Universidad Politécnica del Estado de Morelos

ESTUDIO DEL PASO DE MENSAJES CON ESTRUCTURAS DE DATOS COMPLEJAS EN FUNCIONES DE VECINDAD APLICADO A UN AMBIENTE CLUSTER

T E S I N A

Que para obtener el título de:

INGENIERO EN INFORMÁTICA

P r e s e n t a

RENÉ LÓPEZ RUIZ

Directores de Tesina

INTERNO: M.C. IRMA YAZMIN HERNÁNDEZ BÁEZ
EXTERNO: DR. MARCO ANTONIO CRUZ CHÁVEZ

Jiutepec, Morelos Febrero de 2011

Universidad
Morelos **Politécnica**

INGENIERIA EN INFORMATICA

Jiutepec, Morelos a 18 de Febrero del 2011

AUTORIZACIÓN DE IMPRESIÓN DE TESINA

Los abajo firmantes, miembros del jurado para la evaluación del proyecto de estadía del alumno: **LÓPEZ RUIZ RENÉ**, manifiestan que después de haber revisado su tesina titulada **"ESTUDIO DEL PASO DE MENSAJES CON ESTRUCTURAS DE DATOS COMPLEJAS EN FUNCIONES DE VECINDAD APLICADO A UN AMBIENTE CLUSTER"**, realizado bajo la dirección del **M.C. IRMA YAZMÍN HERNÁNDEZ BAÉZ**, el trabajo se **ACEPTA** para proceder a su impresión.

ATENTAMENTE

M.C. Irma Y. Hernández Báez
Presidenta
Cédula Profesional: 4074354

MC. Alma Delia Nieto Yáñez
Secretaria
Cédula Profesional: 4074359

MC. Edgardo González Hernández
Vocal
Cédula Profesional: 32199991

Dr. Marco Antonio Cruz Chávez
Asesor externo
Cédula Profesional: 1562862

c.c.p M. en A Liliana Márquez Mundo
Lic. Norma Zavala Díaz / Departamento de Servicios Escolares
Expediente

AGRADECIMIENTOS

A Dios en primer lugar, por la ayuda que me ha brindado en el transcurso de mi vida.

Durante todos estos años he conocido y compartido momentos con muchas personas que me han apoyado, no solo en lo académico, sino también en los personal. A todas ellas, y sin dejar a nadie en el olvido, quiero agradecerles su tiempo, sus palabras y su apoyo. Gracias.

A la M.C Irma Y. Hernández Báez Y al DR. Marco A. Cruz Chávez, directores de este trabajo de investigación, por encausar por buen camino el desarrollo del trabajo.

A mis sinodales M.C. Edgardo González Hernández y la M.C. Alma D. Nieto, por sus consejos y comentarios para la culminación de este trabajo.

DEDICATORIAS

A Dios por brindarme la vida, mi familia que tanto amo y las oportunidades de superación.

A mis padres Luis López Gómez y Paulina Ruiz Vázquez, porque ellos fueron mis motores que siempre me hicieron poder salir adelante y conseguir todo lo que me propusiera.

A Olga L. Cruz Miranda, por su ayuda y apoyo incondicional, así como por su comprensión en tiempos difíciles.

A mis hermanos Luis A. López, Karla López y María M. López, que siempre me dieron palabras de aliento para seguir adelante.

A compañeros y amigos con los que compartí mi estancia en la UPEMOR, por los gratos momentos que hicieron que este tiempo quedara grabado en mi mente.

Tabla de Contenido

LISTA DE FIGURAS IV
LISTA DE TABLAS VI
LISTA DE CODIGOS VII
RESUMEN IX
CAPÍTULO I INTRODUCCIÓN 1
1.1 Antecedentes 2
1.2 Planteamiento del problema 3
1.3 Panorama general del proyecto 4
1.4 Objetivo general 4
1.5 Objetivos específicos 4
1.6 Justificación 5
1.7 Alcances y limitaciones 5
1.8 Organización de la Tesina 6
CAPÍTULO II ESTADO DEL ARTE 8
2.1. Introducción 9
2.2. Cómputo paralelo 10
2.3. Plataformas de cómputo paralelo 11
2.3.1 Estructura de control en plataformas paralelas 11
2.3.2 Modelo SPMD: Single Program Multiple Data 13
2.4. Programación con Paso de Mensajes 14
2.4.2 Aspectos de diseño 15
2.4.3 PVM 16
2.4.4 MPI 17
2.4.5 Ejemplo con MPI 21
2.5. Cluster 23
2.5.1 Arquitectura de un Cluster 23
2.5.2 Componentes del Cluster 25
2.5.3 Cluster middleware 26
CAPÍTULO III ANÁLISIS DEL PROBLEMA Y DISEÑO DE LA SOLUCIÓN 27
3.1 Introducción 28

3.2 Metodología 29
3.3 Análisis del problema 29
3.4 Diseño de la solución 33
CAPÍTULO IV IMPLEMENTACIÓN 43
4.1 Implementación del algoritmo 44
CAPÍTULO V PRUEBAS Y RESULTADOS 58
5.1 Pruebas 59
5.2 Análisis de resultados 69
CAPÍTULO VI CONCLUSIONES 73
REFERENCIAS 76

LISTA DE FIGURAS

Capítulo II Estado del arte

Figura 2. 1 Arquitectura típica SIMD (a) y una arquitectura típica MIMD (b) [Grama *et al*., 2003] 12
Figura 2. 2 Estructura básica de un programa SPMD 13
Figura 2. 3 Arquitectura de un sistema PVM [Geist *et al*, 2000] 17
Figura 2. 4 Vista de una comunicación asíncrona [Grama *et al*., 2003] 18
Figura 2. 5 Salida del programa "hola mundo" en MPI 23
Figura 2. 6 Arquitectura simplificada de un solo equipo 23
Figura 2. 7 Arquitectura simplificada de un *Cluster* 24
Figura 2. 8 Componentes de Hardware y Software de un *Cluster* [Maozhen y Baker, 2005] 25

Capítulo III Análisis del problema y diseño de la solución

Figura 3. 1 Envío de datos del nodo maestro a los nodos esclavos 32
Figura 3. 2 Procesamiento de datos en nodos esclavos 32
Figura 3. 3 envío de datos de los nodos esclavos al nodo maestro 33
Figura 3. 4 Algoritmo con paso de mensajes con estructuras de datos complejas en un ambiente *Cluster*. 35
Figura 3. 5 Vista de la memoria con desplazamiento 39

Capítulo IV Implementación

Figura 4. 1 Ejemplo de estructuras de datos simples 44
Figura 4. 2 Ejemplo de estructura de datos compleja 44
Figura 4. 3 Representación grafica del arreglo de estructuras de datos complejos 44
Figura 4. 4 Desplazamiento de memoria de la estructura sxx 46
Figura 4. 5 Desplazamientos de memoria de la estructura maq 48
Figura 4. 6 Desplazamientos en memoria de la estructura de datos trab 50
Figura 4. 7 Desplazamiento en memoria de la estructura de datos compleja (sched) 53

Capítulo V Pruebas y resultados

Figura 5. 1 Visualización de la prueba ejecutada “Programación paralela”.... 63
Figura 5. 2 Visualización de la prueba ejecutada “Comunicación bloqueante, memoria estática” 64
Figura 5. 3 Visualización de la prueba ejecutada “Comunicación no bloqueante, memoria estática” 65
Figura 5. 4 Visualización de la prueba ejecutada “Comunicación bloqueante, memoria dinámica” 67
Figura 5. 5 Visualización de la prueba ejecutada “Comunicación no bloqueante, memoria dinámica” 69
Figura 5. 6 Comunicación con memoria estática no bloqueante 70
Figura 5. 7 Comunicación con memoria estática no bloqueante 71
Figura 5. 8 Comunicación con memoria dinámica 72

LISTA DE TABLAS

Capítulo I Introducción

Tabla 1. 1 Recursos del *Cluster* CIICAp. 6

Capítulo II Estado del arte

Tabla 2. 1 Correspondencia entre tipos de datos soportados por MPI y el lenguaje C 20

Capítulo III Análisis del problema y diseño de la solución

Tabla 3. 1 Tipos de datos primitivos y sus desplazamientos de memoria....... 39

Capítulo V Pruebas y resultados

Tabla 5. 1 Prueba para verificar la paralización del programa 60

Tabla 5. 2 Prueba que verifica la no pérdida de datos en la comunicación bloqueante... 60

Tabla 5. 3 Prueba que verifica la no pérdida de datos en la comunicación no bloqueante con memoria estática 60

Tabla 5. 4 Prueba que verifica la no pérdida de datos en la comunicación bloqueante con memoria dinámica 61

Tabla 5. 5 Prueba que verifica la no pérdida de datos en la comunicación no bloqueante con memoria dinámica 61

Tabla 5. 6 Elementos requeridos para realizar las pruebas 62

Tabla 5. 7 Prueba ejecutada “Programación paralela” 63

Tabla 5. 8 Prueba ejecutada “Comunicación bloqueante, memoria estática” 64

Tabla 5. 9 Prueba ejecutada “Comunicación no bloqueante, memoria estática” 65

Tabla 5. 10 Prueba ejecutada “Comunicación bloqueante, memoria dinámica” 66

Tabla 5. 11 Prueba ejecutada “Comunicación no bloqueante, memoria dinámica” 68

LISTA DE CODIGOS

Capítulo II Estado del arte

Código 2. 1 Ejemplo de comunicación con paso de mensajes......................... 15
Código 2. 2 Programa "Hola mundo" en MPI... 22
Código 2. 3 Compilación del programa "hola mundo.c" en MPI 22
Código 2. 4 Comando para ejecutar un programa MPI 22

Capítulo III Análisis del problema y diseño de la solución

Código 3. 1 Envió individual de datos del mismo tipo en diferentes mensajes 30
Código 3. 2 Envió organizado de datos del mismo tipo en un arreglo y un mensaje 30
Código 3. 3 Ejemplo de la declaración de una estructura de datos................. 37
Código 3. 4 Código para acceder a un miembro de una estructura................. 37
Código 3. 5 Ejemplo de los desplazamientos de memoria 38
Código 3. 6 Ejemplo de la declaración de una estructura de datos compleja 41

Capítulo IV Implementación

Código 4. 1 Declaración de las estructuras de datos simples.......................... 45
Código 4. 2 Declaración del arreglo de bloques para la estructura sxx........... 46
Código 4. 3 Declaración del arreglo desplazamientos para la estructura sxx. 47
Código 4. 4 Declaración del arreglo tipos de datos para la estructura sxx...... 47
Código 4. 5 Declaración de la estructura sxx en MPI 47
Código 4. 6 Sentencia que da a conocer el nuevo tipo de dato al compilador MPI 47
Código 4. 7 Declaración del arreglo de bloques para la estructura maq......... 48
Código 4. 8 Declaración del arreglo desplazamientos para la estructura maq48
Código 4. 9 Declaración del arreglo tipos de datos para la estructura maq ... 48
Código 4. 10 Declaración de la estructura maq en MPI.................................. 49
Código 4. 11 Declaración del arreglo de bloques para la estructura trab....... 49
Código 4. 12 Declaración del arreglo desplazamientos para la estructura trab 50
Código 4. 13 Declaración del arreglo tipos de datos para la estructura trab.. 50

Código 4. 14 Declaración de la estructura trab en MPI 51
Código 4. 15 Declaración de la estructura de datos compleja para ANSI C.... 51
Código 4. 16 Declaración del arreglo de bloques para la estructura sched.... 52
Código 4. 17 Declaración del arreglo desplazamientos para la estructura sched 53
Código 4. 18 Declaración del arreglo tipos de datos para la estructura sched 53
Código 4. 19 Declaración de la estructura sched en MPI 53
Código 4. 20 Sintaxis del operador new 54
Código 4. 21 Sintaxis del operador delete 55
Código 4. 22 Sintaxis de la estructura de datos compleja con memoria dinámica 55
Código 4. 23 Arreglo de estructuras complejas 55
Código 4. 24 Asignación de memoria dinámica a las estructuras de datos complejas 56
Código 4. 25 Sintaxis de la barrera (MPI_Barrier) 57
Código 4. 26 Sintaxis de la barrera (MPI_Wait) 57

RESUMEN
En este trabajo de investigación se desarrolló un algoritmo con paso de mensajes, para el cual se utilizo la librería MPI (del inglés: Message Passing Interface), con el objeto de proveer un procedimiento para poder mandar información encapsulada en estructuras de datos complejas. Se propuso e implemento un procedimiento para la creación y asignación de nuevos tipos de datos, lo que permitió hacer posible el envío y recepción de datos en los nodos que componen a un Cluster.

