Saber, Universidad de Oriente, Venezuela.Vol. 19. Nº 1: 74-82. (2007)

# MORPHSKEL: ESQUELETO PROGRAMÁTICO BASADO EN MORFOLOGÍA MATEMÁTICA PARA EL TRATAMIENTO DIGITAL DE IMÁGENES

## MORPHSKEL: PROGRAMMING SKELETONS BASED ON MATHEMATICAL MORPHOLOGY FOR THE DIGITAL TREATMENT OF IMAGES

Juan Francisco Serrano[1], Rodolfo Campos[2]

*[1]Universidad de Oriente, Núcleo de Monagas, Programa de Ingeniería de Sistemas*
*[2] Universidad Católica Andrés Bello, Facultad de Ingeniería, Escuela de Informática*
*jserrano@sucre.udo.edu.ve / rcampos@ucab.edu.ve*

**RESUMEN**

El desarrollo de aplicaciones para el tratamiento digital de imágenes explotando paralelismo es de mucha utilidad, principalmente cuando el tiempo de respuesta es clave en la solución del problema. En este trabajo se presenta un nuevo paradigma de la programación que permite hacer más sencilla la programación paralela sin perder eficiencia en la implementación, utilizando una metodología que es un compromiso entre los extremos de la programación imperativa explícita y la programación funcional implícita. El objetivo es desarrollar un esqueleto de programación basado en morfología matemática para el tratamiento digital de imágenes. Para llevar a cabo el objetivo planteado, se introduce un esquema de programación que implementa una técnica de instanciación específica del paradigma de programación paralela que proporciona un alto nivel de abstracción, encapsulando los patrones de comunicación y primitivas de control en una sola abstracción, para así, luego del reconocimiento de las partes paralelizables e interacción con eficientes rutinas que permiten realizar múltiples operaciones morfológicas definidas en un esqueleto, el tiempo de ejecución de la aplicación se logre reducir significativamente, ocultando al usuario detalles específicos de la implementación. Se comparan los tiempos de ejecución del esqueleto en plataformas paralela y secuencial. Se muestran los resultados numéricos. Los esqueletos de programación permiten a los programadores convencionales explotar paralelismo en sus aplicaciones abstrayéndose de detalles que trunquen el eficiente desarrollo de sus soluciones, conduciendo a un estilo de programación orientada a esqueletos, identificada como una solución promisoria para el cómputo paralelo.

**Palabras claves:** Computación paralela, esqueletos de programación, esqueletos algorítmicos, tratamiento digital de imágenes, morfología matemática.

**ABSTRACT**

The development of applications for the digital treatment of images exploding parallelism it is of much utility, mainly when the response time is a key aspect in the solution of the problem. In this paper, we present a new paradigm of programming that allows making parallel programming simple without losing efficiency in the implementation, using a methodology which compromises two edges: explicit imperative programming and implicit functional programming. A skeleton programming based on mathematical morphology for the digital treatment of images is proposed. A programming scheme is introduced that implements a technique for specific instantiations of the parallel programming paradigm which provides a higher level of abstraction encapsulating the primitive patterns of communication and control in a single abstraction, thus, as soon as the recognition of the parallelizable parts and interaction of efficient routines that allow to make several morphologic operations defined in a skeleton, the run time of the application is reduced significantly, hiding the user specific details of the implementation. We compared the run times of the skeleton in parallel and sequential platforms. We present the numerical results. The programming skeletons, allow the conventional programmers to exploit parallelism in their applications, and only focus on the development of their solutions. This leads to skeleton oriented programming style which has been identified as a very promising solution for parallel computing.

**Key words:** Parallel computing, programming skeletons, skeletons algorithmic, digital treatment of images, mathematical morphology.

Recibido: diciembre 2005. Aprobado: marzo 2006.
Versión final: marzo 2006

## INTRODUCCIÓN

La necesidad de utilizar los recursos computacionales inteligentemente, ha traído de nuevo la puesta en escena con mayor protagonismo que en las décadas de los 70 y 80, la computación paralela. La programación paralela es una técnica de programación basada en la ejecución simultánea, bien sea en un mismo computador con uno o varios procesadores, o en un *Cluster* de computadores, en cuyo caso se denomina computación distribuida. En términos generales, los sistemas multiprocesadores y multicomputadores consiguen un aumento del rendimiento si se utilizan estas técnicas, no obstante, en los sistemas monoprocesador el beneficio en rendimiento no es tan evidente ya que la unidad central de procesamiento es compartida por múltiples procesos en el tiempo.

