Solución al problema de ordenación usando PipeLine y Binary Tree como composiciones paralelas de alto nivel

Mario Rossainz-López, Ivo H. Pineda-Torres, Patricia Domínguez Mirón

Benemérita Universidad Autónoma de Puebla,
México

{rossainz, ipineda}@cs.buap.mx, paty.dguez.m@gmail.com

Abstract. This paper proposes the model of the High Level Parallel Compositions or CPANs to communication patterns / interaction Pipeline and TreeDV for implementing a sorting algorithm by Structured Parallel Programming approach based on the concept of Parallel Objects. The CPANS TreeDV and PipeLine are displayed using the paradigm of object orientation and sorting problem is solved using two different algorithms; it is using a pipeline process to sort a dataset in disordered (CPAN Pipe) and one that by quick sort uses a binary tree for the ordering of the same dataset disordered by divide and conquer technique (CPAN TreeDVQS). Each proposal of CPAN contains a predefined set of restrictions of synchronization between processes (maximum parallelism, mutual exclusion and synchronization of producer-consumer type), and the use of synchronous, asynchronous and asynchronous future communication modes. Sorting algorithms, their design and implementation as CPANs and comparative performance metrics on a parallel machine 64 processors are shown.

Keywords: CPAN, programación paralela estructurada, objetos paralelos, programación orientada a objetos, pipeline, binary tree, divide y vencerás.

1. Introducción

Actualmente dentro del ámbito de la programación paralela uno de los problemas abiertos de mayor interés es el de la falta de aceptación de entornos de programación paralela estructurada para desarrollar aplicaciones paralelas por parte de los usuarios. Actualmente existen trabajos previos a esta investigación donde se hace patente la necesidad de contar con patrones paralelos de comunicación [1]. En la literatura existen varias propuestas, pero todas ellas coinciden en la importancia de determinar un conjunto completo de patrones y tratar de definir una semántica para ellos [6]. La tendencia para esto es en la actualidad el uso de enfoques de programación orientados a objetos, pues se ha visto que el definir objetos paralelos para el desarrollo de nuevas propuestas de metodologías, modelos y patrones de comunicación de programación paralela ha dado buenos resultados [4, 8]. Las Composiciones Paralelas de Alto Nivel o CPANS que aquí se proponen son patrones paralelos de comunicación bien

definidos y lógicamente estructurados que, una vez identificados en términos de sus componentes y de su esquema de comunicación, pueden llevarse a la práctica como constructos añadidos a un lenguaje de programación orientado a objetos y estar disponibles como abstracciones de alto nivel en las aplicaciones del usuario [11, 13]. Esto es posible gracias a la adopción del modelo de los Objetos Paralelos y con ellos construir el PipeLine y el BinaryTree como composición de éstos en alto nivel y general dichas estructuras de interconexión de procesos para resolver el problema de la ordenación.

2. Las composiciones paralelas de alto nivel o CPANs

Un CPAN es la composición de un conjunto de objetos paralelos de tres tipos: Un objeto manager que representa al CPAN en sí mismo y hace de él una abstracción encapsulada que oculta su estructura interna. El manager controla las referencias de un conjunto de objetos (un objeto denominado Collector y varios objetos denominados Stage), que representan los componentes del CPAN y cuya ejecución se lleva a cabo en paralelo y debe ser coordinada por el propio manager. Los objetos Stage son los encargados de encapsular una interfaz tipo cliente-servidor que se establece entre el manager y los objetos esclavos (objetos pasivos que contienen el algoritmo secuencial de la solución de un problema). Y un objeto Collector que es un objeto encargado de almacenar en paralelo los resultados que le lleguen de los objetos stage que tenga conectados, ver Fig. 1.