Se tomó el modelo de comunicación paralela SPMD (del inglés: Single Process, Multiple Data), que es una técnica empleada para lograr el paralelismo. En este trabajo se realizaron dos algoritmos, con la diferencia que uno de ellos asigna el tamaño de las estructuras de datos de manera dinámica y el otro se asigna forma estática, con la finalidad de encontrar la mejor solución posible al problema tratado.

Las pruebas experimentales se realizaron tomando como plataforma el Cluster del CIICAp de la Universidad Autónoma a del Estado de Morelos, se ejecutaron los dos algoritmos bajo condiciones similares. De acuerdo al análisis experimental realizado, el algoritmo propuesto que trabaja con memoria estática demostró no tener perdida de información en el envío y recepción por el paso de mensajes. Por otro lado, el algoritmo con el cual se trabajó la asignación de memoria dinámica, no obtuvo los mismos resultados, los datos que se envían a los nodos del Cluster no llegan a su destino.

CAPÍTULO I INTRODUCCIÓN

1.1 Antecedentes

La evolución de las computadoras ha estado siempre ligada a la evolución de la tecnología y a las necesidades de las aplicaciones computacionales. El área de las ciencias de la computación y la ingeniería demanda cada vez más recursos de memoria y mayores velocidades de operación. Estas necesidades de mejora han originado el desarrollo de arquitecturas con mejores prestaciones. En esta evolución de las computadoras, las arquitecturas son un paso inevitable en la consecución de mayores velocidades de procesamiento [Lence, 2005].

Al igual que las computadoras evolucionan, el desarrollo de software tiene la necesidad de proveer acceso a la mayor cantidad de recursos disponibles. Anteriormente, el software se orientó hacia la programación en serie, es decir, para resolver un problema, se genera un algoritmo y se implementa en un flujo de instrucciones en serie (en el momento que una instrucción termina su ejecución, empieza a efectuar la siguiente y así hasta terminar con las líneas de código) [Grama *et al*., 2003].

La computación paralela emplea elementos de procesamiento múltiple simultáneos, se basa en el principio de que los problemas grandes se pueden dividir en partes más pequeñas que pueden resolverse de forma concurrente [Lence, 2005].

A diferencia de la programación en serie, la ventaja computación paralela es que la potencia global del sistema no se fundamenta en la potencia individual de cada procesador sino del conjunto. De este modo, este tipo de sistemas se pueden configurar a partir de elementos modestos, obteniendo un alto rendimiento a bajo costo.

La programación paralela de una arquitectura multiprocesador se realiza mediante lenguajes que permitan expresar el paralelismo e intercambiar información entre los procesadores, como los lenguajes de paso de mensajes. Sin embargo, existe un obstáculo fundamental para el avance de la computación paralela: su programación, ya que los compiladores que detectan

paralelismo automáticamente presentan todavía limites a su aplicabilidad [Santiago, 2006].

En el Centro de Investigación en Ingeniería y Ciencias Aplicadas, se ha estado trabajando con algoritmos paralelos para el desarrollo de aplicaciones concurrentes, con el objetivo de obtener mejor tiempos de respuesta en la aplicación del paso de mensajes y la optimización de recursos disponibles. Se ha estado utilizando herramientas (como AUTOMAP) para poder crear y asignar tipos de datos definidos por el usuario, pero se requiere de una mejor organización y manipulación de estos tipos de datos, ya que estas herramientas no tiene flexibilidad para poder asignar memoria, ni la creación de nuevos tipos de datos.

Ante la complejidad de los algoritmos paralelos con paso de mensajes, los cuales trabajan con gran cantidad de datos, se hace necesario aprovechar la capacidad de cómputo que ofrecen varias máquinas en paralelo; para dicho concepto se deben estudiar los modelos de programación paralela y las aplicaciones que hasta ahora se han creado para sistemas distribuidos.

1.2 Planteamiento del problema

La necesidad de trabajar con múltiples datos de tipo complejo para tener una mejor organización de los datos y la no existencia de información sobre cómo hacer este tipo de comunicación, abre un área de oportunidad en la búsqueda de una transferencia óptima de datos complejos en sistemas distribuidos.

El desarrollo de esta investigación es experimental, pues la manipulación de variables en las pruebas, en un conjunto de varias estaciones de trabajo (nodos), permite la obtención de resultados necesarios para llevar a cabo el objetivo propuesto: Implementar una MPI (del inglés: Message Passing Interface) para estructuras de datos complejas, con el propósito de aplicarlas en el desarrollo de algoritmos que optimicen recursos, los cuales permitan evaluar la eficiencia de la transferencia de datos en sistemas distribuidos.

1.3 Panorama general del proyecto

En este trabajo se determinará el rendimiento de un algoritmo de estructuras de datos complejas en una plataforma de programación distribuida basada en MPI.

Es precisamente este enfoque, el de ambientes distribuidos, en el que se plantea la investigación, implementando MPI dada la necesidad de compartir información entre los nodos de un ambiente distribuido, con el propósito de conocer cómo se comportan los algoritmos con estructuras de datos complejas en un ambiente donde sean varios procesadores los encargados de llevarlos a cabo, en comparación con su ejecución computacional centralizada, es decir utilizando un solo procesador.

1.4 Objetivo general

Desarrollar un procedimiento que organice la información en estructuras de datos complejas y maneje el paso de mensajes en un ambiente *Cluster*, verificando que la información enviada entre los nodos del Cluster no se pierda y aportar una mayor comprensión en el entendimiento de programas de cómputo desarrollados para ambientes *Cluster*.

1.5 Objetivos específicos

a) Diseñar e implementar un algoritmo con paso de mensajes que organice la información en estructuras de datos complejas, asignando memoria estática a las mismas.

b) Diseñar e implementar un algoritmo con paso de mensajes que asigne memoria de manera dinámica a las estructuras de datos complejas.

1.6 Justificación

Esta investigación tiene como principal funcionalidad mejorar la organización de la información, manejando estructuras de datos que permitan agrupar datos que no son de igual tipo, que se crean a partir de datos primitivos y se puedan representar como un único tipo de datos en un solo parámetro. Dicha información será enviada a los nodos del *Cluster*, en ellos se modificará la información y se enviará de regreso al nodo principal para verificar que no se pierda la consistencia de los datos.

La investigación planteada permitirá conocer las características más resaltantes del rendimiento de los algoritmos con paso de mensajes en una plataforma distribuida (*Cluster*) y contribuir al conocimiento de los problemas que afectan al desempeño de algoritmos e implementaciones similares, así como los puntos positivos de este enfoque.

Con los resultados de esta investigación se podrán aportar también recomendaciones, sugerencias o nuevos temas de estudio acerca del desarrollo de este u otros procedimientos de paso de mensajes para estructuras de datos complejas, posiblemente optimizados para plataformas de programación distribuida.

1.7 Alcances y limitaciones

Las características de la investigación se hacen claras partiendo de dos vertientes:

1. Software, determinado por el desarrollo de algoritmos que manejen el Paso de Mensajes (MPI) con estructuras de datos complejas (estructuras anidadas), aplicadas a un ambiente *Cluster*.
2. Hardware, para el desarrollo de esta investigación la disponibilidad de los recursos necesarios para llevar a cabo el proyecto se muestra en la tabla 1.1.

<table>
<tr><th>ELEMENTO</th><th>HARDWARE</th><th>SOFTWARE</th></tr>
<tr><td>Comunicaciones</td><td>Switch Cisco C2960
Cableado interno nivel 5</td><td></td></tr>
<tr><td>Nodo Maestro</td><td>Pentium 4, 2793 MHz
512 MB Memoria
80 GB Disco Duro
2 Tarjetas 10/100 Mb/s</td><td rowspan="2">S.O. Red Hat Enterprise Linux 4
Compilador gcc version 3.4.3
OpenMPI 1.2.8
MPICH2-1.0.8
Ganglia 3.0.6
NIS ypserv-2.13-5
NFS nfs-utils-1.0.6-46
Open VPN</td></tr>
<tr><td>Nodos de Procesamiento 01 - 18</td><td>Intel® Celeron® Dual Core, 2000MHz.
RAM 2GB, 160 Disco Duro
Tarjeta 10/100 Mb/s</td></tr>
</table>

Tabla 1. 1 Recursos del *Cluster* CIICAp.

1.8 Organización de la Tesina

En este capítulo se presentó el alcance de la investigación: el problema y el objetivo de la implementación del paso de mensajes en estructuras de datos complejas en un ambiente distribuido. La investigación fue motivada para aportar información para un mejor entendimiento en el envío y recepción de estructuras de datos complejas, ejecutadas en un ambiente distribuido.

El resto de la tesina está organizada de la siguiente manera:

Capítulo 2: Describe de manera puntual los temas que engloban la aplicación de una estructura de datos compleja con el paso de mensajes en un ambiente distribuido.

Capítulo 3: Se da una explicación introductoria del algoritmo propuesto abordando el problema del paso de mensajes con estructuras de datos complejas. Se explica ampliamente los pasos necesarios para poder crear nuevos tipos de datos en el uso de paso de mensajes en MPI. Además de la metodología usada en la investigación y las etapas importantes en la aplicación del paso de mensajes en trabajos futuros.

Capítulo 4: Se proporciona una explicación detallada de los pasos que se realizaron con el objetivo de obtener una buena organización de los datos y la optimización de recursos, se muestran las pruebas aplicadas al procedimiento propuesto. También se mencionan los problemas que se deben tomar en cuenta como base en trabajos futuros que necesiten implementar el paso de mensajes.

Capitulo 5: Se muestran los resultados experimentales obtenidos en las pruebas realizadas al algoritmo propuesto, además de presentar los resultados obtenidos en cuanto a eficiencia y eficacia.

Capítulo 6: Se dan las conclusiones finales del trabajo de investigación, así como los trabajos futuros.