El mayor problema de la computación paralela radica en la complejidad de sincronizar unas tareas con otras, ya sea mediante secciones críticas, semáforos o paso de mensajes para garantizar la exclusión mutua en las zonas del código en las que sea necesario. La meta, reducir al mínimo el tiempo total de cómputo distribuyendo la carga de trabajo entre los procesadores disponibles. Para fines prácticos, es necesario considerar que la paralelización es un factor básico para tener un alto desempeño en los equipos de cómputo y una de las razones principales para utilizar el paralelismo en el diseño de hardware o software es obtener un alto rendimiento o simplemente mayor velocidad al ejecutar un programa.

En el área del paralelismo, uno de los problemas críticos es el procesamiento digital de imágenes. Las nuevas tecnologías sobre todo las referentes a tiempo real, requieren de grandes sistemas capaces de procesar imágenes de gran tamaño en muy corto tiempo. El uso de *Clusters* y *Grids* para esta tarea se ha visto incrementado sustancialmente en aras del uso adecuado de los recursos computacionales. Diversos métodos son propuestos día a día con la intención de procesar imágenes de manera óptima. El concepto de morfología matemática introducido por Matheron y Serra, ha creado un gran impacto entre los investigadores del área desde su concepción, debido al amplio espectro de utilidades que esta técnica introduce en el campo de las imágenes y su procesamiento. La morfología matemática se refiere a un conjunto de análisis y procesamiento de imágenes no lineales que se concentra en la estructura geométrica dentro de una imagen (Dougherty, 1992). Ésta, ofrece dos operaciones básicas: erosión y dilatación, las cuales permiten definir un sin fin de operaciones derivadas de ellas con tan solo la inclusión de operadores de matrices básicas.

La programación paralela introduce fuentes adicionales de complejidad. Si se tuviese que programar al nivel más bajo no solo se incrementaría el número de instrucciones de computadora, sino que también se deberían manejar explícitamente la ejecución de miles de instrucciones y coordinar millones de interacciones entre los procesos. En consecuencia, la abstracción y la modularidad son tanto o más importantes en la programación paralela que en la secuencial. Este alto nivel de complejidad puede ser estudiado a través de cuatro requerimientos fundamentales: modularidad, concurrencia, escalabilidad y localidad; requerimientos básicos que debe manejar un programa paralelo y que deben ser administrados para poder manipular adecuadamente la concurrencia, la localidad, además de ser organizados de tal forma que la aplicación sea escalable y modular, necesitándose alteraciones que sean lo suficientemente simples como para poder trabajar con los diferentes modelos arquitecturales (Foster, 1995). Nuevas técnicas y tecnologías han sido desarrolladas buscando los mismos objetivos: transparencia, portabilidad y escalabilidad.

Diversos conceptos han sido retomados y otros puestos en escena por primera vez, entre ellos el concepto de Esqueletos de Programación, *Programming Skeletons*, a comienzos de los 90 (Cole, 1989). Cole pone de manifiesto que un esqueleto es una abstracción algorítmica común a una serie de aplicaciones que pueden ser implementadas en paralelo. Los esqueletos son embebidos dentro de un lenguaje anfitrión secuencial y ellos son la única fuente de paralelismo en el programa (Botorog y Kuchen, 1995). Se busca con ello, apartar al programador de los detalles generalmente engorrosos referentes al paralelismo y explotar las bondades secuenciales del lenguaje anfitrión. Los esqueletos se utilizan básicamente para presentar los principios de la programación paralela. Actualmente, se realizan significativos esfuerzos por estandarizar los esqueletos paralelos.

Los problemas ordinarios pueden ser resueltos como programas paralelos, aplicando algún esqueleto de programación que se ajuste. Esta característica de reutilidad del mismo esquema algorítmico es una de las ventajas más evidentes del uso de los esqueletos de programación, simplemente porque capturan la forma algorítmica común de los problemas.

Los esqueletos de programación son vistos como la construcción de la programación paralela de alto nivel, necesarios para que el programador de sistemas paralelos pueda trabajar fácilmente (Duncan, 1996). Estas construcciones son necesarias porque la programación

paralela es más compleja que la programación secuencial debido a los mayores grados de libertad que la envuelve. La programación secuencial implica solamente un hilo de cómputo en cualquier instante de tiempo, mientras que la programación paralela implica múltiples hilos de cómputo los cuales necesitan comunicarse y sincronizarse unos con otros. Esta complejidad se incrementa con el procesamiento masivamente paralelo, el cual aumenta el número de hilos que se ejecutan concurrentemente.

