Implementación de Patrones de Diseño Paralelos en JAVA como una biblioteca de clases de Objetos Paralelos

M. Rossainz López[1], J.J. Varela Toledo[1], I. Pineda Torres[1] , M. Capel Tuñón[2]
[1]Benemérita Universidad Autónoma de Puebla, Facultad de Ciencias de la Computación
Avda. San Claudio y 14 Sur, San Manuel, 72570, Puebla, Pue., México
[2]Universidad de granada, DLSI-ETSII. C/Periodista Daniel Saucedo Aranda S/N
18071, Granada, España
rossainz@cs.buap.mx[1], jordanv53@hotmail.com[1], ipineda@cs.buap.mx[1], manuelcapel@ugr.es[2]

Resumen—**El trabajo propone el diseño e implementación de un conjunto de algoritmos paralelos programados en JAVA que proporcionan al programador una forma simple de solucionar problemas diversos que pueden ir desde una solución de ordenación hasta la de un problema de simulación mediante el uso de una biblioteca de clases de Objetos Paralelos. Los algoritmos paralelos implementados se construirán como Patrones de Diseño Paralelos a través de una metodología donde cada diseño algorítmico representa la estructura paralela de control común a una determinada técnica algorítmica generando un programa paralelo general del cual a su vez se deriven dos o más programas modelo que resuelvan problemas particulares. Se ha diseñado la técnica algorítmica de divide y vencerás como un Patrón de Diseño Paralelo y se han implementado algunas aplicaciones de éste (Quicksort, y Búsqueda Binaria). La implementación se hace de forma concurrente mediante la creación de hilos, explicando los pasos a realizar en el desarrollo de éste tipo de Patrones de Diseño Paralelos.**

Palabras Clave—**Algoritmos Paralelos, Concurrencia, Objetos Paralelos, Patrones de Diseño Paralelos, Programación Paralela**

I. Introducción

Una de las más importantes utilidades de los multi-computadores es su uso en la computación algorítmica para obtener velocidad y eficacia. Se propone el diseño e implementación de un conjunto de algoritmos paralelos como patrones de diseño paralelos que deriven en dos o más programas específicos, llamados programas modelo que resuelvan problemas particulares. Andrews [2] clarifica el estudio de algunas técnicas de programación paralela aplicables a numerosos problemas prácticos; sin embargo estas técnicas asumen estructuras de control distintas en problemas similares, lo cual genera dificultades prácticas al utilizarlas como pueden ser las siguientes:

- Es difícil suponer cual es la estructura algorítmica común a todos los algoritmos del mismo patrón de diseño generado.
- Los patrones de diseño propuestos son demasiado dependientes del problema a resolver y no proporcionan intuición práctica para derivar nuevos algoritmos utilizando la orientación a objetos.
- Estos patrones de diseño se entienden mejor como programas particulares más que como auténticos patrones.

Por ello se hace necesario definir un diseño de nivel intermedio entre la descripción vaga de una Programación de Patrones de Diseño Paralelos y un conjunto de Programas Particulares que resuelven un problema concreto. Este nivel intermedio es un Patrón de Diseño original que describe la estructura de control común compartida por todos los algoritmos que corresponden al mismo patrón. Ni los tipos de datos, ni el código de procedimientos específicos dependientes de los datos necesitan ser detallados en este nivel, ya que éstos dependen de problemas concretos.

El objetivo general de la investigación es el de obtener eficiencia en aplicaciones paralelas derivando estructuras de control comunes a problemas distintos que lleven dentro de sí el mismo patrón de programación y tratar con ello de minimizar las dependencias de los datos globales entre procesos paralelos. Para hacer esto se requiere que el componente secuencial de los algoritmos paralelos tenga un mínimo de dependencias con el código que implementa el patrón de programación; por tanto la creación de una estructura algorítmica general debe ser adaptada a las diferentes aplicaciones concretas cuyas formas de solución pertenezcan al mismo patrón. Un objetivo particular es el de implementar el patrón divide y vencerás en el lenguaje JAVA como una estructura general o plantilla, donde sus procesos paralelos deben ser especificados mediante la creación de hilos de forma concurrente. Cada hilo representa por tanto un proceso genérico a ejecutarse en el programa e incluye como parámetros formales el problema secuencial a resolver o subproblemas. Cabe mencionar que la recursividad juega un papel muy importante en ciertos patrones como lo es el mencionado, el cual es definido como un algoritmo de partición recursiva, es decir; divide y vencerás resuelve un problema dividiéndolo en problemas subordinados los cuales a su vez son resueltos recursivamente mediante la división de