CAPÍTULO II ESTADO DEL ARTE

2.1. Introducción

Un *Cluster* es un grupo de equipos independientes que ejecutan una serie de aplicaciones de forma conjunta y aparecen ante clientes y aplicaciones como un solo sistema. Los *Cluster* permiten aumentar la escalabilidad, disponibilidad y fiabilidad de múltiples niveles de red, pueden presentarse como una solución de especial interés, ya que se pueden tener equipos actualizados por un precio bastante más económico y con unas capacidades de computación que en muchos casos pueden llegar a superar a hardware de última generación.

Hoy en día los *Cluster* junto con las bibliotecas de paso de mensajes y los sistemas de memoria compartida distribuida han puesto a la computación paralela al alcance de todos. La programación con paso de mensajes actualmente es portable pero complicada, excepto para simples patrones de acceso a datos como son las estructuras de datos simples, que están compuestas por datos de tipo primitivo. Además, la programación con paso de mensajes se vuelve realmente compleja cuando la cantidad de datos a procesar excede la capacidad de la memoria. Un sistema de memoria compartida distribuida simplifica la programación, es por eso que esta investigación se centra en estudiar y desarrollar algoritmos para mandar datos en estructuras complejas en un ambiente distribuido, las cuales permitan hacer del algoritmo un programa de cómputo simple y fácil de entender y programar, donde la información de los datos se encuentre organizada de la mejor manera de tal forma que simplifique la programación del algoritmo.

Actualmente, la sociedad está inmersa en la tecnología vista como un conjunto de teorías y técnicas que permiten el aprovechamiento práctico del conocimiento científico [Cegarra, 2004]. Esta realidad moderna del mundo tecnológico no es producto de la casualidad, sino un reflejo de la búsqueda del hombre por mejorar su calidad de vida. Es a partir de este propósito que la tecnología se ha desarrollado y se sigue desarrollando de la misma manera.

En el ramo de las ciencias de la computación, dicha búsqueda viene evidenciándose en el aumento de capacidad y organización de almacenamiento de datos y el aumento en potencial de procesamiento de los artefactos a nuestra disposición, con la respectiva reducción de tiempo empleado en múltiples tareas. Además, los ambientes distribuidos definidos como un gran número de computadoras organizadas en conjunto incrustados en una infraestructura de telecomunicaciones distribuida vienen surgiendo con ciertas posibilidades y aplicaciones que los convierten en un enfoque interesante a la hora de estudiar ciertos problemas de procesamiento de datos [Rosano, 1998].

El cómputo paralelo es utilizado para mejorar el tiempo de ejecución de muchas aplicaciones [Santiago, 2006]. En este capítulo se presenta una breve reseña del cómputo paralelo y cómo ha evolucionado hacia los *Clusters* como una excelente alternativa para ejecutar aplicaciones paralelas.

2.2. Cómputo paralelo

El cómputo paralelo es una técnica de programación en la que muchas instrucciones se ejecutan simultáneamente. Se basa en el principio de que los grandes problemas se pueden dividir en partes pequeñas que pueden procesarse de forma concurrente "paralelo" [Lence, 2005].

Uno de los usos básicos de la computación paralela es ejecutar una aplicación existente en una computadora diferente. La computadora en donde la aplicación se ejecuta normalmente podría estar ocupada debido a una sobrecarga de trabajo. El trabajo en cuestión se podría estar ejecutando en una computadora inactiva en algún lugar.

Hay al menos dos requisitos previos para el escenario mencionado anteriormente. En primer lugar, la aplicación debe ser ejecutable de manera remota y sin gastos indebidos. En segundo lugar, la computadora remota no tiene que tener un hardware, software o algún requisito impuesto por las aplicaciones.

El cómputo paralelo ha sido ampliamente utilizado para mejorar el desempeño de aplicaciones que demandan gran cantidad de cómputo en ambientes distribuidos como son los *Clusters*. Algunas de estas aplicaciones en general se caracterizan por que realizan gran cantidad de operaciones de cálculo en la solución de problemas de ciencias e ingeniería [Geist *et al*., 2000].

2.3. Plataformas de cómputo paralelo

Los elementos de las plataformas de cómputo paralelo que son críticas para el rendimiento orientado y portable en un ambiente distribuido, están basadas en la organización lógica y física. La organización lógica se refiere a la vista que el programador tiene de la plataforma y la organización física, está relacionada al hardware que contiene la plataforma. En la perspectiva de un programador, estos dos componentes críticos de la computación paralela son formas de expresar tareas paralelas y mecanismos para especificar la iteración entre las mismas. La organización lógica también se conoce como la estructura de control y la física como el modelo de comunicación [Grama *et al*., 2003].

2.3.1 Estructura de control en plataformas paralelas

Las tareas paralelas pueden ser vistas en varios niveles de granularidad. En un extremo, cada tarea como un programa en un juego de programas. Otro extremo como un conjunto de instrucciones individuales dentro de un programa. Entre estos dos extremos, existe una serie de modelos que especifican la estructura de control de programas y la arquitectura correspondiente soportada para ellos.

Las unidades centrales de procesamiento en un ambiente distribuido funcionan bajo el control centralizado de una unidad de control o el control por separado. En arquitecturas referidas a un único flujo de instrucciones (del inglés: *single instruction stream, multiple data stream;* SIMD), solo una unidad de control envía instrucciones a cada unidad de procesamiento. La Figura 2.1 (a) ilustra una arquitectura típica SIMD. En una computadora paralela, la misma instrucción es ejecutada síncronamente por todas las unidades de procesamiento.

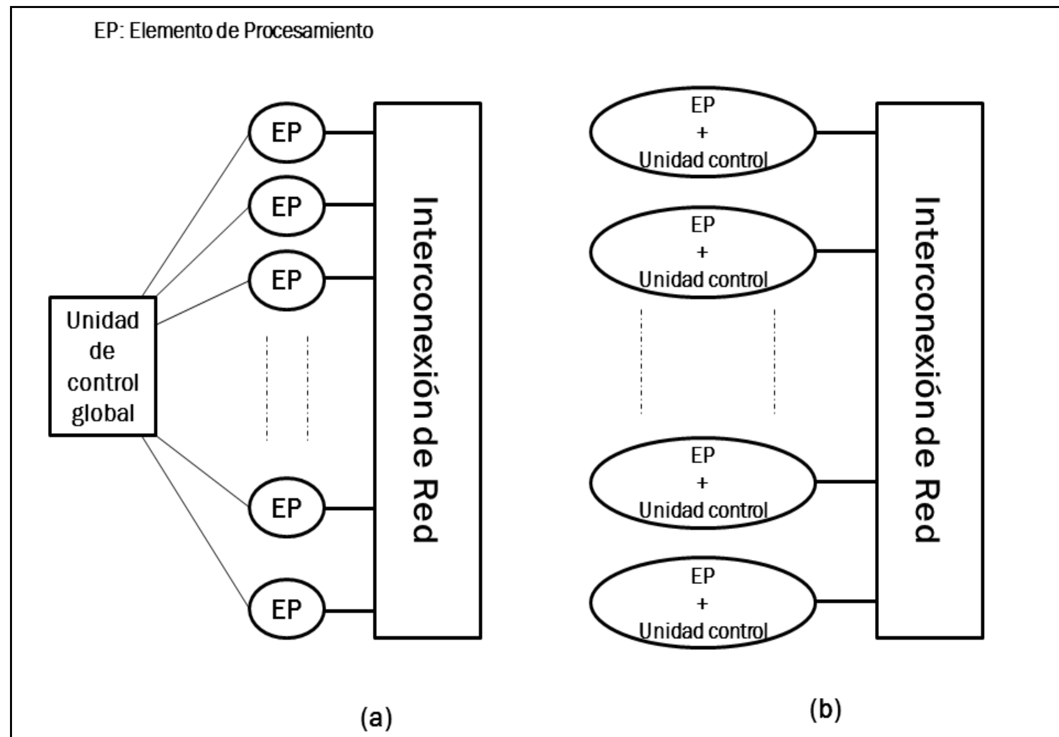

Figura 2. 1 Arquitectura típica SIMD (a) y una arquitectura típica MIMD (b) [Grama *et al*., 2003]

En contraste con la arquitectura SIMD, computadoras en la que cada de procesamiento es capaz de ejecutar un programa diferente independiente de los otros procesadores son llamados de flujo de instrucciones múltiples (del inglés: *multiple instruction stream, multiple data stream*; MIMD) modelos. La Figura 2.1 (b) muestra una arquitectura típica MIMD. Una simple variación de este modelo, llamado modelo de programa único de múltiples datos (del inglés: *single program multiple data*; SPMD), se basa en varias instancias del mismo programa de ejecución pero con datos diferentes. El modelo SPMD tiene la misma expresividad que el modelo MIMD, ya que cada uno de los bloques de instrucciones se puede insertar en un gran conjunto de If-Else con las condiciones específicas para identificar las tareas. El modelo SPMD es ampliamente utilizado por muchas plataformas paralelas y requiere soporte de arquitectura mínima [Grama *et al*., 2003].

2.3.2 Modelo SPMD: Single Program Multiple Data

La ejecución de un programa de una aplicación en un nodo puede hacerse en forma fácil al dar el nombre del programa en la línea de comandos del sistema operativo. Si se ejecuta la misma aplicación en varios nodos y utilizamos diferentes programas, la implementación, el arranque y la ejecución de cada programa se hace más complicado conforme el número de nodos crece. Sin embargo, si se ejecuta una copia del mismo programa en cada nodo, la tarea se simplifica.

Por otro lado, si se desea construir un programa portable y escalable es recomendable escribir un solo programa, el cual decide en ejecución lo que realmente va a hacer, en lugar de estructurar nuestra aplicación como programas separados, se escribe una sola pieza del código fuente el cual en ejecución decide que acción tomar dependiendo del identificador del procesador en donde se está ejecutando el programa.

EL uso de *Clusters* bajo cómputo paralelo es generalmente bajo el modelo SPMD, el cual es el más comúnmente utilizado para desarrollar aplicaciones paralelas en *Cluster.* En este modelo el mismo código del programa de la aplicación es dividido y distribuido en los procesadores donde va ejecutar el programa. Así, cada procesador ejecuta básicamente la misma pieza de código pero sobre una parte diferente de los datos [Santiago, 2006]. La Figura 2.2 muestra una representación esquemática de este paradigma.

Figura 2. 2 Estructura básica de un programa SPMD

2.4. Programación con Paso de Mensajes

En el punto de vista lógico de una computadora de plataforma de paso de mensajes, se compone de nodos de procesamiento, cada uno con su propio espacio de dirección exclusiva. Cada uno de estos nodos de procesamiento puede ser procesadores de uno o varios núcleos. Las instancias de este punto de vista vienen evidenciándose en la creación de plataformas llamadas *Cluster*, en tales plataformas, las iteraciones entre los procesadores que se ejecutan en diferentes nodos debe llevarse a cabo por medio de mensajes, de ahí el nombre de "Paso de Mensajes". Este intercambio de mensajes se utiliza para transferir datos, trabajo y para sincronizar las acciones entre los procesos. En su forma general, la ejecución del paradigma paso de mensajes se basa en un programa diferente en cada uno de los nodos de procesamiento.