Los esqueletos algorítmicos corresponden a una programación de alto nivel en términos de la clasificación propuesta por McColl (1993) para modelos de cómputo paralelo, pues proporcionan una notación para la descripción precisa y correcta de métodos eficientes para la solución de problemas computacionales (McColl, 1993). De tal manera, los esqueletos algorítmicos pueden ayudar a reducir errores de programación paralela. El programador cuenta con una caja de herramientas, llena de abstracciones para solucionar sus problemas.

En el presente trabajo se desarrollo un esqueleto de programación, MorphSkel, que permite la aplicación de operaciones básicas de morfología matemática binaria para el tratamiento digital de imágenes. Para ello, fue implementado un algoritmo esquelético diseñado bajo el enfoque de paralelismo de datos Darlington *et al.* (1993), desarrollado sobre: Mpich, MpiJava y J2EE (Java Platform Enterprise Edition). Se obtuvieron resultados satisfactorios en cuanto al rendimiento de la aplicación en ambientes multiprocesador multihilos en términos del tiempo de ejecución en relación con el ambiente secuencial, por lo que se cuenta con una herramienta computacional funcional orientada a esqueletos paralelos que permitirá darle solución a problemas de características similares, como por ejemplo, problemas matriciales donde se pueda particionar el conjunto inicial de datos.

## METODOLOGÍA

La idea central es sustituir la programación paralela explícita, por la selección e instanciación de una variedad de formas algorítmicas paralelas pre-empaquetadas conocidas como esqueletos. Los esqueletos son funciones polimórficas de orden superior, cuya parte estática dentro de un programa es la descripción del algoritmo y su parte dinámica se puede describir usando estrategias apropiadas para el problema (Cole, 1999). Combinando los esqueletos con estrategias de programación eficientes, se pueden obtener implementaciones funcionales del paradigma de la programación paralela. La metodología se puede analizar en tres componentes principales: el diseño del esqueleto, el modelo de funcionamiento y la transformación del programa.

* Diseño del esqueleto. El esqueleto captura la representación algorítmica común de formas paralelas para su posterior implementación en la construcción de aplicaciones paralelas. En este trabajo, el esqueleto se ha desarrollado empleando una forma de orden superior definido como un algoritmo *FARM Skeleton*, Figura 1 (Darlington *et al.* 1993). El FARM Skeleton captura la forma más simple del paralelismo de datos. Una función se aplica a cada una de los trabajos, dicha función toma el ambiente en el cual se representan los datos que son comunes a todos los trabajos y el paralelismo es alcanzado utilizando procesadores múltiples. Su significado es independiente de cualquier implementación particular del esqueleto, esto permite que el programa esquelético sea rápidamente un prototipo sobre plataformas secuenciales y absolutamente portable entre diversas plataformas paralelas. Las funciones a las cuales se aplica el esqueleto, se ejecutan secuencialmente. Todos los aspectos del comportamiento paralelo del esqueleto, se tratan explícitamente durante la implementación.

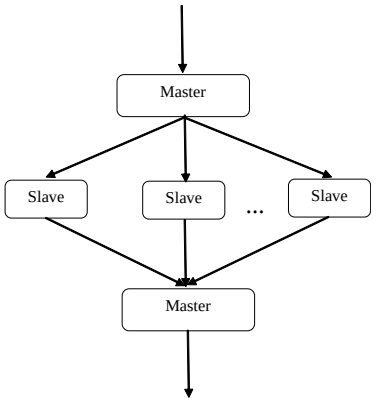


Figura 1. Farm Darlington *et al.* Skeleton Model.