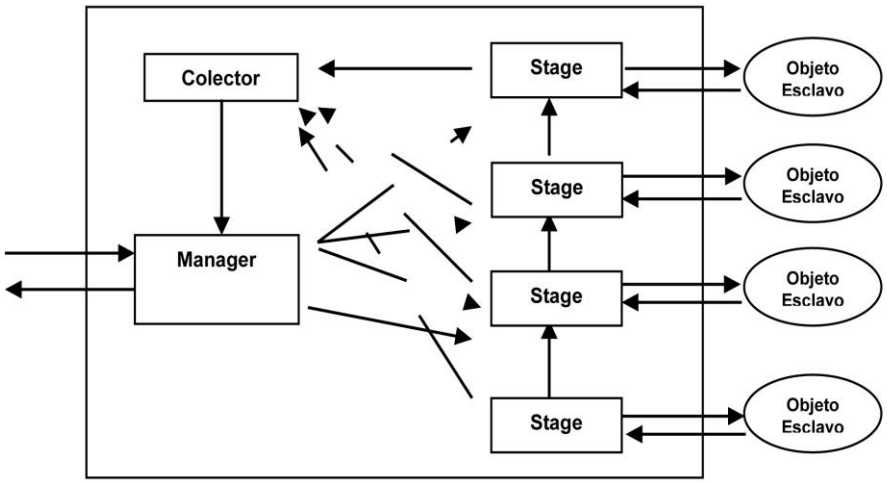

Fig. 1. Estructura Interna de un CPAN.

Los objetos manager, collector y stages son Objetos Paralelos (PO) [5, 6]. Los Objetos Paralelos tienen capacidad de ejecución en sí mismos [5]. Las aplicaciones dentro del modelo PO pueden explotar tanto el paralelismo entre objetos (inter-object) como el paralelismo interno de ellos (intra-object). Un PO tiene una estructura similar

a la de un objeto en Smalltalk, pero además incluye una política de planificación determinada a priori que especifica la forma de sincronizar una o más operaciones de la clase del objeto susceptibles de invocarse en paralelo [5, 7]. Los objetos paralelos soportan herencia múltiple, que permite derivar una nueva especificación de PO completa a partir de una que ya existe. Un CPAN cuenta con las siguientes propiedades que se pueden estudiar a detalle en [15]: Modo asíncrono y futuro asíncrono de comunicación entre los objetos paralelos del CPAN [6], objetos con paralelismo interno., disponibilidad de mecanismos de sincronización; Paralelismo Máximo (MaxPar), Exclusión Mutua (MuTex) y Sincronización (Sync) del tipo Productor-Consumidor, disponibilidad de control de tipos genéricos, transparencia en la distribución de aplicaciones paralela y rendimiento satisfactorio: Programabilidad-Portabilidad-Performance.

2.1 Clases base que conforman los Objetos Paralelos de un CPAN

En los PO las clases básicas necesarias para definir los objetos manager, colector, stages de un CPAN son: Una instancia de una clase concreta derivada de la clase ComponentManager (llamada manager) representa un CPAN dentro de una aplicación programada según el modelo de objeto paralelo. Las instancias (llamadas stages) de una clase concreta derivada de la clase ComponentStage se conectan entre sí para implementar una composición de stages. Cada stage ordena la ejecución en paralelo de un objeto llamado esclavo (slave), que es controlado por el propio stage. Por otro lado, es preciso señalar que la creación de los stages y de los colectores y sus interacciones posteriores son manejadas transparentemente al código de la aplicación por el manager. Desde el punto de vista de un usuario interesado en reutilizar el comportamiento paralelo ya definido en algunas clases CPAN, la clase de interés será la del manager. Cuando un usuario está interesado en usar un CPAN dentro de una aplicación, tiene que crear una instancia de una clase manager concreta, esto es, una que implemente el comportamiento paralelo requerido por la aplicación y que la inicialice con la referencia apropiada a los objetos esclavos que van a ser controlados por cada stage, así como indicar el nombre del método solicitado, para más detalles ver [12].