Dado que las iteraciones se realizan mediante el envío y recepción de mensajes, las operaciones básicas en este paradigma de programación son enviar y recibir, las llamadas correspondientes pueden variar dependiendo a la interfaz de programación que se esté usando, pero la semántica es prácticamente la misma. Además, en las operaciones de envío y recepción se debe especificar la dirección de destino, tiene que haber un mecanismo para asignar un identificador (ID) único a cada uno de los procesos de ejecución en un programa paralelo. Esta identificación se hace normalmente accesible para el programa que utiliza una función como -***whoami,*** que devuelve a un proceso que solicita a su identificación. Hay una función que es necesaria para complementar el conjunto básico de operaciones de paso de mensajes, -***numprocs*** especifica el número de procesos que participan en el programa. Existen diferentes APIs de paso de mensajes, como MPI, la máquina virtual paralela (PVM) y fortran de alto rendimiento (HPF), que soportan estas operaciones básicas y una variedad de funciones a nivel superior, bajo los nombres de funciones diferentes [Grama *et al*., 2003].

2.4.2 Aspectos de diseño

En el modelo de programación con paso de mensajes, para transmitir un mensaje la siguiente información debe ser especificada:

- ¿*Qué* procesador está transmitiendo el mensaje?
- ¿Dónde está *localizado* el dato del procesador transmisor?
- ¿Qué *tipo* de dato se está transmitiendo y *cuántos* datos están ahí?
- ¿*Cuál(es)* procesador (es) está(n) recibiendo el mensaje?
- ¿En *dónde* sería dejado el dato sobre el procesador receptor?
- ¿*Cuántos* datos está preparado para aceptar el procesador receptor?

La utilización de estas bibliotecas no es una tarea fácil y tiende a complicarse aún más por la presencia de uno o varios de los siguientes elementos: la utilización de estructuras de datos complejas, un número grande de nodos de procesamiento o cuando el tamaño del problema crece.

Al transmitir dos mensajes entre dos procesos, el proceso transmisor y el proceso receptor cooperarán en proporcionar la información para que pueda ser enviado y recibido el mensaje de manera confiable. En el código 2.1 se tienen dos procesos cada uno con su propio espacio de almacenamiento de datos *x* y *y.* El proceso 0 envía un mensaje al proceso 1 de una `LONGITUD` dada a través de la red de comunicación del sistema. Ambos procesos identifican al proceso con el que intercambian los datos (proceso origen y destino), determina el tipo de datos del mensaje y utilizan una etiqueta que permite identificar el mensaje que se transmite.

Proceso 0

```
char x[LONGITUD]; send(destino, &x, LONGITUD, tipo
dato, etiqueta);
```

Proceso 1

```
char y[LONGITUD];
recv(origen, &y, LONGITUD, tipo dato, etiqueta);
```

Código 2. 1 Ejemplo de comunicación con paso de mensajes

La operación de transmisión puede ser síncrona o asíncrona. La transmisión síncrona (envío bloqueante) termina solamente cuando el mensaje correspondiente está siendo recibido por un proceso receptor. El emisor y receptor se sincronizan para intercambiar cada mensaje. La transmisión asíncrona (envío no bloqueante) termina tan pronto como el mensaje correspondiente se haya entregado al sistema de comunicación. El emisor sigue ejecutándose sin esperar a que la comunicación se complete.

Actualmente, las interfaces de paso de mensajes tales como PVM y MPI, son las más utilizadas en la programación de aplicaciones paralelas en arquitecturas de memoria distribuida. Estas interfaces hacen posible escribir programas paralelos que son portables a una amplia variedad de plataformas sin sacrificar el rendimiento [Santiago, 2006].

2.4.3 PVM

PVM (Máquina Virtual Paralela) es un conjunto integrado de herramientas de software que emula de manera general y flexible, a un conjunto heterogéneo de computadoras interconectadas como una sola computadora virtual. El objetivo general del sistema PVM, es permitir que un conjunto de equipos se utilicen en forma cooperativa y concurrente (paralela).

El sistema PVM es compuesto por dos partes. La primera parte es un demonio, llamado *pvmd3* y algunas veces abreviado *pvmd,* que reside en todos los equipos que componen la computadora virtual. La segunda parte del sistema es una librería con las rutinas de la interfaz PVM. La librería contiene un completo repertorio de funciones que son necesarias la comunicación entre las tareas de una aplicación paralela. Esta biblioteca contiene rutinas para el paso de mensajes, generación de procesos, coordinación de tareas y modificar la computadora virtual.

El modelo de computación PVM está basado en que una aplicación se compone de muchas tareas (una tarea es la ejecución de una instancia del algoritmo). Cada tarea es responsable de una parte de los datos que la aplicación le asigne. En este modelo todas las tareas son las mismas, pero

cada una recibe y resuelve una parte de los datos. En la Figura 2.3, se muestra la vista de la arquitectura del sistema PVM, resaltando que las computadoras son de carácter heterogéneo [Geist *et al*, 2000].

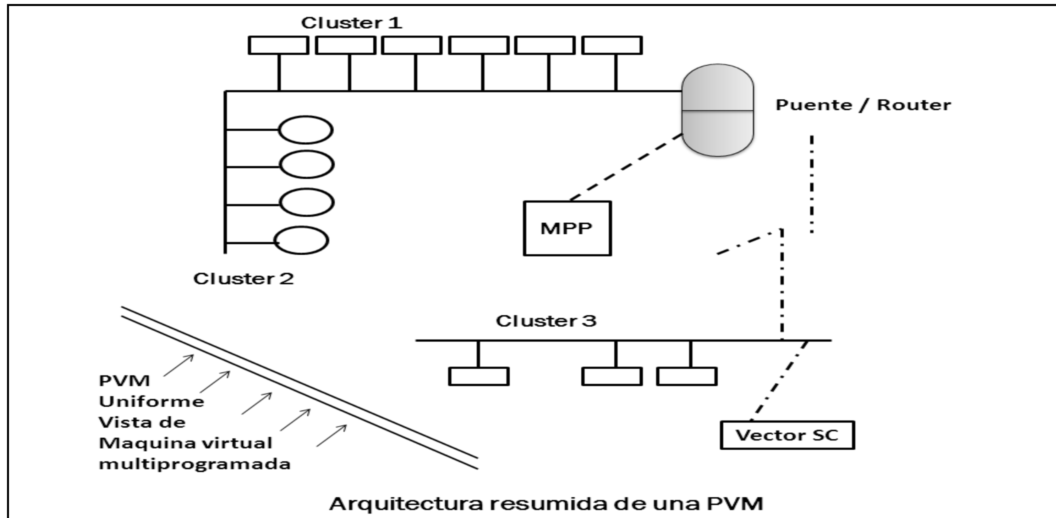

Figura 2. 3 Arquitectura de un sistema PVM [Geist *et al*, 2000]

La mayor ventaja de PVM es su portabilidad en ambientes heterogéneos. Un programa PVM puede ejecutarse sobre cualquier plataforma que contenga computadoras con múltiples arquitecturas entre ellas. La principal desventaja de PVM es que su rendimiento no están bueno como otros sistemas de paso de mensajes, por mencionar alguno MPI. Esto es debido principalmente porque PVM sacrifica rendimiento por flexibilidad al permitir heterogeneidad en la arquitectura de las computadoras [Martínez, 2006].

2.4.4 MPI

MPI es un estándar que define la sintaxis y la semántica de las funciones contenidas en una biblioteca de paso de mensajes diseñada para ser usada en programas que exploten la existencia de múltiples procesadores. MPI es un protocolo de comunicación entre computadoras. Es el estándar para la comunicación entre los nodos que ejecutan un programa en un sistema de memoria distribuida y consiste enteramente en las declaraciones los lenguajes de programación C y Fortran convencionales y las directivas anteriores de los procesadores.

MPI no es un nuevo lenguaje de programación, es simplemente una colección de definiciones y funciones que pueden estar en programas de lenguajes C y Fortran. El objetivo principal de MPI es lograr la portabilidad a través de diferentes máquinas, tratando de obtener un lenguaje de programación que permita ejecutar de manera transparente, aplicaciones sobre sistemas heterogéneos [Pacheco, 1997].

Tipos de comunicaciones

Los programas de paso de mensajes se pueden escribir usando los modos de comunicación asíncrona o síncrona.

No bloqueante: El proceso que envía, no espera a que el mensaje sea recibido y continúa su ejecución, siendo posible que vuelva a generar un nuevo mensaje y a enviarlo antes que se haya recibido el anterior Figura 2.4 (b).

Bloqueante: El proceso que envía el mensaje espera a que un proceso la reciba para continuar con la ejecución Figura 2.4 (a) y (c).

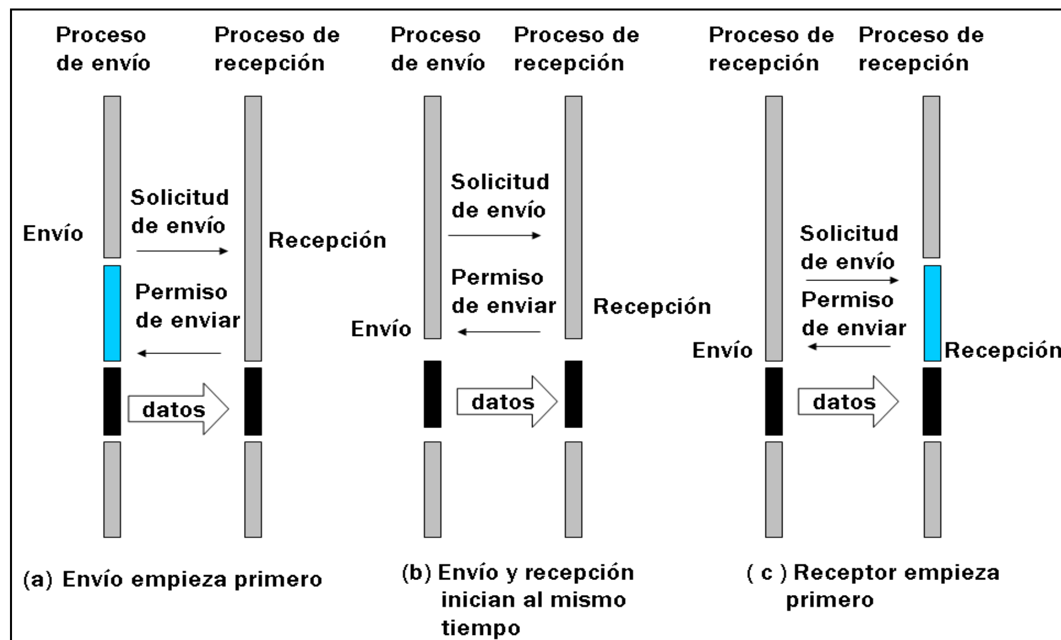

Figura 2. 4 Vista de una comunicación asíncrona [Grama *et al.*, 2003]

Comunicación bloqueante y no bloqueante

La función que se utiliza para la recepción de mensajes bloqueantes es: `MPI_Recv`. Esto significa que cuando un proceso llama a la rutina `MPI_Recv`, si el mensaje no está disponible, el proceso permanecerá inactivo hasta que esté disponible. Se tiene que tener en cuenta que esto no es lo mismo que la comunicación síncrona. En la comunicación síncrona, los dos procesos que se comunican: proceso 0 no empieza a enviar el mensaje hasta que el mensaje empiece a ser recibido por el proceso 1.