* Modelo de funcionamiento. El esqueleto tiene asociado un modelo de funcionamiento que le permite implementarlo de manera razonablemente eficiente, logrando una ejecución adecuada del mismo en términos de rendimiento del programa paralelo. La información generada producto de las distintas observaciones realizadas durante la ejecución del programa, es utilizada para modificar la decisión de la asignación de recursos a los distintos procesos. Se implementa el modelo de Granja de Tareas, *Task Farming* Foster (1995), para el procesamiento paralelo, Figura 2. Se estructura el proceso de diseño en cuatro etapas distintas: particionamiento (*split*), establecimiento de la correlación de datos particionados (*map*), resolución (*solve*) y unión de los subresultados y generación de la solución (*join*). El paralelismo es alcanzado utilizando múltiples procesadores para evaluar los trabajos, obteniéndose resultados que nos permitan asignar eficientemente una distribución de carga para mejorar el rendimiento de la aplicación en términos del tiempo total de ejecución.

* Transformación del programa. La transformación proporciona una ruta natural a la portabilidad del programa escrito en términos de que un esqueleto, para ser implementado en diferentes arquitecturas. La transformación de MorphSkel se expresa durante el proceso de desarrollo en todos los niveles, considerando la portabilidad proporcionada por la naturaleza de alto nivel de las especificaciones originales del programa. Para el desarrollo de MorphSkel se utilizó Java como lenguaje anfitrión, debido a las innumerables bondades que éste ofrece para el tratamiento digital de imágenes, además de las características propias del lenguaje como: portabilidad, escalabilidad y orientación a objetos (herencia, polimorfismo y encapsulamiento), esta particularidad hace MorphSkel una aplicación que puede implementarse en cualquier arquitectura. La versión específica del Java Development Kit (JDK) y Java Runtime Environment (JRE) utilizada fue la J2EE 1.4 SDK de Sun Microsystems. La versión de MpiJava utilizada fue la 1.2.5 sobre mpich 1.1.2.

El desempeño de MorphSkel se verifica en varios escenarios sobre una imagen. Cada escenario se expresa en términos de la distribución de la carga según el número de procesos esclavos asignados. Se efectúan varias ejecuciones de MorphSkel generándose en cada una de ellas el tiempo total de procesamiento de la imagen, basado en las operaciones morfológicas de Apertura y Cierre.

## RESULTADOS COMPUTACIONALES

El producto final del esqueleto de programación puede representarse mediante cinco clases, las cuales son presentadas en la Figura 3, a través de un Diagrama de Clases UML (*Unified Modeling Language*) el cual proporciona una forma estándar para escribir los planos del sistema, cubriendo lo conceptual y funcional del mismo. La clase principal de la aplicación es Function, esta clase representa la interfaz entre el desarrollador que pretende utilizar morfología matemática de forma paralela y el resto de las clases. La clase cuenta con varios métodos bases, un método constructor y métodos *getters* y *setters* que permiten introducir los valores que condicionan la ejecución del programa.

Se crearon dos clases: Img2Msp y Msp2Img, para la conversión de imágenes de formato PNG (*Portable Network Graphics*), JPG (*Joint Photographic Expert Group*) y GIF (*Graphics Interchange Format*) a MSP (*MorphSkel Pictures*) y viceversa. Ambos programas trabajan por consola siguiendo el esquema: *java Programa archivo-fuente archivo-destino.* El formato convencional de imágenes por defecto es PNG y en el caso de que se deseara convertir a otro formato (JPG o GIF) este debe especificarse como argumento al utilizar Msp2Img antes del archivo-fuente.

Las pruebas experimentales se realizaron sobre una arquitectura secuencial y sobre una arquitectura paralela correspondiente a un *Cluster* de seis nodos, conectados por switches de alta velocidad. Para evaluar el desempeño de MorphSkel, se aplicó sobre imágenes en formato PNG de 12 Kb (900x900 px/px) y 112 Kb (1024x1024 px/px), la operación de OPENCLOSE, definida como:

OPENCLOSE (f) = CLOSING (OPENING (f, g), ROTATION (g))

La operación aplicada puede traducirse de la siguiente manera: realizar primero la Apertura de f tomando como elemento estructurante g y al resultado obtenido aplicar una Cerradura con la rotación de g (-g) como elemento estructurante. La Tabla 1 y la Tabla 2 muestran los resultados obtenidos en términos de la distribución de la carga, número de procesos esclavos utilizados y el tiempo total de procesamiento. Adicionalmente, se han agregado los tiempos parciales notados en el sistema para las operaciones principales realizadas durante la ejecución, a saber: Apertura y Cierre. Es importante mencionar que el tiempo de lectura y escritura de los archivos de imágenes no ha sido tomados en consideración, éste es de aproximadamente 1,5 seg. y 15,5 seg. respectivamente.

