Diseño e Implementación de Composiciones Paralelas de Alto Nivel como un Modelo de Objetos Paralelos

Mario Rossainz-López[1], Ivo H. Pineda-Torres[1], Patricia Domínguez Mirón[1], Manuel Capel Tuñón[2]

[1]Benemérita Universidad Autónoma de Puebla
Avda. San Claudio y 14 Sur, San Manuel. 72000, Puebla, Pue. México
{rossainz,ipineda}@cs.buap.mx, paty.dguez.m@gmail.com
[2]Universidad de granada, DLSI-ETSII
C/Periodista Daniel Saucedo Aranda S/N, 18071, Granada, España
manuelcapel@ugr.es

Abstract. This paper presents the design of a model for programming parallel applications in a structured way. To achieve this, the model of High Level Parallel Compositions or CPANS under the paradigm of object orientation is used to solve problems which are parallelizable algorithms, using the concept of structured parallelism and possible integration within the paradigm of orientation objects to make way for the definition and model CPAN, using the Parallel objects (PO). The types of communication of PO, the templates that define the components of a CPAN as well as the syntactic and semantic definition of a CPAN are also described in this paper.

Keywords: CPAN, Programación Paralela Estructurada, Objetos Paralelos, Programación Orientada a Objetos, Paralelismo Estructurado.

1 Introducción

Un enfoque que intenta atacar el problema de la paralelización de algoritmos y programas consiste en tratar de hacer que el usuario desarrolle sus programas según un estilo de programación secuencial y ayudado de un sistema o entorno específico de programación, intentar producir su contraparte paralela automáticamente [1], [6]. Existen, sin embargo, dificultades de implementación intrínsecas a la definición de la semántica formal de los lenguajes de programación que impiden la paralelización automática sin contar con la participación del usuario, por lo que el problema de generar paralelismo de manera automática para una aplicación de propósito general sigue considerándose un problema abierto. Un enfoque alternativo es el paralelismo estructurado. Las aplicaciones paralelas generalmente siguen patrones predeterminados de ejecución, que son rara vez arbitrarios y no estructurados en su lógica [4]. Las Composiciones Paralelas de Alto Nivel o CPANS 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

©L. Berthet, J. C. Chimal, E. A. Santos, E. Castillo
Advances in Computer Science
Research in computing Science 81, 2014 pp. 113-123

disponibles como abstracciones de alto nivel en las aplicaciones del usuario. Esto es posible ya que las estructuras de interconexión de procesadores más comunes, como los pipelines, los farms y los trees, pueden ser abstraídas utilizando CPANS dentro de un modelo general de desarrollo de Objetos Paralelos [3].

2 El Paralelismo Estructurado

El enfoque estructurado para la programación paralela se basa en el uso de patrones de comunicación/interacción (pipelines, farms, trees, etc.) predefinidos entre los procesos de una aplicación de usuario. Éste permite diseñar aplicaciones en términos de CPANS capaces de implementar los patrones ya mencionados. La encapsulación de un CPAN debe seguir el principio de modularidad y debe proporcionar una base para obtener la reusabillidad del comportamiento paralelo de la entidad software que éste implementa. Se crea entonces un patrón paralelo genérico que proporciona una posible representación de la interacción entre los procesos independientes de la funcionalidad de estos. Así, en lugar de programar una aplicación paralela desde el principio y de programar con un nivel de detalle cercano a la capa protocolo de red, el usuario simplemente identifica los CPANS que implementan los patrones adecuados para las necesidades de comunicación de su aplicación y los utiliza junto con el código secuencial que implementa los cómputos que realizan sus procesos [4], [5].

2.1 El Paradigma de la Orientación a Objetos (OO)

El paradigma de la OO se utiliza en el presente trabajo para encapsular y abstraer patrones comunes de interacciones paralelas hacia un estilo de paralelismo estructurado. La idea es considerar a los CPANS como objetos encargados de controlar y coordinar la ejecución de sus componentes internos proporcionando características importantes de la OO tales como la uniformidad, la genericidad y la reusabilidad.

3 La definición del modelo CPAN

El objetivo es la de representar cualquier tipo de patrones paralelos de comunicación entre los procesos de un algoritmo paralelo y distribuido como clases, siguiendo el paradigma de la Orientación a Objetos [2]. Un CPAN es una composición de un conjunto de objetos de tres tipos. A partir de dichas clases, un objeto puede ser instanciado y la ejecución de un método del objeto se puede llevar a cabo a través de una petición de servicio (ver fig. 1). Los tipos de objetos que componen un CPAN son:
- **un objeto manager** que controla las referencias de un conjunto de objetos Collector y Stages, 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** que son objetos encargados de encapsular una interfaz tipo cliente-servidor que se establece entre el manager y los objetos esclavos (entidades externas que contienen el algoritmo secuencial que constituye la solución del problema), y de proporcionar la conexión necesaria entre ellos para implementar la semántica del patrón de comunicación que se pretende definir.