La mayoría de los sistemas proveen una alternativa, una operación de recepción no bloqueante. En MPI, la rutina es: `MPI_Irecv`, en donde la I se refiere a inmediato, que significa que el proceso regresa inmediatamente después de la llamada a la rutina. Esta rutina tiene un parámetro más que la de `MPI_Recv`: *request*.

El uso de las comunicaciones no bloqueantes pueden ser usadas para proporcionar mejoras en el desempeño de los programas de paso de mensajes. Si un nodo de un sistema paralelo tiene la capacidad de calcular y comunicar al mismo tiempo, la sobrecarga debida a la comunicación puede disminuirse [Pacheco, 1997].

Tipos de rutinas

Las rutinas en MPI se dividen en cuatro clases:

- Administrar comunicaciones: Son llamadas a rutinas que tienen como función iniciar, gestionar o finalizar una sesión de comunicación.
- Transferir datos entre dos procesos: Se encargan de definir las reglas de comunicación para dos procesos.
- Transferir datos entre varios procesos: Operaciones grupales, proveen operaciones de comunicación entre grupos de procesos.
- Crear tipos de datos definidos por el usuario: MPI provee flexibilidad en la construcción de nuevos tipos de datos que el usuario puede definir, como pueden ser las estructuras de datos, esta investigación está

basada en la creación de estructuras de datos complejas, es decir, estructuras que sus tipos de datos sean otras estructuras. [Karniadakis, 2007].

Tipos de datos MPI

En la Tabla 2.1 se muestra la correspondencia entre los tipos de datos MPI y los que provee el lenguaje de programación C.

Tipo de dato MPI	Tipo de dato C
MPI_CHAR	signed char
MPI_SHORT	signed short int
MPI_INT	signed int
MPI_LONG	signed long int
MPI_UNSIGNED_CHAR	unsigned char
MPI_UNSIGNED_SHORT	unsigned short int
MPI_UNSIGNED	unsigned int
MPI_UNSIGNED_LONG	unsigned long int
MPI_FLOAT	float
MPI_DOUBLE	double
MPI_LONG_DOUBLE	long double
MPI_BYTE	
MPI_PACKED	

Tabla 2. 1 Correspondencia entre tipos de datos soportados por MPI y el lenguaje C

Nótese que para todos los tipos de datos que provee el lenguaje C, existen un equivalente tipo de dato MPI. Sin embargo, MPI tiene dos tipos de datos adicionales que no son parte del lenguaje C. Estos son MPI_BYTE y MPI_PACKED.

MPI_BYTE corresponde a un byte (8 bits) y MPI_PACKED corresponde a una colección de elementos de datos que han sido creados en un paquete de datos no contiguos. Note que la longitud del mensaje en MPI_SEND, así como en cualquier otra rutina de MPI, se expresa en términos del número de entradas que se envían y no en el número de bytes [Grama *et al*., 2003].

2.4.5 Ejemplo con MPI

El primer programa de MPI que se muestra en la siguiente código 2.2, imprime el mensaje “Hola mundo” con una variante, este programa hace uso de múltiples procesos y cada uno de estos procesos envía un saludo a otro proceso. En la mayoría de los sistemas paralelos, el proceso involucrado en la ejecución de un programa paralelo se identifica por una secuencia de números enteros no negativos. Si hay P procesos ejecutando un programa, ellos tendrán rangos de 0, 1,..., $P - 1$.

Entonces, una posibilidad es que para cada proceso distinto de 0 envié un mensaje al proceso 0, para conocer el mensaje recibido en dicho proceso, los mensajes tendrán que ser impresos. En el código 2.2 se muestra un programa MPI que hace esto [Pacheco, 1997].

```
#include<stdio.h>
#include<string.h>
#include”mpi.h”
main(int argc, char* argv[]){
int miRango;            /* Identificador de procesos */
int procesos;           /* Numero de procesos */
int fuente;             /* Rango de emisor */
int destino;            /* Rango de receptor */
int etiqueta = 0;       /* Etiqueta del mensaje */
char mensaje[100];      /* Almacenamiento de mensaje */
MPI_Status estado;      /* Estado del receptor */
MPI_Init(&argc, &argv); /* Inicia MPI */
/* Conocer identificador de procesos */
MPI_Comm_rank(MPI_COMM_WORLD, &miRango);
/* Conocer numero de procesos */
MPI_Comm_zise(MPI_COMM_WORLD, &procesos);
```

```
if(miRango == 0){       /* ¿proceso maestro? */

/* Crea y envía mensaje */

sprintf(mensaje, "Hola desde proceso: %d!", miRango);

destino = 0;

MPI_Send(mensaje, strlen(mensaje) + 1, MPI_CHAR,
destino, etiqueta, MPI_COMM_WORLD);

} /* Fin del maestro */

else {   /* Recibe el mensaje */

MPI_Recv(mensaje, 100, MPI_CHAR, fuente, etiqueta,
MPI_COMM_WORLD, &etiqueta);

Printf("%s \n", mensaje);

}   /* Fin del else */

MPI_Finalize();         /* Termina comunicación MPI */

}   /* Fin del main */
```

Código 2. 2 Programa "Hola mundo" en MPI

Compilación y ejecución del programa

Los detalles de compilación y ejecución del programa dependen al sistema que se esté usando. La compilación puede ser como se muestra en el código 2.3.

```
% mpicc –o holamundo holamundo.c
```

Código 2. 3 Compilación del programa "hola mundo.c" en MPI

El segmento de código 2.3 genera un archivo ejecutable con el nombre holamundo del programa holamundo.c, esto si en la compilación no se encontraron errores. Para ejecutar el programa se ingresa la siguiente línea de código 2.4, en este ejemplo el programa es ejecutado con 4 procesadores.

```
% mpirun –np 4 holamundo
```

Código 2. 4 Comando para ejecutar un programa MPI

Sin embargo, también puede haber una secuencia de comandos especiales dependiendo al sistema en el que el programa se esté corriendo. Cuando el programa se compila y ejecuta con cuatro procesos, la salida debe ser la mostrada en la Figura 2.5:

Hola desde proceso 1!
Hola desde proceso 2!
Hola desde proceso 3!

Figura 2. 5 Salida del programa "hola mundo" en MPI

2.5. Cluster

Un *Cluster* es un tipo de sistema de procesamiento paralelo o distribuido, que consiste en una colección de computadoras interconectadas independientemente trabajando juntos como una sola, recurso de computación integrado [Maozheny Baker, 2005].

El termino *Cluster* es generalmente usado para describir una amplia gama de sistemas de procesamiento distribuido. Gregory Pfister en su libro en 1997 *"En busca de Clusters"*, el propone la siguiente definición:
Un Cluster es un tipo de sistema de datos paralelo o distribuido:

- *Consiste en una colección de computadoras interconectadas entre si.*
- *Se utiliza como un solo recurso informático unificado.*

2.5.1 Arquitectura de un Cluster

Utilizando la definición anterior de que todos los *Cluster* deben actuar como "*un único recurso informático unificado".* Un ejemplo de un recurso informático unificado es un solo equipo, como se muestra en la Figura 2.6.

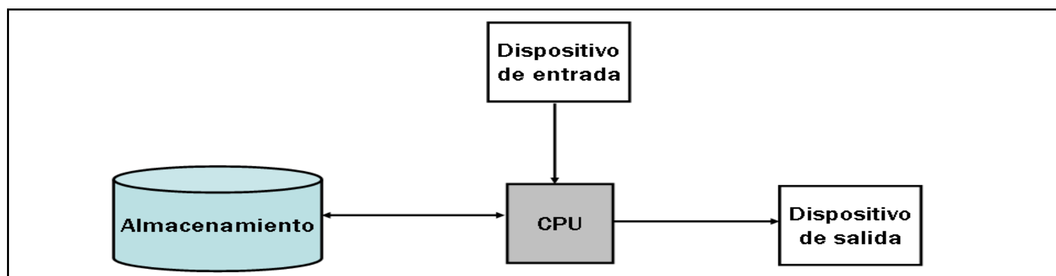

Figura 2. 6 Arquitectura simplificada de un solo equipo

Si remplazamos el CPU o nodo en la Figura 2.6 con *"una colección de computadoras interconectadas",* el diagrama se podría dibujar como se muestra en la Figura 2.7.

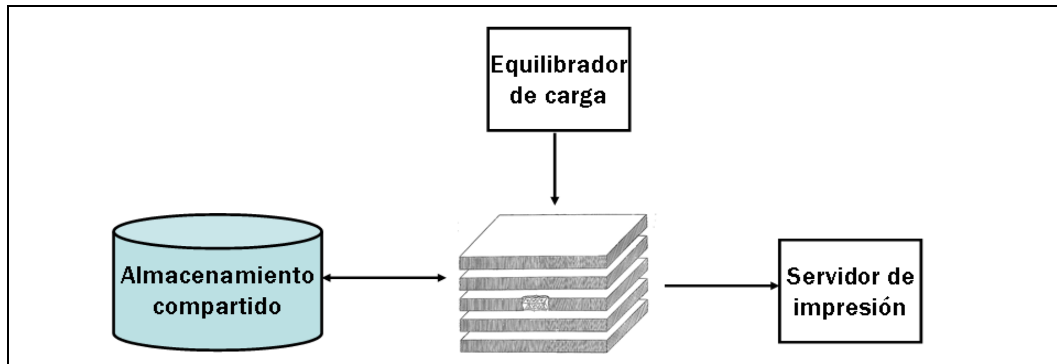

Figura 2. 7 Arquitectura simplificada de un *Cluster*

El equilibrador de carga remplaza los dispositivos de entrada, el almacenamiento compartido remplaza el disco de almacenamiento y el servidor de impresión es un ejemplo de un dispositivo de salida compartido [Kopper, 2005].

Un nodo puede ser una computadora o un sistema multiprocesador (PC, Workstation, SMP) con memoria, facilidades de entrada-salida y un sistema operativo. Los nodos pueden estar en un gabinete o físicamente separados y conectados vía LAN. Los nodos Interconectados (basados en una LAN), aparecen como un solo sistema de usuarios y aplicaciones. Este sistema puede proporcionar un costo bajo para obtener las características y beneficios (servicios rápidos y confiables), que históricamente han sido encontrados sólo en los más caros sistemas de memoria compartida. La arquitectura típica de un *Cluster* se muestra en la Figura 2.8.