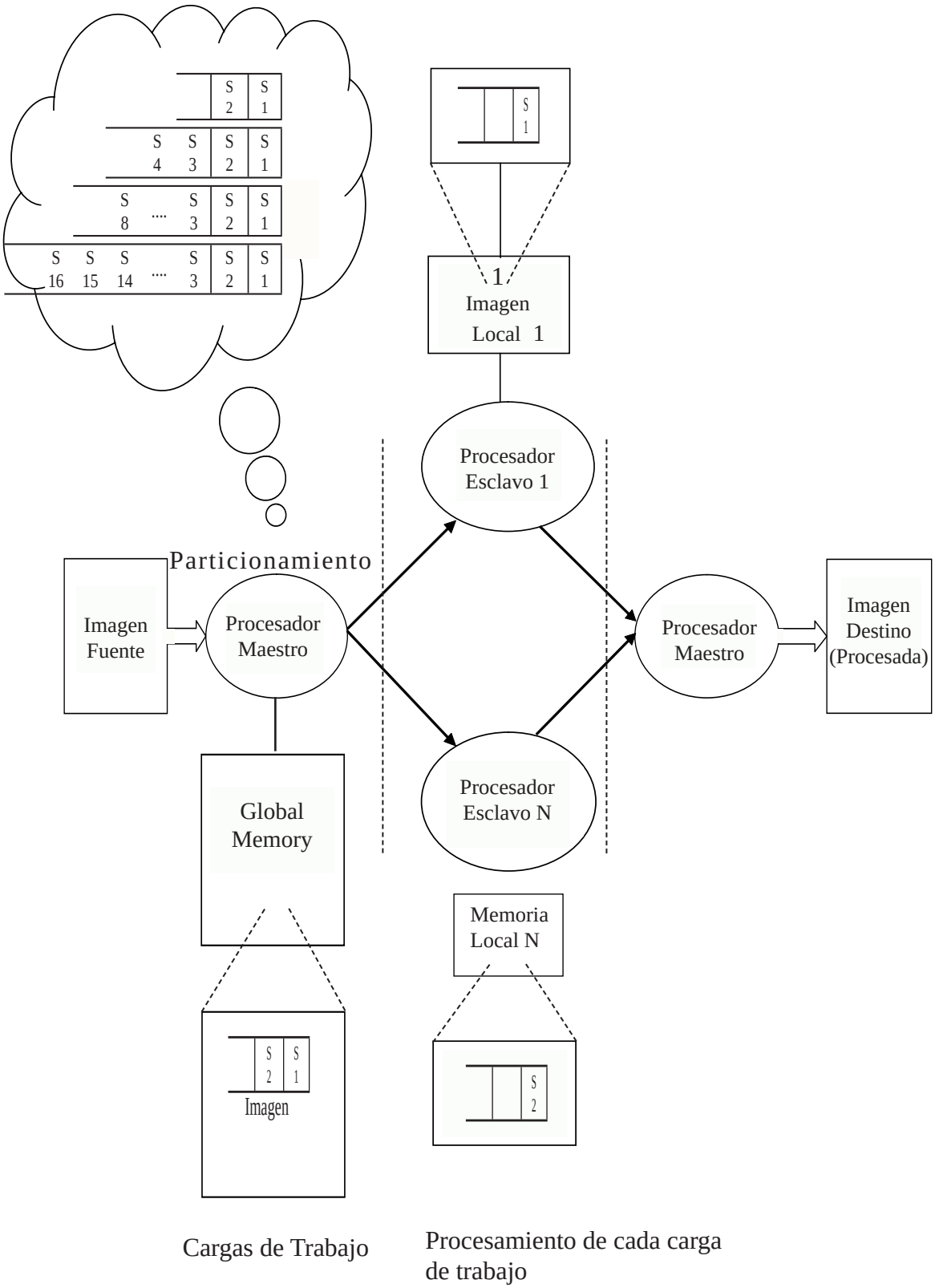


Figura 2. Modelo Task Farming de Procesamiento Paralelo.