- **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.

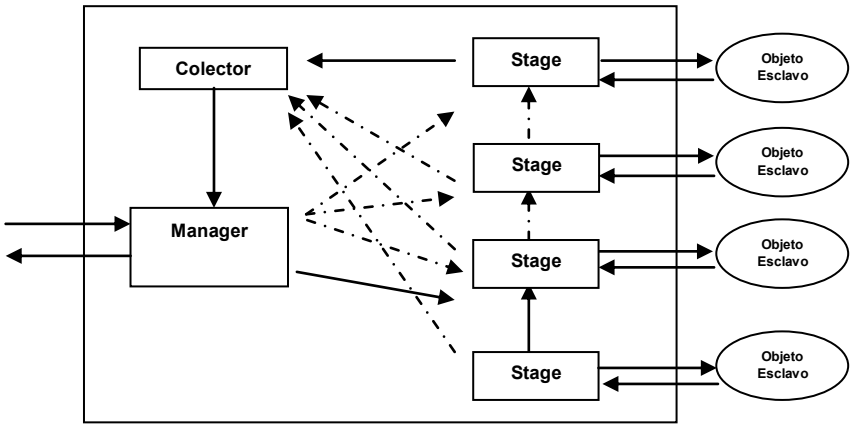

Fig. 1. Estructura Interna de un CPAN (Composición de sus Componentes)

3.1 El CPAN visto como una composición de objetos paralelos

Los objetos manager, collector y stages se engloban dentro de la definición de Objeto Paralelo (PO) [3], [4]. Los Objetos Paralelos son objetos activos, con capacidad de ejecución en sí mismos. Las aplicaciones dentro del modelo PO pueden explotar tanto el paralelismo entre objetos, como el paralelismo interno de ellos. Un objeto 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 (restricciones de sincronización, exclusión mutua y paralelismo máximo).

3.1.1 Definición Sintáctica

Desde el punto de vista sintáctico, la definición de la clase que define un Objeto Paralelo sigue el siguiente patrón o template:

```
CLASS: <nombre de la clase> INHERITS_FROM: <lista de los
nombres de clases padres directas>
    INSTANCE_STATE: <lista de los nombres y tipos de las
                     variables de instancia>
    INSTANCE_METHODS: <lista de métodos públicos>
    PRIVATE_METHODS: <lista de métodos privados>
    SCHEDULING_PART: <lista de restricciones de
                      sincronización>
END_CLASS <nombre de la clase>
```

3.1.2 Tipos de Comunicación en los Objetos Paralelos

1. El modo síncrono detiene la actividad del cliente hasta que el objeto activo servidor le cede la respuesta.
2. El modo asíncrono no fuerza la espera en la actividad del cliente. El cliente envía simplemente la petición al objeto servidor activo y continúa su ejecución. Su uso en la programación de aplicaciones también resulta fácil. Sólo hay que crear un hilo y lanzarlo para su ejecución.
3. El modo futuro asíncrono hace esperar la actividad del cliente sólo cuando, dentro de su código, se necesita el resultado del método para evaluar una expresión, [7], [4], [8]. Su uso también es sencillo, aunque su implementación requiere de un cuidado especial para conseguir un constructo con la semántica deseada.

4 Definición Sintáctica y Semántica de las clases base de un CPAN

En los PO las clases básicas necesarias para definir los objetos manager, colector, stages de un CPAN son: la clase abstracta ComponentManager, la clase abstracta ComponentStage y la clase concreta ComponentCollector. Las definiciones sintácticas siguientes han sido escritas utilizando una gramática libre de contexto publicada en [9].