3. El CPAN pipeline

La técnica del procesamiento paralelo del pipeline se presenta como una Composición Paralela de Alto Nivel aplicable a la solución de un amplio rango de problemas que son parcialmente secuenciales en su naturaleza, de tal forma que el CPAN Pipe garantiza la paralelización de código del algoritmo secuencial.

3.1 La técnica del pipeline

La idea es dividir un problema a resolver en una serie de tareas que tienen que ser completadas una después de otra. En un pipeline cada tarea puede ser ejecutada por un proceso, un thread o un procesador independiente [9], ver Fig. 2.

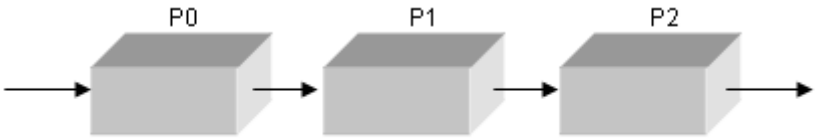

Fig. 2. Un Pipeline.

Algunas veces a los procesos del pipeline se les llama etapas (stages) del pipeline [10]. Cada etapa puede contribuir a la solución del problema total y puede pasar la información necesaria a la siguiente etapa del pipeline. Este tipo de paralelismo es visto muchas veces como una forma de descomposición funcional, ya que el problema es dividido en funciones separadas que pueden ser ejecutadas individual e independientemente; pero con esta técnica las funciones se ejecutan en sucesión. [9].

3.2 El modelo del pipeline como un CPAN

La Fig.3 representa el patrón paralelo de comunicación PipeLine como un CPAN.

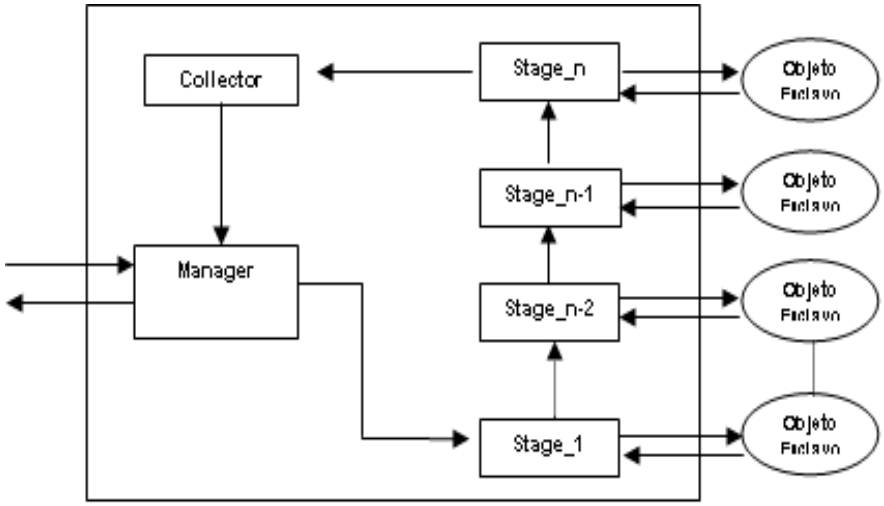

Fig. 3. El Pipeline como un CPAN.

Los objetos Manager y stage_i del modelo gráfico del CpanPipe son instancias de clases concretas, digamos PipeManager y PipeStage, que heredan las características de las clases base ComponentManager y ComponentStage, respectivamente, para poder así redefinir los métodos abstractos de las superclases. El objeto Collector es el único que será una instancia de la clase base ComponentCollector ya que ésta es una clase concreta. Una vez creados los objetos, y conectados de manera apropiada de acuerdo al patrón paralelo Pipeline, se tendrá entonces un CPAN para un tipo de patrón paralelo específico, después ya se puede resolver la asignación de los objetos esclavos asociados a los stages. Para ver más detalles consultar [12].