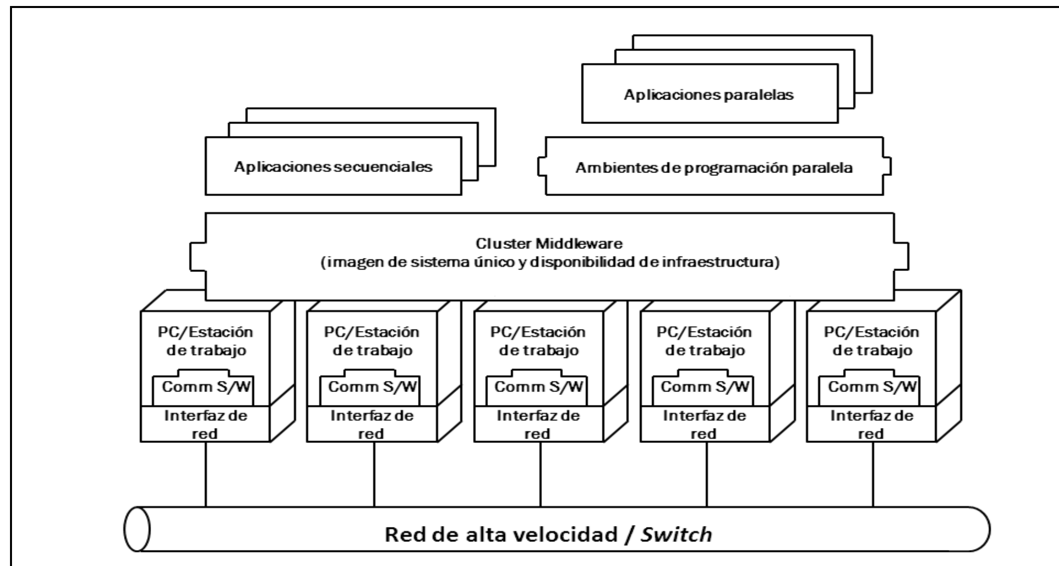

Figura 2. 8 Componentes de Hardware y Software de un *Cluster* [Maozhen y Baker, 2005]

2.5.2 Componentes del Cluster

Los siguientes son algunos de los componentes importantes que conforman un *Cluster*:

- Equipos de alto rendimiento (PC, Workstation o SPM).
- Sistema operativo (Kernel).
- Redes de alto rendimiento (Gigabit Ethernet).
- Tarjetas de interfaz de red (NIC).
- Protocolos de servicios y comunicaciones rápidos.
- *Cluster* middleware.
- Ambientes y herramientas de programación paralela (como compiladores, PVM y MPI).
- Aplicaciones (Paralelas).

En un *Cluster* el hardware de interfaz de red actúa como un procesador de comunicaciones y es responsable de transmitir y recibir paquetes de datos entre los nodos del *Cluster* a través de una red o *Switch*. Mientras que el middleware de *Cluster* es responsable de mostrar una imagen de un sistema unificado y la disponibilidad de una colección de computadoras independientes pero interconectadas [Maozheny Baker, 2005].

2.5.3 Cluster middleware

El middleware es un software de conectividad que ofrece un conjunto de servicios que hacen posible el funcionamiento de aplicaciones distribuidas sobre plataformas heterogéneas. Funciona como una capa de abstracción de software distribuida, que se sitúa entre las capas de aplicaciones y las capas inferiores (sistema operativo y red). El middleware abstrae de la complejidad y heterogeneidad de las redes de comunicaciones subyacentes, así como de los sistemas operativos y lenguajes de programación, proporcionando una API para la fácil programación y manejo de aplicaciones distribuidas. Dependiendo del problema a resolver y de las funciones necesarias, serán útiles diferentes tipo de servicios de middleware. Por lo general el middleware del lado cliente está implementado por el Sistema Operativo subyacente, el cual posee las bibliotecas que implementan todas las funcionalidades para la comunicación a través de la red [Prabhu, 2008].

El middleware tiene como principal objetivo generar la sensación de que el usuario está utilizando un solo ordenador muy potente y no un conjunto de computadoras conectadas entre si. También el Cluster middleware detecta automáticamente nuevos equipos conectados al Cluster, para poder usarlos posteriormente. Las ventajas que si tienen con el Cluster middleware son el precio y el rendimiento. El precio por que existen muchos actualmente libres. El rendimiento es muy alto ya que distribuye los trabajos de manera que el proceso se ejecute más rápido y el sistema no sufra sobrecargos. Teniendo en cuenta que el middleware proporciona facilidad de acceso al Cluster y su excelente relación precio/rendimiento, estas plataformas se han convertido en una opción especialmente atractiva.

CAPÍTULO III ANÁLISIS DEL PROBLEMA Y DISEÑO DE LA SOLUCIÓN

3.1 Introducción

Cuando se desea resolver un problema es necesario seleccionar o diseñar un algoritmo. Lógicamente, se puede disponer de distintas soluciones y, en principio y a pesar de que unos sean más rápidos que otros, o que unos consuman más recursos que otros, puede parecer irrelevante esta selección. Además se puede pensar que el incremento de potencia en los recursos de las computadoras actuales, permite disponer de soluciones en un tiempo relativamente pequeño y por tanto, incluso los algoritmos más lentos se ejecutarán en poco tiempo.

Sin embargo, esta creencia es totalmente falsa. Considere el siguiente ejemplo. Se tiene el problema de asignar 50 trabajadores a 50 trabajos distintos. El perfil de cada trabajador indica un rendimiento distinto dependiendo del trabajo que se le asigne. Se cuantifica dicho rendimiento con un determinado valor, de forma que se plantea el problema de descubrir aquella asignación que sume el valor más alto. En principio, se puede estar tentado a diseñar un algoritmo simple, es decir, la computadora busca la solución evaluando todas las asignaciones posibles. Si se fija detenidamente, esto implica la evaluación de 50 posibilidades, es decir, del orden de 10^{64} asignaciones. Una computadora que evaluará 1 billón de posibilidades por segundo, y hubiera empezado hace 15000 millones de años, todavía no habría acabado [Garrido y Fernández, 2006].

El ejemplo anterior muestra que es necesario tener en cuenta la eficiencia de los algoritmos que usamos. Ahora bien, para este trabajo, no existe aún procedimiento que haya abordado el tema de investigación, por lo tanto la solución al problema es la propuesta de un algoritmo que cumpla con el objetivo propuesto, usando estructuras de datos con el fin de tener una mejor organización de la información y tener la certeza que los datos enviados entre los nodos no se pierden.

3.2 Metodología

En investigación científica o tecnológica, hay varias formas de atacar un problema, pero no todas son igualmente efectivas, siendo necesaria la utilización de un método que permita obtener resultados eficientes en todos los casos. Ello no significa que con su aplicación se pueda resolver de forma satisfactoria el problema propuesto, pero al menos, el investigador estará seguro de no haber dejado de lado fases importantes del proceso investigador o haber efectuado experiencias innecesarias. Mediante el método, generalmente aplicado por la mayoría de los investigadores, e independiente del resultado positivo o negativo, se pretende llevar a término la investigación de manera eficiente con un mínimo de esfuerzo, tiempo y costo.

El método científico es la forma de obtener conocimiento más elevado que puede aplicar el ser humano. En la actualidad es la cota intelectual más alta a la que ha llegado el Hombre; aun así no es autosuficiente ni infalible.

A partir de la observación o de la experimentación son elaboradas hipótesis, las cuales se mantienen mientras no puedan ser refutadas. A partir de varias evidencias demostradas se elaboran teorías que expliquen algunos aspectos de la realidad. [Álvarez, 1996].

La hipótesis de este trabajo es: Al enviar estructuras de datos complejas utilizando el paso de mensajes en un ambiente Cluster manejando memoria estática, la información no se pierde.

La hipótesis de este trabajo es: Al enviar estructuras de datos complejas utilizando el paso de mensajes en un ambiente Cluster manejando memoria dinámica, la información no se pierde.

3.3 Análisis del problema

Tomando en cuenta que los costos computacionales de las comunicaciones, entre menos mensajes se envíen, el rendimiento del programa será mejor. Basándose en esto MPI provee varias formas de empaquetar datos. Esto le permite maximizar el soporte de la información intercambiada en cada

mensaje. El paquete de mensajes en `MPI_Send` se compone de una dirección de memoria, un número de elementos y un tipo de datos. Es evidente que este mecanismo puede ser utilizado para enviar múltiples piezas de información en mensajes distintos. Por ejemplo, en el código 3.1 se utilizan tres llamadas a `MPI_Send` para distribuir los valores de tres números enteros a todos los procesos.

```
MPI_Send(&numero1,1,MPI_DOUBLE,dest,0,MPI_COMM_WORLD);
MPI_Send(&numero2,1,MPI_DOUBLE,dest,1,MPI_COMM_WORLD);
MPI_Send(&numero3,1,MPI_DOUBLE,dest,2,MPI_COMM_WORLD);
```

Código 3. 1 Envió individual de datos del mismo tipo en diferentes mensajes

Se puede eliminar una llamada a la rutina `MPI_Send` poniendo en un arreglo código 3.2 o en una estructura de datos y enviar los datos en una sola llamada a la rutina de MPI.

```
arreglo[0] = numero2;
arreglo[1] = numero3;
MPI_Send(&numero1,1,MPI_DOUBLE,dest,1,MPI_COMM_WORLD);
MPI_Send(arreglo,2,MPI_INT,dest,2,MPI_COMM_WORLD);
```

Código 3. 2 Envió organizado de datos del mismo tipo en un arreglo y un mensaje

Para que esto funcione, los elementos deben estar en lugares contiguos en la memoria. Si bien esto es cierto para las matrices, no hay garantías para las variables en general. Esta es ciertamente una manera legitima de escribir código y al enviar bloques de datos, es muy razonable y eficiente.

En el ejemplo anterior solamente se elimino una llamada y no se pudo incluir el elemento `numero1` ya que es un numero entero en lugar de un doble. En este caso una estructura de datos es la forma lógica de evitar este tipo de problema, ya que los elementos en una estructura están garantizados para estar en la memoria contigua y además es una estructura puedes tener datos heterogéneos [Sloan, 2005].

Una estructura de datos es una forma de organizar un conjunto de datos elementales (datos primitivos) con el objetivo de facilitar la manipulación o gestión de dichos datos como un todo, ya sea de manera general o particularmente [Martínez y Martín, 2003].

De la misma manera, si se necesita enviar más de una estructura es posible enviar dichas estructuras en diferentes llamadas a rutinas de envío y recepción. Con el objetivo de optimizar los recursos y de obtener el mejor resultado de la comunicación entre los nodos de un *Cluster,* las estructuras de datos se vuelven miembros de otras estructuras para así poder hacer una sola llamada a la rutina MPI_Send y transferir los datos complejos que se deseen.

Teniendo en cuenta que las comunicaciones entre los nodos afectan en gran manera el rendimiento general del sistema, para poder alcanzar un alto rendimiento en procesamiento paralelo, es necesario un sistema de comunicación entre nodos que presente una baja latencia y un alto ancho de banda. Para conseguir aumentar la velocidad de transmisión se debe utilizar una red de comunicación de alta velocidad, para que el nivel de latencia sea bajo, es importante no sólo aumentar la velocidad de transmisión de datos en la red, sino también reducir de forma significativa el tiempo necesario para establecer y configurar el envío de los mensajes [Lence, 2005].

Los elementos que intervienen en el paso de mensajes son el proceso que envia, el que recibe y el mensaje. En base a este principio en la comunicación del *Cluster*, la informacion, que son las estructuras de datos complejos se envian (figura 3.1) y dependiendo a cada procesador se ejecuten las operaciones programadas en la ejecucion del algoritmo (figura 3.2).