4.1 La clase abstracta ComponentManager

Define la estructura genérica del componente manager de un CPAN, de donde se derivarán todos los objetos manager concretos, dependiendo del comportamiento paralelo que se contemple para la creación de cada CPAN considerado en la aplicación.

```
CLASS ABSTRACT ComponentManager
  {
    ComponentStage[] stages;

    PUBLIC VOID init (ASOCIACION[ ] list) {ABSTRACT;}

    PUBLIC ANYTYPE execution(ANYTYPE datain)
      VAR
         ComponentCollector res;
       {
        res = ComponentCollector CREATE();
        commandStages(datain,res);
        RETURN res.get();
       }

    PRIVATE VOID commandStages(ANYTYPE datain,
                               ComponentCollector res)
```

```
        {ABSTRACT;}

    MAXPAR (execution);
  };
```

4.2 La clase abstracta ComponentStage

Define la estructura genérica del componente stage de un CPAN, así como de sus interconexiones, de donde se derivarán todos los stages concretos dependiendo del comportamiento paralelo que se contemple en la creación del CPAN.

```
CLASS ABSTRACT  ComponentStage
  {
   ComponentStage[] otherstages;
   BOOL am_i_last;
   METHOD meth;
   OBJECT obj;

   PUBLIC VOID init (ASOCIACION[ ] list)
      VAR
       ASOCIACION item;
     {
      item = HEAD(list);
      obj = item.obj;
      meth= item.meth;
      if (TAIL(list) == NULL)
      am_i_last = true;
     }

   PUBLIC VOID request (ANYTYPE datain,
                                ComponentCollector res)
      VAR
       ANYTYPE dataout;
     {
      dataout = EVAL (obj,meth, datain);
      IF (am_i_last)
         TREAD res.put(dataout)
      ELSE commandOtherStages (dataout, res);
     }

   PRIVATE VOID commandOtherStages(ANYTYPE dataout,
                                ComponentCollector res)
      {ABSTRACT;}
   MAXPAR (request);
  };
```

4.3 La clase concreta ComponentCollector

Define la estructura concreta del componente collector de cualquier CPAN. Este componente implementa básicamente un buffer multi-item donde se irán almacenando los resultados de los stages que hagan referencia al collector. De esta forma se puede obtener el resultado del cálculo iniciado por el manager.

```
CLASS CONCRETE ComponentCollector
  {
   VAR
    ANYTYPE[] content;

   PUBLIC VOID put (ANYTYPE item)
     {  CONS(content, item); }

   PUBLIC ANYTYPE get()
      VAR
       ANYTYPE result;
     {
      result = HEAD(content[]);
      content = TAIL(content[]);
      RETURN result;
     }
   SYNC(put,get);
   MUTEX(put);
   MUTEX(get);
  };
```

4.4 Las restricciones de sincronización MaxPar, Mutex y Sync

Cuando se producen peticiones paralelas de servicio en un CPAN, resulta necesario disponer de mecanismos de sincronización para que los objetos que lo constituyen puedan gestionar varios flujos de ejecución concurrentemente y se garantice al mismo tiempo la consistencia de los datos que se están procesando. Para conseguirlo, dentro de cualquier CPAN, se pueden utilizar las restricciones MAXPAR, MUTEX y SYNC, para la correcta programación de sus métodos.

4.4.1 MaxPar

El paralelismo máximo, o MaxPar, es una variable de carácter permanente que indica el número máximo de procesos que pueden ejecutarse al mismo tiempo dentro de un componente en el modelo que estamos utilizando. Dicho de otra forma, el MAXPAR referido a una función del componente representa el número máximo de procesos que pueden ejecutar esa función concurrentemente. El algoritmo que ha servido para llevar a cabo correctamente la restricción de sincronización MAXPAR es el siguiente:

1. Incorporar una variable de clase en la clase de interés e inicializarla a cero.
2. Cada vez que se genere una instancia de esa clase, hay que actualizar su variable de clase sumándole uno al valor que tenía anteriormente, de forma que el número de instancias creadas corresponderá al número máximo de procesos que puedan ejecutar una función particular
3. En la función de interés implementar un semáforo de la siguiente forma:
   - 3.1. Si n es el número máximo de procesos que pueden ejecutar la función y si k es el número de procesos que actualmente se están ejecutando entonces:
   - 3.2. Si k≤n ejecutar en paralelo los k procesos, sino bloquear los k-n procesos y no despertarlos hasta que por lo menos uno de los n procesos que se están ejecutando haya terminado de forma que siempre se mantenga el límite de n procesos ejecutándose concurrentemente.

4.4.2 Mutex

La restricción de sincronización mutex lleva a cabo una exclusión mutua entre procesos que quieren acceder a un objeto compartido. Un mutex es una entidad de sincronización de Programación Concurrente que preserva secciones críticas de código para ser ejecutadas por un solo proceso cada vez, así como permitirle obtener acceso exclusivo a los recursos. El algoritmo que implementa la restricción de sincronización Mutex es el siguiente:

1. Utilizar una variable de condición que funcione como una bandera y un candado mutex.
2. Inicializar la bandera a falso
3. En la función particular de interés implementar un semáforo de la siguiente forma:
   - 3.1. Aquel proceso que adquiera el candado pondrá la bandera a verdadero y podrá ejecutar la función de interés
   - 3.2. Mientras la bandera sea cierta todos los demás procesos que intenten adquirir el candado para ejecutar la función se bloquearán y esperarán a que la bandera sea falsa para que alguno de ellos pueda tomar el candado
   - 3.3. El proceso que actualmente posee el candado podrá ejecutarse
   - 3.4. Al terminar su ejecución, pondrá la bandera a falso y liberará el candado

3.5.Aquel proceso que este al principio en la cola de espera, adquirirá el candado repitiéndose la secuencia desde el paso 3.1.