3.3 El algoritmo de ordenación usando un pipeline

En el funcionamiento de un algoritmo paralelo de ordenación basado en el pipeline se distinguen 3 fases según se menciona en [2, 3]:

- La carga inicial: Se pretende hacer llegar datos a todos los procesos asociados a las etapas del pipeline. En esta fase los procesos suelen ejecutar el mismo código que en la segunda fase, la diferencia consiste en que hay que inicializarlos de forma adecuada para que reciban el primer dato, que les va a llegar de la etapa anterior o de la carga inicial del programa.

- El procesamiento de la secuencia de datos con la máxima eficiencia: Los procesos se comportan de manera cíclica en su ejecución. Admiten datos de la etapa anterior, los procesan y envían el resultado a la etapa siguiente. Cada proceso tiene que estar sincronizado con el de la etapa anterior para que no le envíe nuevos datos cuando todavía no ha terminado de procesar los datos corrientes; pero también con el de la etapa siguiente, para no enviar el resultado hasta que el proceso de dicha etapa no esté preparado para recibirlo. El último proceso tiene un comportamiento especial con respecto a los procesos asociados a las etapas anteriores ya que tiene que ejecutar una rutina o código de salida y presentación de resultados. Su funcionamiento consiste en obtener los datos que le envía el proceso de la última etapa del pipeline y enviarlos a un dispositivo de salida o enviar una condición de terminación al programa principal. La serie de resultados que produce el último proceso habrá de coincidir con el resultado esperado del algoritmo que se ha paralelizado, si el pipeline ha sido paralelizado correctamente.

- La descarga: En esta última etapa los procesos envían el resultado del último dato procesado y ellos mismos detectan la situación de terminación, pues ya no van a recibir más datos de la secuencia de entrada y no se debe suponer ningún control global en el programa que les indique cuando tienen que terminar. Para que los procesos transmitan el dato almacenado en sus etapas antes de terminar, se suele introducir un valor especial al final de la secuencia de entrada que sirve para descargar el pipeline.

La implementación del algoritmo paralelo de ordenación consiste en un pipeline de procesos que es alimentado con una secuencia desordenada de números enteros por una rutina o código de entrada. El resultado obtenido como salida es la secuencia de enteros ordenada de menor a mayor. El número de valores de la secuencia de entrada no puede ser mayor que el número de etapas del pipeline. Cada uno de los procesos del pipeline tiene capacidad para almacenar un número entero, que será el mayor que haya recibido hasta ese momento de la etapa anterior. En cada iteración, un proceso recibe un entero, lo compara con el que tenía almacenado y envía el menor de ambos a la siguiente etapa del pipeline, mientras que el mayor es almacenado [2], [3]. Éste esquema representa el algoritmo paralelo de ordenación utilizando un Pipeline el cual ha sido implementado como CpanPipe

4. El CPAN treeDV

Se presenta la técnica de programación Divide y Vencerás como un CPAN haciendo uso de un patrón de comunicación en forma de árbol binario, aplicable a un amplio rango de problemas que pueden ser paralelizables bajo este esquema.

4.1 La técnica de divide y vencerás usando un árbol binario

Consideremos el algoritmo de divide y vencerás paralelo representado a través de un árbol binario (Fig.4.), donde cada nodo es un proceso.

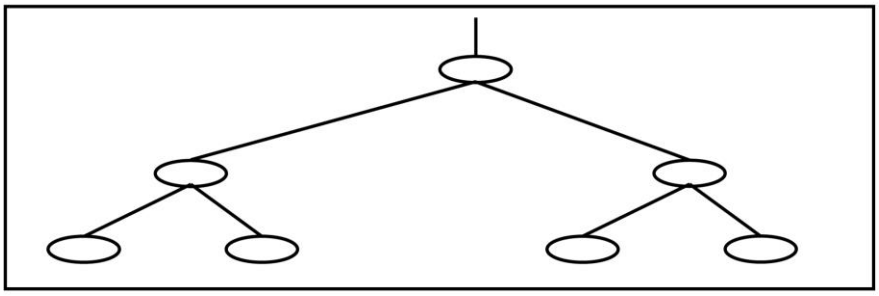

Fig. 4. Representación Gráfica de un árbol binario.