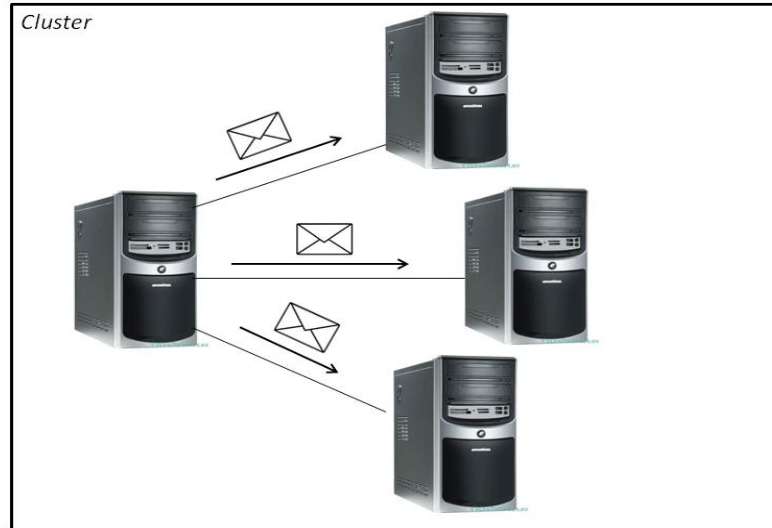

Figura 3. 1 Envío de datos del nodo maestro a los nodos esclavos

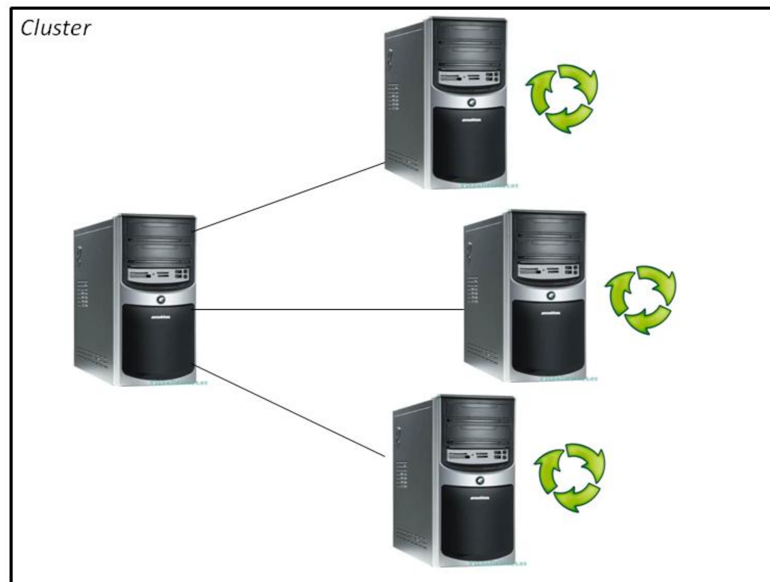

Figura 3. 2 Procesamiento de datos en nodos esclavos

Para concluir la comunicación y verificar que los datos enviados a los nodos esclavos no se pierdan, se crean nuevos mensajes a partir de la información modificada por cada procesador y se envían de vuelta al nodo maestro para comprobar que la comunicación se halla hecho correctamente y los datos no se pierdan (figura 3.3).

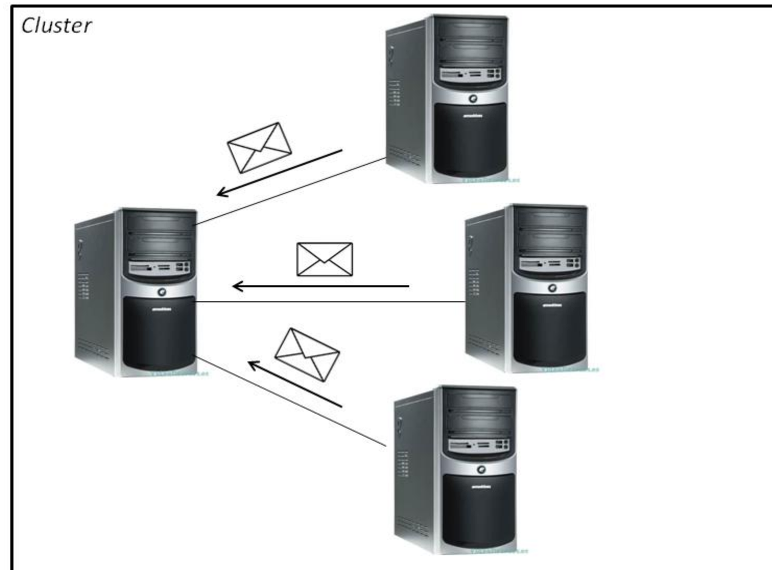

Figura 3. 3 envío de datos de los nodos esclavos al nodo maestro

3.4 Diseño de la solución

La escritura de un programa de computación consiste normalmente en implementar un método de resolución de un problema, que se ha diseñado previamente. Con frecuencia este método es independiente de la computadora utilizada: es igualmente válido para muchas de ellas. En cualquier caso es el método, no el programa, el que debe estudiarse para comprender como está siendo abordado el problema. El termino *algoritmo* se utiliza en informática para describir un método de resolución de un problema que es adecuado para su implementación como programa de computadora. Los algoritmos son la esencia de la informática; son uno de los centros de interés de muchas, si no todas, de las áreas del campo de la informática.

Muchos algoritmos interesantes llevan implícitos complicados métodos de organizando de los utilizados en el cálculo. Los objetos creados de esta manera se denominan estructuras de datos, y también constituyen un tema principal de estudio de informática [Sedgewick, 1992].

3.4.1 Algoritmo propuesto

Como primer paso se desarrolló el algoritmo con estructuras de datos complejas, para resolver el problema de la organización de los datos implementando el paso de mensajes (MPI), con el objetivo de optimizar recursos al enviar los datos en una estructura que engloba otras estructuras, evitando mandar cada dato en rutinas diferentes, se hace un solo envío y una solo llamada a la rutina encargada del envío de los datos.

El diagrama de la Figura 3.4 muestra paso a paso los puntos más relevantes en la aplicación del paso de mensajes con estructuras de datos complejas, en la aplicación del diagrama de flujo es importante señalar que las estructuras complejas que el diagrama muestra, pueden variar en la manera con la que se asigna memoria, es decir, pueden ser de tamaño fijo o dinámico.

Algoritmo con paso de mensajes

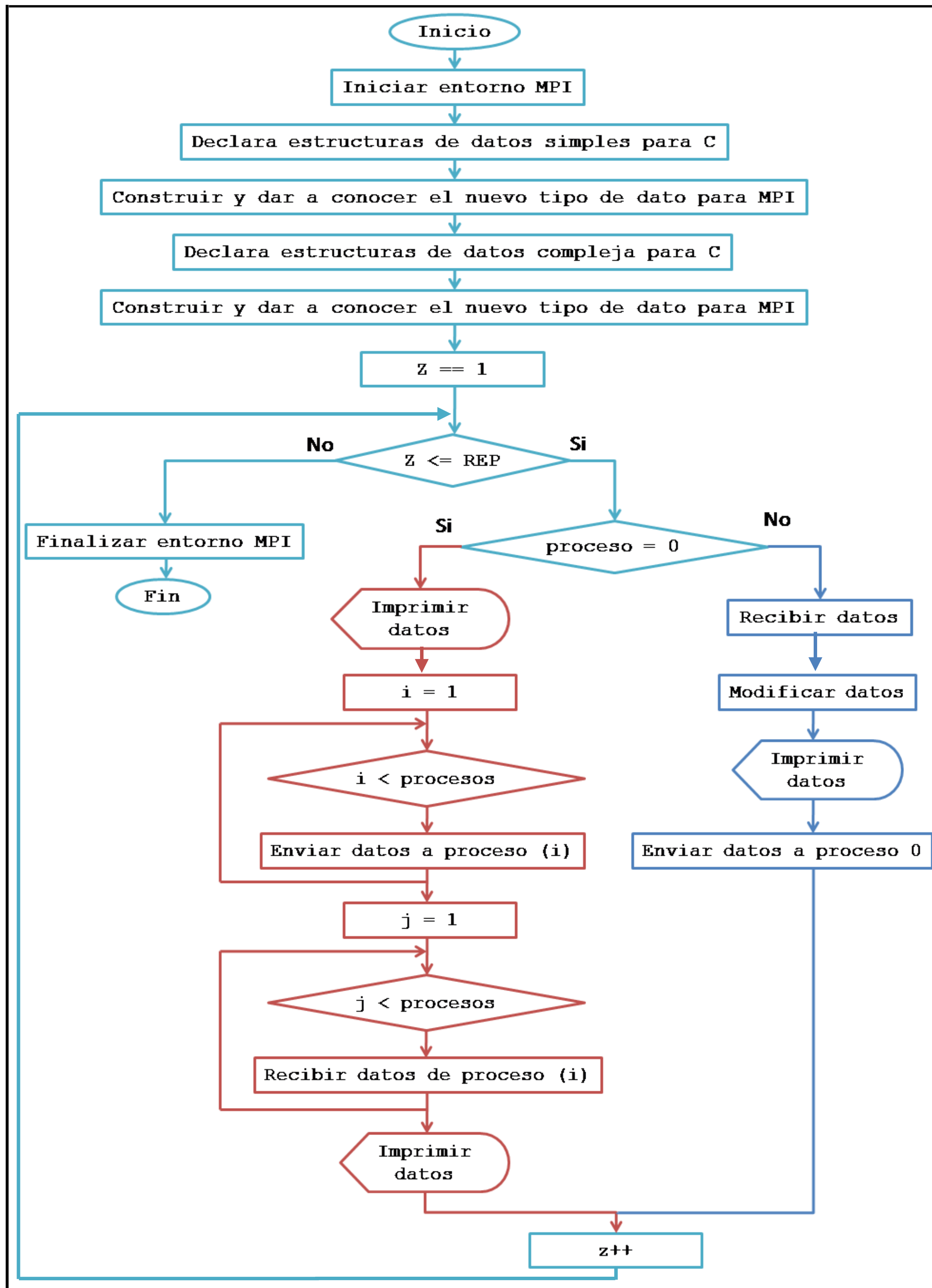

Figura 3. 4 Algoritmo con paso de mensajes con estructuras de datos complejas en un ambiente *Cluster*.

3.4.3 Estructuras de datos

Una estructura es una colección de una o más variables, de tipos posiblemente diferentes, agrupadas bajo un solo nombre para manejo conveniente. Las estructuras ayudan a organizar datos, en particular dentro de programas grandes, debido que permiten que a un grupo de variables relacionadas se les trate como una unidad en lugar de cómo entidades separadas [Kernighan y Ritchie, 1991].

Crear una estructura es definir un nuevo tipo de datos, denominado tipo estructura y declarar una variable de este tipo. En la definición del tipo estructura, se especifican los elementos que la componen así como sus tipos. Cada elemento de la estructura recibe el nombre de miembro [Ceballos, 1997].

Las estructuras de datos son una forma de organizar un conjunto de datos con el objetivo de facilitar su manipulación. Ofrecen ventajas y desventajas en relación a la simplicidad y eficiencia para la realización de operaciones sobre las estructuras de datos, por tal motivo la elección de cómo agrupar la información debe de ir en función a como se va a necesitar acceder a ciertos elementos.