4.4.3 Sync

La restricción SYNC no es más que una sincronización del tipo productor/consumidor . El algoritmo de implementación es el siguiente:

1.Utilizar una variable de condición (digamos, n_items) que contabilice el número de elementos colocados en un contenedor.
2.Inicializar n_items a cero
3.Implementar un semáforo compartido por las funciones productor y consumidor de la siguiente forma:
  3.1. Un proceso consumidor permanecerá bloqueado mientras n_items sea igual a cero, pues con ello se indica que el proceso productor no ha colocado ningún dato en el contenedor que comparten.
  3.2. Un proceso productor colocará un dato en el contendor, adquirirá el candado y aumentará en uno el valor de n_items.
  3.3. Despertará al proceso consumidor y continuará ejecutándose en caso de procesar mas datos
  3.4. En el momento en que el n_items sea mayor que cero, el consumidor adquirirá el candado, disminuirá en uno el valor de n_items, indicando con ello que se ha procesado un dato del contenedor, y sacará el dato.
  3.5. Mientras el n_items sea mayor a cero el consumidor se estará ejecutando en paralelo con el productor.

5 Características importantes del modelo CPAN

- Capacidad de solventar los requerimientos de servicios de los objetos en los modos de comunicación síncrono, asíncrono y futuro asíncrono;
- Los objetos que se han definido poseen paralelismo interno, es decir, capacidad de gestionar varias peticiones de forma concurrente;
- Implementación de mecanismos de sincronización con semántica diferente para atender peticiones paralelas de servicio;
- Mecanismo flexible de control de tipos, es decir, es posible asociar tipos dinámicamente a los parámetros de los métodos de los objetos.

Con este modelo se han implementado los patrones de comunicación de procesos más comúnmente utilizados: cauces o pipelines, granjas o farms y árboles o trees como CPANs para resolver problemas de ordenación, búsqueda y optimización de datos

sobre un modelo de programación de memoria compartida [10]. Además, se han reutilizado algunos de estos CPANs para proponer soluciones paralelas bajo el esquema de éste modelo y resolver problemas NP-Completos como la propuesta de una solución al problema del Agente Viajero usando la técnica de Ramificación y Poda utilizando los CPANs [11]. Finalmente se han obtenido resultados de análisis de rendimiento y aceleración (speedup) de las implementaciones citadas (ver fig. 2 y fig. 3), en una máquina paralela con arquitectura de memoria compartida-distribuida para demostrar el buen rendimiento de dichas aplicaciones, ver detalles en [12], [13], [14].

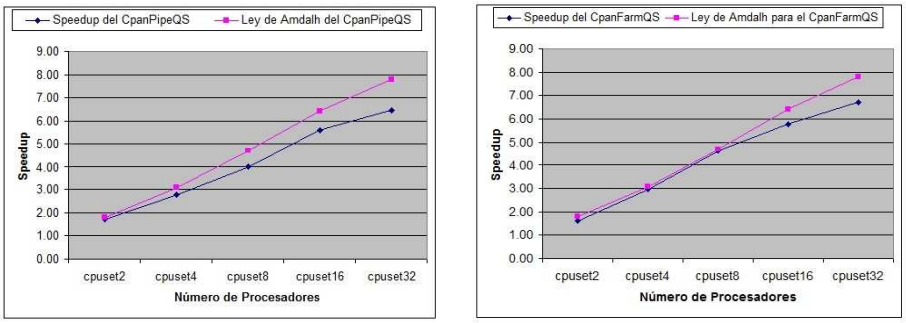

Fig. 2. Escalabilidad del Speedup para el Pipeline y Farm como CPANS con 2, 4, 8, 16 y 32 procesadores para el problema de ordenación de 50000 datos enteros