El nodo raíz del árbol tiene como entrada un problema completo, que se divide en dos partes y envía una parte al nodo hijo izquierdo y la otra parte al nodo hijo derecho. Este proceso de división es repetido en los niveles más altos del árbol. Eventualmente cualquier nodo hoja tiene como entrada un problema dado por su nodo padre, lo resuelve y la solución (que es la salida del nodo hoja) es enviada a su progenitor. Cualquier nodo padre en el árbol obtendrá dos soluciones parciales de sus hijos y las combinará para dar como salida una simple solución que será la salida del nodo padre. Finalmente el nodo raíz dará como salida la solución completa del problema.

4.2 La técnica de divide y vencerás como un CPAN

La representación del patrón treeDV que define la técnica de Divide y Vencerás como CPAN tiene su modelo representado en la fig. 5. A diferencia del modelo anterior, donde los objetos esclavos eran predeterminados, sólo un objeto esclavo es predefinido estáticamente y asociado al primer stage del árbol en este modelo, los siguientes objetos esclavos serán creados internamente por los propios stages de forma dinámica, pues los niveles del árbol dependen del problema a resolver y no se conoce a priori el número de nodos que pueda tener el árbol, ni tampoco su nivel de profundidad. No obstante, al igual que en el modelo anterior, los objetos Manager y stage_i de la fig. 5, son instancias de clases concretas, digamos TreeDVManager y TreeDVStage, que heredan de las clases base denominadas ComponenManager y ComponentStage, respectivamente. El objeto Collector es aquí también una instancia

de la clase base ComponentCollector. De acuerdo con la técnica de Divide y Vencerás se crea la clase TreeDVManager, que hereda de ComponentManager e implementa un patrón de comunicación de un árbol binario. Los nodos del árbol binario están representados por los stages que son objetos de la clase TreeDVStage que, a su vez, hereda de ComponentStage. Cualquier instancia de la clase TreeDVManager se encarga sólo del primer stage o nodo raíz del árbol en el momento de su inicialización. Durante la ejecución de una petición de servicio, la raíz del árbol binario, es decir, el primer stage de la estructura, es iniciada por el manager; además, todo nodo padre creado en los niveles más bajos del árbol ordenarán internamente la ejecución de sus respectivos nodos hijos, dentro del proceso de obtención de la solución del problema en esta técnica de diseño. Para ver más detalles consultar [12].

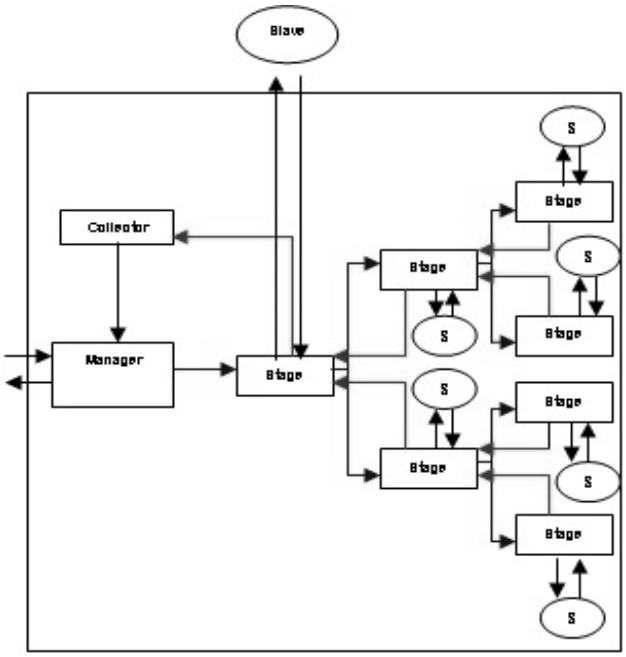

Fig. 5. El CPAN de un TreeDV.