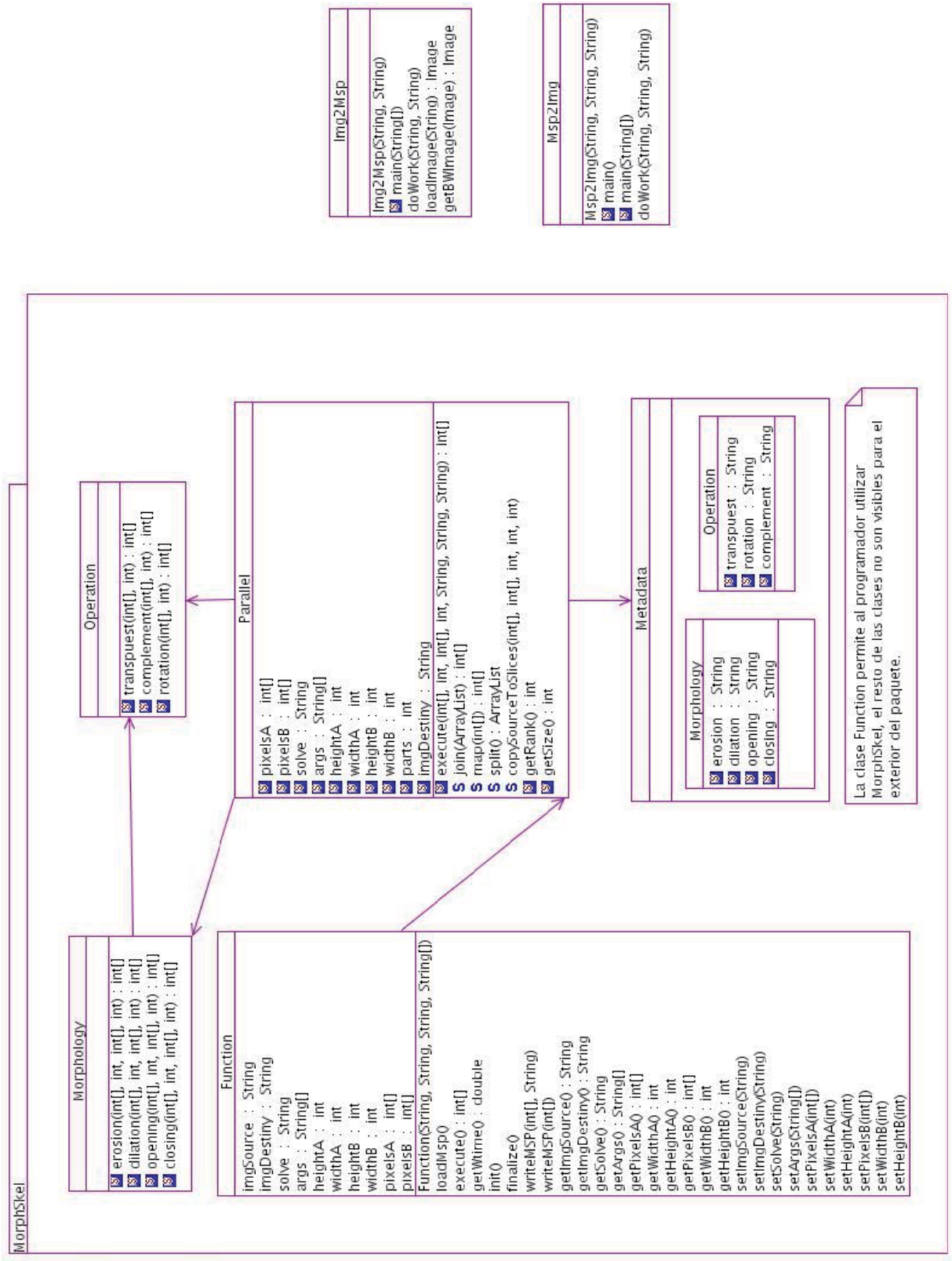


Figura 3. Diagrama de clases de MorphSkel.

Tabla 1. Resultados obtenidos del proceso de apertura y cierre en función del número de procesos para la imagen de 900x900 px/px.

| Prueba | Número de Procesos Esclavos | Tiempo Parcial (seg.) | | Tiempo Total (seg.) |
|---|---|---|---|---|
| 1 | 1 | Apertura<br>Cierre | 1.30020<br>0,95080 | 2,25100 |
| 2 | 4 | Apertura<br>Cierre | 1,21090<br>0,91669 | 2,12760 |
| 3 | 10 | Apertura<br>Cierre | 1,04392<br>1,00459 | 2,00918 |
| 4 | 12 | Apertura<br>Cierre | 1,02890<br>1,01315 | 2,04205 |
| 5 | 14 | Apertura<br>Cierre | 1,00472<br>1,29941 | 2,09405 |
| 6 | 20 | Apertura<br>Cierre | 0,90464<br>1,18941 | 2,30413 |