ellos en el futuro. El resultado final se obtiene igualmente de forma recursiva combinando las soluciones de los subproblemas. El paralelismo nace casi de forma natural en este tipo de diseños algorítmicos donde cada problema puede ser resuelto independientemente de los otros. Para especificar un algoritmo de partición recursiva como lo es divide y vencerás hay que tener en cuenta componentes importantes tales como:

- Estructuras de datos para representar el problema y la solución.
- Un predicado que indique cuando un problema es indivisible o si éste puede ser dividido en el futuro.
- Una función que regrese problemas indivisibles en soluciones.
- Una función que divida problemas en subproblemas.
- Una función que combine soluciones en una sola solución.

El uso del lenguaje de programación JAVA para este tipo de patrones de diseño provee de un elegante camino para expresar estos parámetros y conservar el paralelismo de forma dinámica al generar las tareas en tiempo de ejecución y tener una estructura en forma de árbol.

II. Paradigmas para la programación paralela

Los paradigmas para la programación paralela, en particular para la teoría de algoritmos paralelos, definen una metodología de programación dentro de la ciencia computacional. Esta metodología se basa en la aplicación de un proceso de derivación informal para la programación de dichos paradigmas. Algunas clasificaciones que se tienen sobre algoritmos paralelos o paradigmas han sido propuestas por G.A. Andrews [2], [3], Brinch-Hansen [7], [8] y F. Rabhí [16]. En este trabajo se retoman las ideas y los estudios de Brinch-Hansen, junto con otros trabajos previos [4] por ser la propuesta más acorde con el concepto de Programación de Paradigmas en el sentido siguiente: Un paradigma es una clase de algoritmos que resuelve diferentes problemas que tienen la misma estructura de control. Algunos de estos paradigmas son:

- Multiplicación de tuplas o tuple-Multiplication
- Divide y Vencerás o divid-and-conquer

Para cada paradigma se crea un Programa General que define la estructura de control común a aquellos problemas que puedan ser resueltos con una misma técnica de diseño. Al programa general se le da el nombre de Esqueleto Algorítmico o template. Posteriormente desde un programa paralelo general, se derivan dos o más Programas Modelo que ilustren el uso del paradigma para resolver problemas específicos.

Un programa general incluye algunos tipos de datos que no se especifican y procedimientos que varían de una aplicación a otra. Un programa modelo se obtiene al reemplazar estos tipos de datos y procedimientos por los correspondientes tipos de datos y procedimientos de un programa secuencial que resuelve un problema específico. En otras palabras, la esencia de esta metodología de programación es que un programa modelo tiene un componente paralelo que implementa un paradigma y un componente secuencial para una aplicación específica (Fig.1).

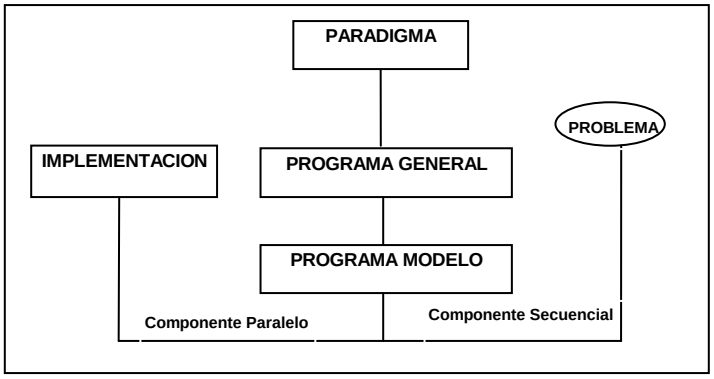

Fig. 1. El Modelo de Paradigma

Los programas modelos correspondientes a paradigmas como los mencionados anteriormente, pueden resolver problemas típicos en la ciencia e ingeniería tales como:

- Multiplicación de Matrices.
- Rutas de caminos más cortos en grafos.
- Ordenamiento Rápido.
- Búsquedas de elementos.