4.3 El algoritmo de ordenación QuickSort usando el CPAN treeDV

El problema de ordenación rápida o Quicksort fue creado por Hoare y se basa en el paradigma de divide y vencerás. Como primer paso el algoritmo selecciona como pivote uno de los elementos del conjunto de datos que haya que ordenar. A continuación el conjunto se parte a ambos lados del pivote. Se desplazan los elementos de tal manera que los que sean mayores que el pivote queden a su derecha, mientras que los que sean menores queden a su izquierda. Posteriormente las partes del conjunto que quedan a ambos lados del pivote se ordenan independientemente de forma paralela y recursiva, en este caso a través de los objetos stage del modelo del CPAN TreeDV. El resultado final es un conjunto completamente ordenado.

5. Rendimiento

El análisis de rendimiento del CpanPipe y del CpanTreeDV para la implementación de los algoritmos paralelos de ordenamiento se llevó a cabo en una computadora paralela con 64 procesadores, 8 Gb de memoria principal, buses de alta velocidad y arquitectura de memoria compartida distribuida. Las medidas de rendimiento obtenidas en la ejecución de los CPANs que resuelven el problema de ordenación se llevó a cabo con las siguientes condiciones de ejecución

- Implementación paralela del algoritmo secuencial de ordenación basado en un pipeline para el caso del Cpan Pipe e implementación paralela del algoritmo secuencial de ordenación QuickSort basado en un árbol binario utilizando la técnica de Divide y Vencerás para el caso del CpanTreeDV

- En ambos casos, tanto para el CPan Pipe como para el Cpan TreeDV, se implementó el mismo algoritmo secuencial de comparación de valores en cada uno de los objetos esclavos asociados a las etapas o stage de los CPANs

- La ordenación de un conjunto de 50000 números enteros obtenidos de manera aleatoria en un rango de 0 a 50000, lo que ha permitido suponer una carga suficiente para los procesadores y con ello apreciar la mejora del rendimiento de los Cpans Pipe y TreeDV

- Ejecución del CpanPipe y del CpanTreeDV para 2, 4, 8, 16 y 32 procesadores con dedicación exclusiva para resolver el problema de ordenación en ambos casos.

Las tablas 1 y 2 y las Figuras 6 y 7 muestran las series de medidas obtenidas en los CPANs mencionados incluyendo las de sus correspondientes versiones secuenciales, tiempo de ejecución en segundos, ciclos ejecutados por instrucción, la magnitud del speedup encontrado y la cota superior de la magnitud del speedup utilizando para ello la ley de Amdahl.

Tabla 1. Rendimiento del Cpan Pipe Paralelo para la ordenación por de 50000 números enteros.

CPAN Pipe	Pipe Secuencial	cpuset2	cpuset4	cpuset8	cpuset16	cpuset32
Tiempo de ejecución en seg.	238.50	127.27	123.83	116.97	115.63	111.03
Consumo de tiempo en seg. de CPU	231.82	224.79	217.80	203.98	198.30	191.89
CPI	1.862	0.909	0.904	0.901	0.886	0.871
Speedup	1.00	1.87	1.93	2.04	2.06	2.15
Amdalh	1.00	1.89	3.39	5.63	8.42	11.19

Tabla 2. Rendimiento del TreeDV Paralelo para la ordenación por de 50000 números enteros.