Tabla 2. Resultados obtenidos del proceso de apertura y cierre en función del número de procesos para la imagen de 1024x1024 px/px.

| Prueba | Número de Procesos Esclavos | Tiempo Parcial (seg.) | | Tiempo Total (seg.) |
|---|---|---|---|---|
| 1 | 1 | Apertura<br>Cierre | 5,00500<br>4,32978 | 9,33478 |
| 2 | 4 | Apertura<br>Cierre | 5,00029<br>3,02621 | 8,02650 |
| 3 | 10 | Apertura<br>Cierre | 4,00284<br>1,97717 | 5,98001 |
| 4 | 14 | Apertura<br>Cierre | 3,30041<br>1,70082 | 5,00123 |
| 5 | 20 | Apertura<br>Cierre | 2,59000<br>1,57000 | 4,16700 |
| 6 | 40 | Apertura<br>Cierre | 3,10083<br>1,60917 | 4,71000 |

## DISCUSIÓN

La aplicación MorphSkel se ejecuto sobre un *Cluster* Linux de seis (6) máquinas homogéneas y se explotó el paralelismo mediante la técnica de particionamiento de datos al implementar el modelo *Task Farming,* figura 2. La comunicación utilizada es de tipo Maestro-Esclavo. El Maestro se encarga de leer los archivos MSP para cargar los datos de la imagen A y la imagen B, que representan en el caso de la morfología matemática, la imagen y su elemento estructurante. Luego realiza el particionamiento de datos considerándose el efecto de solapamiento, y envía sincrónicamente un arreglo de datos a cada uno de los esclavos. Los esclavos procesan la información aplicando la operación especificada y envían de vuelta la data procesada sincrónicamente al maestro quien une los resultados obtenidos y devuelve un resultado final al usuario de la aplicación. Las operaciones de particionamiento (*split*), establecimiento de la correlación de datos particionados (*map*), resolución (*solve*) y unión de los subresultados y generación de la solución (*join*) son inherentes al algoritmo *Task Farming,* base del esqueleto (Darlington *et al.*1993).

Para la imagen de 900x900 px/px, al realizar las primeras dos pruebas se pudo observar una mejora de

0,118 seg. aproximadamente con relación al tiempo total de procesamiento, lo que nos indica que la distribución de cargas de 4 a 10 procesos mejora el rendimiento de la aplicación. Ahora bien, al duplicar el número de procesos esclavos a 20, se observó un descenso en la eficiencia de 0,294 seg. Estos resultados permiten establecer un límite práctico en cuanto a la distribución de la carga para imágenes de 900x900 *pixels*, distribución en la cual se obtiene el mejor rendimiento de MorphSkel.

Para la imagen de 1024x1024 px/px, se pudo observar como el tiempo de respuesta es cada vez menor a medida que se aumenta la cantidad de procesos esclavos, obteniéndose un mejor tiempo de ejecución de la aplicación, estableciéndose un intervalo para la distribución de carga entre 20 – 40 procesos esclavos para alcanzar un rendimiento satisfactorio de MorphSkel, lográndose apreciar la ventajas del esqueleto de programación al evaluarlo con respecto a la ejecución secuencial, al considerar como métrica de rendimiento el tiempo de ejecución.

Puede notarse en la tabulación de los resultados luego de ejecutar MorphSkel en el ambiente paralelo, que al asignar un número inicial pequeño de procesos esclavos, los tiempos de procesamiento presentan ciertas variaciones con tendencia a incrementarse, para luego notar que en un rango de procesos asignados, se alcanza el mejor rendimiento de la aplicación. Posteriormente, al aumentar la distribución de la carga, es decir, el número de procesos esclavos, se hace más notorio el aumento del tiempo de procesamiento. Esta característica se evidencia por la presencia de retardos de propagación durante la comunicación entre procesos. Sin embargo, en ambos casos, se observa que la aplicación de las operaciones de morfología matemática brinda un mejor rendimiento en ambientes paralelos, apreciándose mejor en imágenes de grandes dimensiones.