Estos programas paralelos pueden ejecutarse en arquitecturas tan variadas como pipelines, árboles, cubos o matrices de procesos, según el problema a resolver.

A. *El Paradigma Multiplicación de Tuplas*

Para seguir ilustrando los beneficios del desarrollo de algoritmos genéricos para la programación en paralelo de paradigmas, consideremos la multiplicación de matrices que bien puede ser implementada como un pipeline hacia delante como lo muestra la Fig.2.

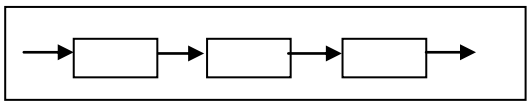

Fig. 2. El modelo del Pipeline

Las líneas de una matriz A son distribuidas entre los nodos del pipeline. Entonces las columnas de una matriz B pasan sobre el pipeline mientras cada nodo calcula una porción de la matriz producto AXB. Finalmente el pipeline da como salida la matriz producto. Para los algoritmos secuenciales es conocido que la multiplicación de matrices es similar al problema de encontrar la ruta más corta entre cualquier par de nodos de un grafo dirigido con n-nodos. Después de estudiar ambos algoritmos el concepto de unificación encontrado es un paradigma llamado multiplicación de tuplas: el producto de 2 n-tuplas A y B es una matriz C de tamaño nxn; Cij= F(Ai,Bj). En la multiplicación de matrices, los elementos de la tupla son líneas y columnas respectivamente y cualquier valor de la función F es el producto punto de una línea por una columna.

En el caso del problema de los caminos más cortos la entrada es la matriz de adyacencia de un grafo. La salida es una matriz de distancia calculada por una secuencia de multiplicación de tuplas. El valor de la función F(Ai,Bj) define la longitud de la ruta más corta encontrada entre dos nodos i e j del grafo donde la tupla A se forma de las n-líneas de la matriz de adyacencia y la tupla B de las n-columnas de la matriz de distancia. La tarea siguiente es obvia, señalar el paradigma en cuestión, el programa general y los programas modelo.

B. El Paradigma Divide y Vencerás

Ahora consideremos el algoritmo de divide y vencerás paralelo representado a través de un árbol binario (Fig.3.), donde cada nodo es un proceso.

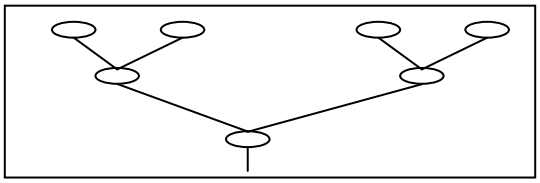

Fig. 3. Representación gráfica de un árbol binario

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. En consecuencia la transformación del algoritmo general en versiones paralelas de quicksort, búsqueda binaria, o encontrar el máximo y el mínimo de un vector de elementos es relativamente fácil.

C. Características Generales

La tabla1 muestra las características generales de los paradigmas vistos:

PARADIGMA	PROGRAMA	ARQUITECTURA
Multiplicación	Matrices	PipeLine
	Rutas Grafos	
Divide y Vencerás	Ordenación	Árbol
	Búsqueda	

Tabla 1. Características Generales de los paradigmas: multiplicación de tuplas y Divide y Vencerás

D. Derivación Informal

Para cualquier paradigma se define el siguiente método de búsqueda:

1. Identificar dos problemas computacionales con la misma estructura de control.
2. Para cada problema escribir un tutorial que explique su teoría computacional e incluya un programa completo.
3. Escribir un programa paralelo para la programación del paradigma.
4. Probar el programa paralelo en una computadora secuencial.
5. Derivar un programa paralelo para cada problema mediante sustituciones triviales de algunos tipos de datos, variables, procedimientos, etc., y analizar la complejidad de esos programas.
6. Rescribir los programas paralelos en un lenguaje de implementación y medir su rendimiento en una multicomputadora.
7. Escribir descripciones claras de los programas paralelos.
8. Publicar los programas y sus descripciones en su totalidad.

En el capítulo siguiente se presenta el aprovechamiento de esta metodología de programación paralela mediante la implementación del paradigma divide y vencerás como una plantilla, en el lenguaje de programación JAVA. Este paradigma es, el que a mi juicio, representa mayor facilidad y entendimiento para programarlo bajo esta base metodológica.