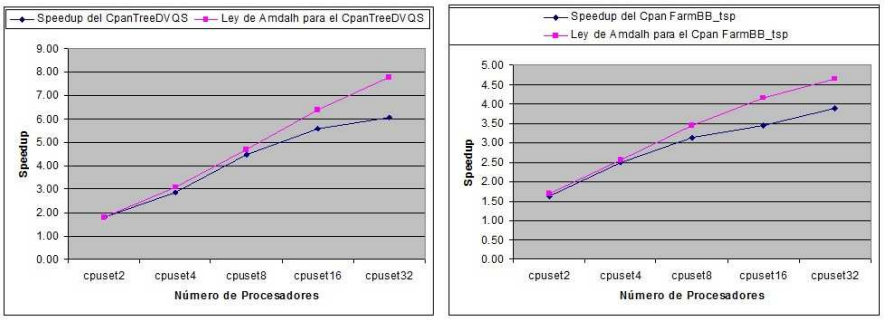

Fig. 3. Escalabilidad del Speedup para el Tree y la técnica de Branch & Bound como CPAN con 2, 4, 8, 16 y 32 procesadores para el problema de ordenación de 50000 datos enteros y del Agente Viajero para 100 ciudades respectivamente

6 Conclusiones

Se ha definido el modelo de las Composiciones Paralelas de Alto Nivel o CPANS. El modelo de los CPANS puede ser explotado para definir nuevos patrones en base a los ya construidos. Se han programado las restricciones de sincronización sugeridas por el modelo del CPAN para su funcionamiento paralelo y concurrente: MaxPar, Mutex y Sync. La programación del modo de comunicación futuro asíncrono para resultados

"futuros" dentro de los CPANS se ha llevado a cabo de manera original mediante objetos y clases. Se han programado los otros dos modos de comunicación clásicos entre procesos: el modo de comunicación síncrono y asíncrono. La programabilidad del modelo está garantizada al adoptar el enfoque basado en patrones como clases y objetos paralelos en la creación de los CPANS. La Portabilidad se da por la transparencia en la distribución de aplicaciones paralelas, lo que permite que los CPANS puedan ser portables desde algún sistema centralizado hacia un sistema distribuido sin necesidad de afectar el código fuente.

Referencias

1. Bacci, Danelutto, Pelagatti, Vaneschi: SklE: A Heterogeneous Environment for HPC Applications. Pp. 1827-52. Parallel Computing 25. (1999).
2. Brinch Hansen: Model Programs for Computational Science: A programming methodology for multicomputers. Concurrency: Practice and Experience, Vol. 5, No. 5, Pp. 407-423. (1993).
3. Corradi A., Leonardi L.: PO Constraints as tools to synchronize active objects. Pp: 42-53. Journal Object Oriented Programming 10. (1991).
4. Corradi A, Leonardo L, Zambonelli F.: Experiences toward an Object-Oriented Approach to Structured Parallel Programming. DEIS technical report no. DEIS-LIA-95-007. (1995).
5. Danelutto, M.; Orlando, S; et al.: Parallel Programming Models Based on Restricted Computation Structure Approach. Technical Report-Dpt. Informatica. Universitá de Pisa (1999).
6. Darlington et al.: Parallel Programming Using Skeleton Functions. Proceedings PARLE'93, Munich (1993).
7. Rabhi; Fethi.: A Parallel Programming Methodology based on Paradigms. In Transputer and Occam Development (18th WoTUG Technical Meeting). P. Nixon IOS Press, 239-249 (1995).
8. Roosta, Séller: Parallel Processing and Parallel Algorithms. Theory and Computation. Springer (1999).
9. Rossainz L.M.: Una Metodología de Programación Basada en Composiciones Paralelas de Alto Nivel (CPANs). Thesis (PhD). Universidad de Granada (2005).
10. Rossainz L. M, Capel T. M.: High Level Parallel Compositions (CPANs) for the Parallel Programming based on the use of Communication Patterns: 6th International Congress on ComputerScience - IPN: México (2006)
11. Rossainz L. M., Capel T. M.: Using the CPAN Branch & Bound for the Solution of Travelling Salesman Problem. Advances in Computer Science and Engineering. Volume 45. ISSN: 1870-4069. México (2010).
12. Rossainz L. M., Capel T. M.: Design and Implementation of the Branch & Bound Algorithmic Design Technique as an High Level Parallel Composition. Proceedings of International Mediterranean Modelling Multiconference. Barcelona, Spain (2006).
13. Rossainz L. M., Capel T. M.: A Parallel Programming Methodology using Communication Patterns named CPANS or Composition of Parallel Object. Proceedings of 20TH European Modeling & Simulation Symposium. Campora S. Giovanni. Italy (2008).
14. Rossainz L. M., Capel T. M.: Compositions of Parallel Object to Implement Communication Patterns. Proceedings of XXIII Jornadas de Paralelismo, pp.8-13. September 19-21. Elche, Spain (2012).