia e Innovación con fondos FEDER de la Unión Europea (proyecto TIN2007-67537-C03-02 y proyectos MTM2006-08075, MTM2007-67596-C02-01). Agradecemos a la Red Gallega de Computación de Altas Prestaciones (Red G-HPC) por promover colaboraciones interdisciplinares entre grupos de la red, así como los comentarios y sugerencias del profesor Ramón Doallo.

Referencias

[1] IBM, Sony, and Toshiba, *Cell Broadband Engine Architecture*, IBM, 2006.

[2] M.J. Castro, J.A. García-Rodríguez, J.M. González-Vida, and C. Parés, "Solving shallow-water systems in 2D domains using Finite Volume methods and multimedia SSE instructions," *Journal of Computational and Applied Mathematics*, vol. 221, pp. 16–32, 2008.

[3] J. Lobeiras, M. Arenaz, M. Amor, B. B. Fraguela, J. A. García, and M.J. Castro, "Optimización de simulación de aguas superficiales con contaminante en una GPU mediante Brook+," *XX Jornadas de Paralelismo*, 2009.

[4] IBM, *PowerPC Architecture, Book I*, IBM, Enero 2005.

[5] IBM, *PowerPC Architecture, Book II*, IBM, Enero 2005.

[6] IBM, *PowerPC Microprocessor Family: Vector/SIMD Multimedia Extension Technology Programming Environments Manual*, IBM, 2006.

[7] IBM, Sony, and Toshiba, *C/C++ Language Extensions for Cell Broadband Engine Architecture*, IBM, 2006.

[8] M.J. Castro, J.A. García-Rodríguez, J.M. González-Vida, and C. Parés, "A parallel 2D finite volume scheme for solving systems of balance laws with nonconservative products: Application to shallow flows," *Computer methods in applied mechanics and engineering*, pp. 2788–2815, Julio 2005.

[9] M.J. Castro, E.D. Fernández-Nieto, A.M. Ferreiro, J.A. García-Rodríguez, and C. Parés, "High order extensions of Roe schemes for two dimensional nonconservative hyperbolic systems," *Comput. Journal of Sci. Comput.*, vol. 39, pp. 67–114, 2009.

[10] M.J. Castro and C. Parés, "On the well balance property of Roe's method for nonconservative hyperbolic systems. application to shallow-water systems," *ESAIM:M2AN*, vol. 38, pp. 821–852, 2004.