III. PARADIGMA DIVIDE Y VENCERÁS PARALELO EN JAVA

El uso de multithreading (multihebras) es considerado como una parte importante en el lenguaje JAVA, pues proporciona herramientas elementales utilizadas en la programación concurrente, las cuales se hacen más potentes al utilizar los conceptos propios de la orientación a objetos tales como el concepto de encapsulación, herencia y/o polimorfismo. El aprovechamiento de éstas, se lleva a cabo en este sentido a través de la implementación de las técnicas de diseño de algoritmos o paradigmas descritos para resolver diferentes problemas en la ciencia y/o la ingeniería. Para cada paradigma el objetivo es crear un programa general, esqueleto algorítmico, template o plantilla en JAVA donde se defina la estructura de control común a aquellos problemas que puedan ser resueltos con una misma técnica de diseño. Una vez creada la plantilla, la idea es crear a través de ella los programas modelos de los problemas que se resuelven con el mismo paradigma.

A. El algoritmo Divide y Vencerás

Divide y vencerás es una técnica de diseño de algoritmos que consiste en descomponer el caso que haya que resolver en un cierto número de subcasos más pequeños del mismo problema, resolver sucesiva e independientemente todos estos subcasos y combinar después las soluciones obtenidas de esta manera para obtener la solución del caso original. El caso general de los algoritmos de divide y vencerás es:

```
funcion DyV(x)
   descomponer x en casos más pequeños x1,...,xn
   para i que va de 1 to l hacer yi= DyV(xi)
   recombinar los yi para obtener una solución y
   de x
devolver y
```

Como se puede observar el aprovechamiento o ventaja de la técnica radica en el uso de la recursividad.

B. Programa General o Plantilla Divide y Vencerás

La plantilla del paradigma divide y vencerás se programó en JAVA utilizando Threads en la definición del componente paralelo, pensando en un árbol binario cuyo nodo raíz del árbol tiene como entrada un problema completo (el componente secuencial), que se divide en dos partes y envía una parte al nodo hijo izquierdo del árbol y la otra parte al nodo hijo derecho. Este proceso de división es repetido en los niveles más altos del árbol mediante la recursividad. Cada nodo del árbol será un Thread en la programación JAVA los cuales se irán creando conforme se vayan necesitando hasta cumplirse una condición de paro para que en el backtraking del proceso recursivo se vayan obteniendo las soluciones parciales de los subproblemas y se combinen de forma que el nodo raíz, al final, obtenga la solución completa del problema en cuestión. La plantilla divide y vencerás de nombre DyV es la siguiente:

```
class DyV implements Runnable
  {
   /*Declaracion de Variables de Instancia*/

   private DyV(/*parametros*/)
     {
      /*Cuerpo del Constructor de la clase DyV*/
     }

   private static Thread DVThread(/*parametros*/)
     {
       DyV dv= new DyV(/*argumentos*/);
       Thread dvt= new Thread(dv);
       dvt.start();
       return dvt;
      }

    public void run()
      {
       aplicacion_secuencial(/*datos*/);
      }

    public static void divide_y_vence(/*parámetros
                                                  */)
      {
       Thread qst= DVThread(/*valores iniciales
                              del problema*/);
       try {
            qst.join();
           } catch (InterruptedException e) {;}
       }

     public static void aplicacion_secuencial
                                 ( /*parametros*/)
       {
         /*Cuerpo del problema secuencial*/
         divide_y_vence(/*parte izq. del
                            prob. Seq.*/);
         divide_y_vence(/*parte der. del
                            prob, Seq. */ );
        }

     /* demas metodos del usuario */
    }
```

C. Un Programa Modelo Divide y Vencerás

Tanto el problema como su solución se definen ambos por un arreglo de n-elementos del mismo tipo. El tipo de los elementos y procedimientos para la división del problema y combinación de soluciones, forma parte del algoritmo paralelo que depende de la naturaleza del programa específico o programa modelo. Esta es la característica principal que hace que la transformación de la plantilla en versiones paralelas de problemas específicos sea más fácil; pues sólo hay que añadir a la plantilla el problema secuencial que se pretenda resolver bajo esta técnica, además de los tipos de datos y procedimientos referentes a dicho problema secuencial. Para demostrar la eficiencia de la plantilla en su versión paralela, se programaron 3 tipos de problemas secuenciales basados en el paradigma de divide y vencerás:

- Quicksort u ordenación rápida de un conjunto de datos.
- Maxmin o problema de encontrar el elemento mayor y el elemento menor de un conjunto de datos.
- Busbin o búsqueda binaria de un elemento dentro de un conjunto de datos.