CPAN TreeDVQS	TreeDVQS Secuencial	cpuset2	cpuset4	Cpuset8	cpuset16	cpuset32
Tiempo de ejecución en seg.	20.55	11.48	7.21	4.59	3.68	3.40
Consumo de tiempo en seg. de CPU	7.60	7.18	6.31	6.88	6.45	6.35
CPI	1.261	0.900	0.892	0.858	0.849	0.836
Sppedup	1.00	1.79	2.85	4.48	5.58	6.04
Amdalh	1.00	1.82	3.08	4.71	6.40	7.80

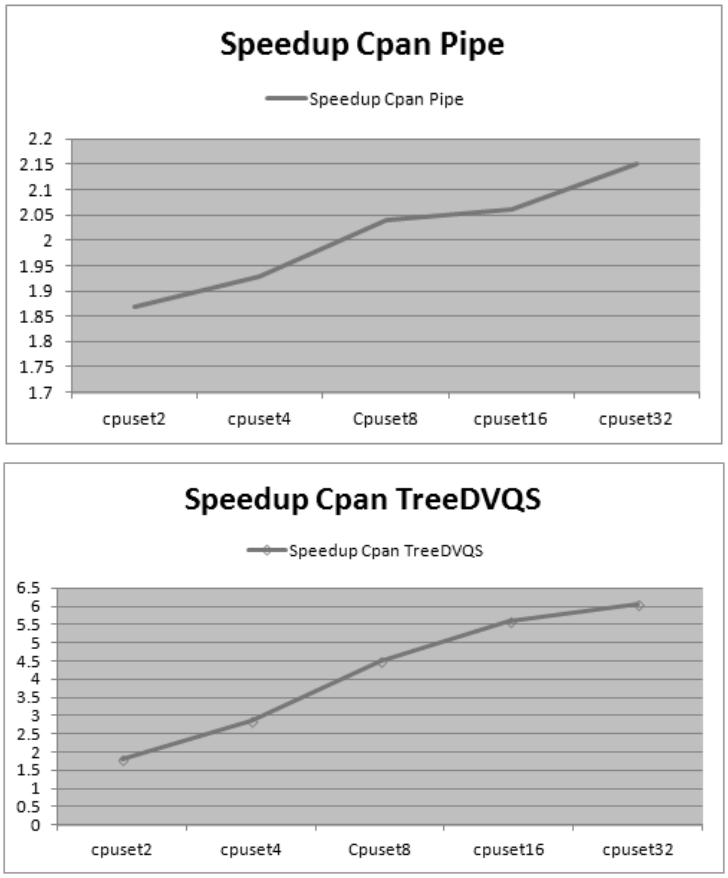

Fig. 6. Escalabilidad de la magnitud del Speedup encontrado para el Cpan Pipe y Cpan TreeDV con 2, 4, 8, 16 y 32 procesadores exclusivos.

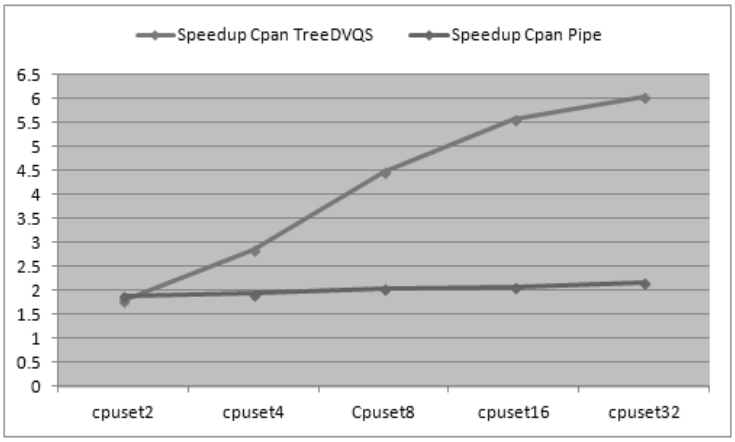

Fig. 7. Comparativo de la magnitud del Speedup de los Cpan Pipe y TreeDV en la solución del problema de ordenación con 2, 4, 8, 16 y 32 procesadores exclusivos.