## CONCLUSIONES

Con el desarrollo del presente trabajo se presenta un paradigma de la programación paralela, que permite al programador no preocuparse por los detalles de bajo nivel, lo cual no afecta la estructura del programa, ni su implementación. La clave para un eficiente desempeño paralelo a través de este esquema de programación, es asegurar durante la definición del esqueleto una estructura computacional paralela para la cual se conoce una implementación eficiente, en este caso, el modelo *Task Farming* para el procesamiento paralelo.

En particular, se muestra a través de las pruebas realizadas un excelente rendimiento de MorphSkel en ambientes paralelos, al considerar su ejecución con relación a ambientes secuenciales donde un solo proceso esclavo interviene durante su ejecución, lo que se observa al cuantificar los tiempos totales de procesamiento en ambos ambientes. Cabe destacar, que al implementar el modelo *Task Farming* para el procesamiento paralelo en el diseño de MorphSkel, éste adiciona tiempos de propagación (*delay*) ocasionados por el envío asíncrono de data entre el maestro y los esclavos, lo cual aumenta el tiempo de procesamiento, sin embargo, el buen rendimiento de MorphSkel se mantiene.

MorphSkel muestra una aplicación que por medio de un pequeño conjunto de operaciones básicas de morfología matemática y matrices binarias, brinda a cualquier programador la posibilidad de realizar un sin fin de operaciones sobre imágenes digitales explotando el paralelismo, sin tener que lidiar con detalles complejos. Múltiples operaciones de morfología y matrices pueden ser introducidas a MorphSkel y nuevos conceptos de comunicación de datos, asíncronos y síncronos, pueden ser manejados en trabajos posteriores, además, se podrían realizar estudios para comprobar los valores exactos de retardo debido al envío de mensajes a través de la red. Finalmente podemos concluir que el uso de esta técnica es una herramienta muy valiosa y representa una excelente solución para los programadores de sistemas paralelos, por lo que su uso podría verse incrementado de manera abrumadora.

## REFERENCIAS BIBLIOGRÁFICAS

Botorog G.; Kuchen H.1995. Algorithmic Skeletons in an Imperative Language for Distributed Programming. Report 9504, Universidad de Giessen, Alemania.

Cole Murray. 1989. Algorithmic Skeletons: Structured Management of Parallel Computation MIT Press, pp 25-77.

Cole Murray. 1999. Algorithmic Skeletons, In: Hammond, K., Michaelson, G., eds., Research Directions in Parallel Functional Programming, Springer-Verlag, pp 289-303.

Darlington John; Field, A.J.; Harrison, P.G.; Kelly, P.H.J.; Sharp, D.W.N.; Wu, Q.; While, R.L. 1993.

Parallel Programming Using Skeleton Functions In: Proceedings de PARLE 93, Springer-Verlag, LNCS Volumen 694: 146-160.

DARLINGTON JOHN; YI-KE GUO; HING WING TO; JIN YANG 1995. Parallel Skeletons for Structured Composition, ACM SIGPLAN Notices, Volumen 30 (8):19-28.

DOUGHERTY E. 1992. An Introduction to Morphological Image Processing. Publication of SPIE, Washington, USA, pp 33-69.

DUNCAN K. G. CAMPBELL 1996. Towards the classification of Algorithmic Skeletons. Department of Computer Science, University of York, 01-12-2004. http://citeseer.ist.psu.edu/campbell96towards.html

FOSTER IAN 1995. Designing and Building Parallel Program. Addisson Wesley Longman Publishing Co, Inc Boston, MA, USA, pp 32-48.

J2EE. Java 2 Platform, Enterprise Edition 1.4. 25-9-2004 http://java.sun.com/j2ee/1.4/

MCCOLL W.1993.An architecture Independent Programming Model for Scalable Parallel Computing. British Computer Society Parallel Processing Specialist Group. University of Westminster, pp 1-17.

MOORE G. 1965. Cramming More Components Onto Integrated Circuits, Electronics Volumen 38 (8): 114-117.

MPICH-A Portable Implementation of MPI. 25-9-2004. http://www-unix.mcs.anl.gov/mpi/mpich/.

MPIJAVA. MpiJava Home Page. 25-09-2004. http://www.hpjava.org/mpiJava.html

SERRANO J. 2005. Desarrollo de una librería de procedimientos portable imagelib para el tratamiento digital de imágenes. caso de estudio: Técnicas de segmentación. Trabajo de Ascenso, Dpto. de Matemáticas, Programa de Licenciatura en Informática, U.D.O. Cumaná, Venezuela, pp 7-68.