En todos estos problemas secuenciales el componente paralelo es el mismo puesto que tal componente es ni más ni menos que la técnica de divide y vencerás bajo la cual éstos problemas son resueltos. Tomemos por ejemplo el problema de ordenación rápida o Quicksort para implementarlo en la plantilla DyV. Este algoritmo de ordenación creado por Hoare 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. El resultado final es un conjunto completamente ordenado. El algoritmo secuencial de Quicksort es el siguiente:

```
Procedimiento QuickSort (conjunto, inicio,fin)
   Si (inicio < fin)
   Entonces
      pivote=conjunto[inicio]
       nuevo_inicio= inicio
       Para i = (inicio+1) a fin hacer
         Si (conjunto[i] <= pivote)
          Entonces
            nuevo_inicio = nuevo_inicio+1
           intercambia(conjunto[nuevo_inicio],
                       conjunto[i])
       intercambia(conjunto[inicio],
                   conjunto[nuevo_inicio] )
        QuickSort(conjunto,inicio,nuevo_inicio)
        QuickSort(conjunto,nuevo_inicio+1,fin)
    fin de QuickSort.
```

Para implementarlo en la plantilla DyV sólo tendrá que incluirse en ella el procedimiento QuickSort tal cual junto con sus tipos de datos correspondientes y procedimientos propios tales como el procedimiento intercambia y el programa principal creando así el programa modelo formado por el componente paralelo o paradigma Divide y Vencerás y el problema secuencial u ordenación rápida Quicksort (Fig.1),

como se ilustra a continuación:

```
// QuickSort Paralelo recursivo usando Threads
class DyV implements Runnable
   {
    private static int[] nums;
    private int inicio, fin,n;

    private DyV(int[] ar,int inicio, int fin)
        {
         this.nums=ar;
         this.inicio=inicio;
         this.fin=fin;
        }

    private static Thread DVThread(int[] ar,
                        int inicio, int fin)
      {
       DyV dv= new DyV(ar,inicio,fin);
       Thread dvt= new Thread(dv);
       dvt.start();
       return dvt;
      }

    public void run()
      {
       quicksort(nums,inicio,fin);
      }

    public static void divide_y_vence(int[] ar,
                          int inicio, int fin)
      {
        Thread qst= DVThread(ar,inicio,fin);
        try {
             qst.join();
            } catch (InterruptedException e) {;}
      }

    private void quicksort(int[] ar,int inicio,
                                      int fin)
      {
        if (inicio<fin)
          {
           int newinicio, piv;
           piv=ar[inicio];
           newinicio=inicio;

           for (int i=(inicio+1);i<fin;i++)
               {
                if (ar[i]<= piv)
                  {
                   newinicio=newinicio+1;
                   Herramientas.intercambia(ar,
                                  newinicio,i);
                  }
                }
           Herramientas.intercambia(ar,inicio,
                                    newinicio);
           divide_y_vence(ar,newinicio+1,fin);
           divide_y_vence(ar,inicio,newinicio);
          }
      }

     public static void main(String[] args)
        {
         int[] ar={3,2,1,5,8,4,3,7,0,9,6,10,5,0,0};

         System.out.println("\n QUICKSORT PARALELO
                              RECURSIVO");
         System.out.println("USANDO EL PARADIGMA DE
                            DIVIDE Y VENCERAS \n");
         Herramientas.imprime(ar);
         divide_y_vence(ar,0,ar.length);
         Herramientas.imprime(nums);
        }
     }
```

De manera similar se programan los problemas de búsqueda binaria y el de encontrar el máximo valor y el mínimo valor de un conjunto de datos.