6. Conclusiones

Se han implementado los CPAN PipeLine y TreeDV los cuales forman parte de una librería de clases de Objetos Paralelos que proporciona al usuario los patrón citados como Composiciones Paralelas de Alto Nivel más aparte otros patrones de comunicación como las granjas o farms y los árboles o Tree y cuyos detalles están publicados en [14]. Los CPANs Pipe y TreeDV pueden ser explotados gracias a la adopción del enfoque orientado a objetos para definir nuevos patrones en base a los ya construidos [14]. Se han transformado algoritmos secuenciales de ordenación en algoritmo paralelizables mediante el uso de un Pipeline y de un árbol binario utilizando el paradigma de Divide y Vencerás como CPANs. Se ha obtenido soluciones elegantes al problema de ordenación usando mediante dos tipos de algoritmos; aquél que utiliza descomposición funcional en forma de cauce y el de ordenación rápida o QUickSort Se han programado las restricciones de sincronización sugeridas por el modelo del CPAN: el paralelismo máximo (MaxPar), la exclusión mutua (Mutex) y la sincronización de comunicación de procesos productor/consumidor (Sync) en ambos CPANs.

Se ha probado el rendimiento de los CPANs Pipe y TreeDV mediante métricas de Speedup, Ley de Amdahl y eficiencia para demostrar que el comportamiento paralelo de los CPANs propuestos es mejor que su contraparte secuencial.

Referencias

1. Bacci, D., Pelagatti, V.: SklE: A Heterogeneous Environment for HPC Applications. Parallel Computing 25, pp. 1827–52 (1999)
2. Barry, W., Allen, M.: Parallel Programming. Techniques and Applications Using Networked Workstations and Parallel Computers. Prentice Hall (1999)

3. Blelloch, Guy E: Programming Parallel Algorithms. Comunications of the ACM, Vol. 39, No. 3 (1996)
4. Brinch, H.: Model Programs for Computational Science: A programming methodology for multicomputers. Concurrency: Practice and Experience, Vol. 5, No. 5, pp. 407–423 (1993)
5. Corradi, A., Leonardi, L.: PO Constraints as tools to synchronize active objects. Journal Object Oriented Programming 10, pp. 42–53 (1991)
6. Corradi, A.L., Zambonelli, F.: Experiences toward an Object-Oriented Approach to Structured Parallel Programming. DEIS technical report no. DEIS-LIA-95-007 (1995)
7. Danelutto, M.: Orlando, S.: Parallel Programming Models Based on Restricted Computation Structure Approach. Technical Report, Dpt. Informatica, Universitá de Pisa (1999)
8. Darlington: Parallel Programming Using Skeleton Functions. In: Proceedings PARLE'93, Munich (1993)
9. Robbins, K.A., Robbins, S.: UNIX Programación Práctica. Guía para la concurrencia, la comunicación y los multihilos. Prentice Hall (1999)
10. Roosta, Séller: Parallel Processing and Parallel Algorithms. Theory and Computation, Springer (1999)
11. Rossainz, M.: Una Metodología de Programación Basada en Composiciones Paralelas de Alto Nivel (CPANs). Universidad de Granada (2005)
12. Rossainz M., Capel M.: An Approach to Structured Parallel Programming Based on a Composition of Parallel Objects. In: Congreso Español de Informática CEDI-2005, XVI Jornadas de Paralelismo, Editorial Thomson, España (2005)
13. Rossainz, M., Capel, M.: A Parallel Programming Methodology using Communication Patterns named CPANS or Composition of Parallel Object. In: Proceedings of 20th European Modeling & Simulation Symposium, Campora S. Giovanni (ed.), Italy (2008)
14. Rossainz, M., Capel, M.: Compositions of Parallel Object to Implement Communication Patterns. In: Proceedings of XXIII Jornadas de Paralelismo, pp. 8–13, Elche, Spain (2012)
15. Rossainz, M., Pineda, I., Dominguez, P.: Análisis y Definición del Modelo de las Composiciones Paralelas de Alto Nivel llamadas CPANs. Modelos Matemáticos y TIC: Teoría y Aplicaciones. Dirección de Fomento Editorial, pp. 1–19, México (2014)