D. Análisis de la Plantilla DyV

El problema secuencial es un parámetro de entrada del método divide_y_vence( ) el cual implementa la técnica de Divide y Vencerás en su versión paralela lanzando una hebra para empezar a procesarlo. Se genera entonces un objeto de tipo Thread que lleva en primera instancia el problema completo que tratará de resolver al ejecutarse el correspondiente método run de la hebra. En éste método se hace el llamado a la función que implementa el problema secuencial, en este caso quicksort( ) la cual obtiene el pivote a partir del que se hará la división del conjunto de datos con el que se esta trabajando, de manera que los elementos menores al pivote se procesaran en paralelo con los elementos mayores al pivote, al lanzarse las hebras correspondientes a estos dos subproblemas; los cuales se resolverán en forma recursiva generando hebras hasta cumplirse la condición de paro del programa secuencial. Al final el resultado de la primera hebra lanzada será el resultado del problema general, es decir, el conjunto de datos ya ordenado. Los datos que inicialmente esta desordenados son pasados por referencia a todos los threads generados, los cuales varían durante el cálculo.

IV. CONCLUSIONES

Una de las metas de investigación más importantes en el desarrollo del software para multicomputadores es la construcción de metodologías y herramientas para generar algoritmos y programas paralelos de forma sistemática y segura. En este sentido las conclusiones a las que se han llegado con el presente trabajo son las siguientes:

1. Se ha desarrollado una metodología para la programación de algoritmos paralelos en base a un conjunto de paradigmas.
2. Los paradigmas son programas que dan soluciones elegantes a problemas no triviales y cambian nuestra habilidad intelectual y ejercitan nuestra programación.
3. Un paradigma es un concepto unificado que revela similaridades inesperadas entre algoritmos e incorpora nuevas cuestiones a cerca de algoritmos familiares.
4. Los paradigmas paralelos capturan la esencia de las arquitecturas paralelas tales como los pipelines, árboles, hipercubos y matrices.
5. El lenguaje JAVA es un lenguaje de programación adecuado para la implementación de los paradigmas.
6. El estudio de programación de paradigmas provee una visión de arquitecturas científicas de computación paralela.

V. TRABAJOS FUTUROS

Se pretende continuar refinando la programación del paradigma divide y vencerás en JAVA. La idea es producir objetos activos que implementen la técnica divide y vencerás en paralelo mediante un árbol de hebras con el problema original como raíz y los subproblemas como hojas de dicho árbol. La implementación debe ser genérica, es decir; independiente del problema concreto que se pretende resolver. El esquema que

se propone es implementar problemas tales como QuickSort, Búsqueda Binaria o Máximo-Mínimo como instancias del paradigma Divide y Vencerás. En cada uno de estos problemas secuenciales, el componente paralelo será el paradigma. Por ejemplo, la implementación del paradigma Divide y Vencerás en JAVA consistiría de los siguientes componentes:

La Interfaz: Todos los problemas que se resuelvan mediante éste paradigma deberán de implementar esta interfaz. Con esto garantizamos que dichos problemas sean compatibles con ella y por tanto puedan implementar los métodos abstractos de la misma. De esta manera la clase que implementa el paradigma a la que le llamaremos DyVParadigma podrá resolver los problemas de forma genérica. La clase DyVParadigma recibe entonces referencias a objetos compatibles con la interfaz y por lo tanto podrá invocar los métodos abstractos de esta, los cuales son:

`public boolean base()`: Que nos devuelve un valor TRUE si los datos del objeto representan un problema base o indivisible.
`public Object resolver()`: Nos devuelve una solución a un subproblema base cuando se invoca este método.
`public Interfaz[] dividir()`: Divide un problema no-base en un vector de sub-problemas.
`public Object combinar(Object[])`: Recibe un vector de objetos solución a subproblemas, los combina y devuelve una solución al problema combinado.

La clase DyVParadigma: Ésta producirá objetos activos que implementan la técnica divide y vencerás en paralelo, mediante un árbol de hebras con el problema original como raíz y los subproblemas como hojas de dicho árbol. La implementación deberá ser independiente del problema concreto que se pretenda resolver. El constructor de esta clase iniciará entonces una hebra. Recibe un problema a resolver y devolverá su solución. Si el problema que recibe es base entonces se resuelve, y si no, se invoca al método dividir(). Por cada subproblema se creará una instancia de la clase DyVParadigma. Se enviarán los subproblemas a las hebras hijas y se obtendrán las soluciones de los subproblemas, para después combinar dichas soluciones y devolver la solución del problema que se recibió al principio.

La clase Problema: Esta clase será utilizada para crear sólo instancias del problema a resolver el cual deberá ser resoluble con un algoritmo secuencial concreto (quicksort, busbin, etc.). Esta clase deberá implementar la interfaz.

La clase Solución: Una instancia de esta clase contendrá una solución al problema que se pretende resolver, por ejemplo, en el caso de QuickSort, la clase solución deberá contener instancias de ella misma conteniendo un vector ordenado.

La clase Montaje: Esta es la clase que contendrá el programa principal o método main( ). En esta clase un vector de elementos es creado para después obtener una instancia de la clase problema que contendrá al problema original (valga la redundancia). Entonces una hebra es lanzada con el problema inicial para resolverlo al crear una instancia de la clase DyVParadigma, cuyo parámetro será el objeto que contenga al problema original, y posteriormente se espera recibir un objeto solución con el resultado de la clase DyVParadigma.

Posteriormente se pretende continuar con la implementación concurrente de algunos otros paradigmas importantes tales como Branch and Bound, para tratar de cubrir un objetivo principal: Generar una librería de paradigmas en JAVA que le permita al usuario resolver problemas secuenciales con una estructura paralela y así con ello obtener velocidad y eficacia en las soluciones encontradas.

REFERENCIAS

[1] Aho, Ullman; Hopcroft; The Design and Analysis of Computer Algorithms, Addison-Wesley, 61-65. 1974.
[2] Andrews, G.R.; Concurrent Programming. Principles and Practice. California, The Benjamin/Cummings Publishing Company. 1991.
[3] Andrews, G.R.; Paradigms for Process interaction in Distributed Programs. ACM Computing Surveys, Volume 15, Number 1, 49-90, 1991.
[4] Araque, F; Capel, M.; Palma, A.; Paradigms for Parallel Distributed Programming. Granada, 1995.
[5] Blelloch, Guy E.; Programming Parallel Algorithms. Comunications of the ACM. Volume 39, Number 3, 85-97, 1996.
[6] Brassard, G.; Bratley, P.; Fundamentos de Algoritmia, Madrid, Prentice-Hall. 1997.
[7] Brinch Hansen; Model Programs for Computational Science: A programming methodology for multicomputers, Concurrency: Practice and Experience, Volume 5, Number 5, 407-423, 1993.
[8] Brinch Hansen; SuperPascal- a publication language for parallel scientific computing, Concurrency: Practice and Experience, Volume 6, Number 5, 461-483, 1994.
[9] De Simone; et al; Design Patterns for Parallel Programming. PDPTA'96 International Conference; 231-240. 1997.
[10] Flanagan, David; JAVA in a nutshell; O'Reilly & Associates, USA, 1996.
[11] Gosling, James; El lenguaje de programación JAVA, Madrid, Addison-Wesley, 1997.
[12] Hartley, Stephen J.; Concurrent Programming. The JAVA Programming Language, New York, Oxford University Press, 1998.
[13] Kindervater, G.A.P.; Trienekens, H.W.J.M.; Experiments with Parallel Algorithms for Combinatorial Problems. European Journal of Operational Research, Volume 33, 65-81, 1988.
[14] Kumar, Vipin; et al; Introduction to parallel computing. Design and Analysis of Algorithms, USA, Benjamin-Cummings,1994.
[15] Lea, Doug; Concurrent Programming in JAVA. Design, Principles and Patterns; USA, Addison-Wesley, 1996.
[16] 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.
[17] Sanz Alamillo, Javier; Concurrencia y Sincronización de Threads; Sólo Programadores, año 5, Número 49, 8-14, 1998.
[18] Sanz Alamillo, Javier; Programación en JAVA usando Threads; Sólo Programadores, año 5, Número 50, 8-16, 1998.
[19] Sanz Alamillo, Javier; Utilización de Threads; Sólo Programadores, año 5, Número 48, 56-62, 1998.
[20] Sedgewick, Robert; Algorithms; USA, Addison-Wesley Publishing Company, Second Edition, 1998.