3.4.3.1 Estructuras de datos ANSI C

Una estructura es un grupo de variables las cuales pueden ser de diferentes tipos sostenidas o mantenidas juntas en una sola unidad. La unidad es la estructura. Una estructura es una variable que representa lo que normalmente conocemos como registro, esto es, un conjunto de uno o más campos de igual o diferentes tipos [Ceballos, 1997].

En los lenguajes de programación *C/C++* se forma una estructura utilizando la palabra reservada `struct`, seguida por un campo etiqueta opcional, y luego una lista de miembros dentro de la estructura. La etiqueta opcional se utiliza para crear otras variables del tipo particular de la estructura (código 3.3):

```
struct campo_etiqueta {
     Tipo miembro_1;
     Tipo miembro_2;
     Tipo miembro_3;
};
```

Código 3. 3 Ejemplo de la declaración de una estructura de datos

Un punto y coma finaliza la definición de una estructura puesto que ésta es realmente una sentencia de los lenguajes de programación *C/C++.*

Para referirse a un determinado miembro de la estructura, se utiliza la notación como se muestra en el código3.4.

```
campo_etiqueta.miembro = 232323;
printf ("Valor = %d", campo_etiqueta.miembro);
```

Código 3. 4 Código para acceder a un miembro de una estructura.

La estructura del código 3.3 es conocida como simple porque los tipos de datos de los miembros de dichas estructuras están basados en los datos primitivos del lenguaje ANSI C. Por ejemplo: enteros (int), caracteres (char), reales (float) [Ceballos, 1997].

3.4.3.2 Estructuras de datos MPI

Antes de que una estructura se pueda utilizar en una función de MPI, es necesario definir un nuevo tipo de dato MPI. Un tipo de dato definido por el usuario se puede utilizar en lugar de los tipos de datos predefinidos por MPI. Tal puede ser utilizado como el tipo de datos en cualquier función de comunicación de MPI. Las funciones MPI que son los constructores de tipos de datos se utilizan para describir el diseño de memoria de estos nuevos tipos a partir de tipos de datos primitivos. Los tipos de datos definidos por el usuario o derivados, son objetos opacos que especifican las secuencias de los tipos de datos primitivos utilizados y una secuencia de desplazamientos de memoria. MPI proporciona el siguiente mecanismo para hacer esto.

MPI_Type_struct se utiliza para definir el nuevo tipo de dato. Esta función tiene cinco argumentos. Los primeros cuatro son los parámetros de entrada, mientras que el último es el parámetro de salida [Pacheco, 1997].

Sintaxis

```
MPI_Type_struct (int Contador, int Bloques, MPI_Aint Desplazamientos, MPI_Datatype Tipos_de_datos, MPI_Datatype *Nuevo_tipo)
```

`MPI_Aint`: Especifica una dirección valida de cualquier tipo de dato en C.

`MPI_Datatype`: Determina los tipos de datos validos para MPI.

Código 3. 5 Sintaxis para definir un nuevo tipo de dato en MPI

Parámetros de entrada

`Contador`: Número de bloques, también el número de entradas en los arreglos Bloques, Desplazamientos y Tipos de datos.

`Bloques`: Número de elementos en cada bloque, es decir las copias que se van a crear de un bloque especifico.

`Desplazamientos`: Desplazamientos en bytes de los elementos de cada bloque. Los desplazamientos son una secuencia de `n` pares (ecuación 3.1).

$$\{(t_0, d_0), (t_1, d_1), \ldots, (t_{n-1}, d_{n-1})\}$$

Ecuación 3. 1 Secuencia de n pares de los desplazamientos en memoria

Donde:

t_i → Es un tipo de dato primitivo

d_i → Son los desplazamientos en bytes.

En el ejemplo del código 3.5 los desplazamientos de la memoria se ven gráficamente en la figura 3.5:

```
{(MPI_FLOAT, 0), (MPI_FLOAT, 16), MPI_INT, 24}
```

Código 3. 6 Ejemplo de los desplazamientos de memoria

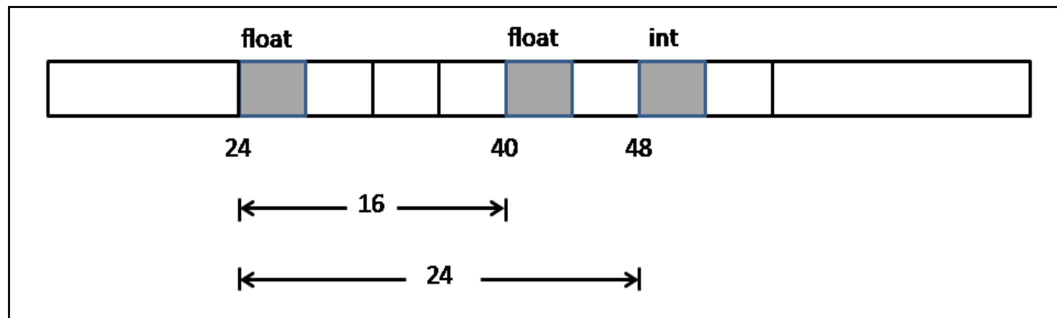

Figura 3. 5 Vista de la memoria con desplazamiento

Supongamos que el primer dato de tipo flotante (float) tiene una dirección de memoria 24 (figura 3.5), en la declaración de la estructura de datos tiene un desplazamiento 0 en la memoria, el siguiente dato del mismo tipo tiene un desplazamiento de memoria de 16 Bytes, por lo tanto la dirección de memoria del segundo datos es la 40 y por el ultimo el tercer dato de tipo entero tiene un desplazamiento en la memoria de 24 bytes por lo tanto la dirección de memoria es la 40 [Pacheco, 1997].

Los desplazamientos en la memoria son referidos al tamaño que los tipos de datos primitivos tienen, en la Tabla 3.1 se muestra el tamaño en bytes de los tipos de datos primitivos.

Tipo de dato	Desplazamiento
char	1 byte
int	4 bytes
float	4 bytes
double	8 bytes

Tabla 3. 1 Tipos de datos primitivos y sus desplazamientos de memoria

El tamaño de los tipos de datos varía dependiendo del sistema operativo y del compilador de C que utilice. Por ejemplo, en la mayoría de las estaciones de trabajo UNIX, un entero (int) tiene una longitud de 32 bits, mientras que la mayoría de los compiladores de C solo manejan 16 bits en una maquina basada en DOS. Así que se puede medir el tipo de datos por medio del

operador `sizeof` que proporciona C. La forma general del operador `sizeof` es como se muestra en el código 3.7.

```
sizeof(expresión)
```

Código 3. 7 Sintaxis del operador sizeof

Aquí, **expresión** es el tipo de datos o variable cuyo tamaño es medido por el operador *sizeof.* El operador *sizeof* evalúa el tamaño, en bytes, de su operando. El operando del operador *sizeof* puede ser una palabra reservada de C que represente a un tipo de datos (como *int, char, float*) o puede ser una expresión que se refiera a los tipos de datos cuyo tamaño se puede determinar (como una constante o el nombre de una variable) [Zhang, 2001].

`Tipos_de_datos`: Tipo de dato de los elementos en cada bloque (arreglo).

Parámetros de salida

`Nuevo_tipo`: Identificador del nuevo tipo de dato creado.

Teniendo en cuenta la creación de estructuras de datos con anterioridad, se crean las declaraciones necesarias para que el compilador de MPI pueda crear un nuevo tipo de dato y poder utilizar las estructuras en el *Cluster.*

3.4.4 Estructuras de datos complejos

Una estructura puede estar dentro de otra estructura, a esto se le conoce como anidamiento o estructuras anidadas. Ya que se trabajan con datos en estructuras si definimos un tipo de dato en una estructura y necesita definir ese dato dentro de otra estructura solamente se llama el dato de la estructura anterior [Alonso *et al*., 1998].

Las estructuras de datos se emplean con el objetivo principal de organizar los datos contenidos dentro de la memoria de la PC. En base a los tipos de datos primitivos, se pueden crear nuevos tipos con estructuras compuestas por uno o más de uno de los tipos mencionados [Cairo, 2006].

Teniendo en cuenta que el principal objetivo de las estructuras es organizar los datos del programa, es importante señalar que las estructuras permiten modificar globalmente las variables sin tener que recorrer el código, separando desde el inicio el espacio de memoria que se va a utilizar. En el anidamiento de estructuras se tiene que poner atención en la longitud de los datos y los tipos de datos, ya que se podría separar espacio que nunca se utiliza y terminar con la memoria disponible.

Para declarar un miembro como una estructura, es necesario haber declarado previamente ese tipo de estructura. En particular una estructura no puede ser miembro de ese mismo tipo de estructura, pero si puede contener un puntero o referencia a un objeto del mismo tipo. Ejemplo en el código 3.6 basándose en la declaración de las estructuras simples en la sección 3.4.3.1.

```
struct campo_etiqueta {
     Tipo_struct nodo_1;
     Tipo_struct nodo_2;
     Tipo_struct nodo_3;
};
```

Código 3. 8 Ejemplo de la declaración de una estructura de datos compleja

Los miembros de una estructura son almacenados secuencialmente, en el mismo orden en el que son declarados [Ceballos, 1997].

Con una variable declarada como una estructura, pueden realizarse las siguientes operaciones:

- Obtener su dirección por medio del operador `&`
- Acceder a uno o a todos sus miembros
- Copiar valores de una estructura de datos a otra estructura de datos

Una de las aplicaciones más interesantes y potentes de la memoria dinámica y de los punteros son las estructuras de datos. Las estructuras simples en lenguaje de programación C y C++ tienen una importante limitación: no pueden cambiar de tamaño durante la ejecución.

En muchas ocasiones se necesitan estructuras que puedan cambiar de tamaño durante la ejecución del programa. Por supuesto, podemos crear arreglos dinámicos, pero una vez creados, tu tamaño también será fijo, y para hacer que crezcan o disminuyan de tamaño, se debe reconstruir desde el principio.

CAPÍTULO IV IMPLEMENTACIÓN

4.1 Implementación del algoritmo

En primer lugar, se tiene que tener en cuenta que un algoritmo no tiene un tiempo fijo de ejecución: sino que ese tiempo depende del tamaño del problema. El algoritmo que se propone en la Sección 3.4, tiene las siguientes características:

- Tres estructura de datos primitivos con cuatro miembros de tipo entero Figura 4.1.
- Una estructura de datos compleja que tiene como miembros un arreglo de `n*n` de cada una de las estructuras de datos anteriores Figura 4.2.
- Un arreglo de tamaño `m`, de las estructuras de datos complejas Figura 4.3.

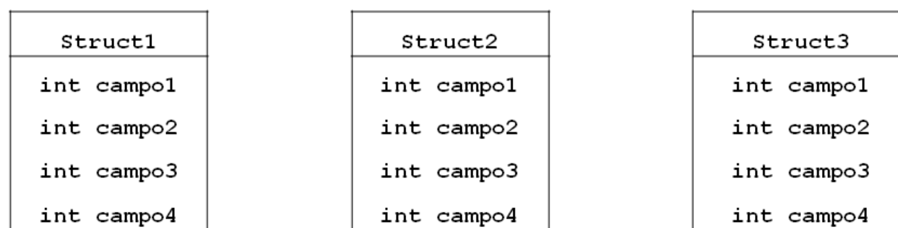

Figura 4. 1 Ejemplo de estructuras de datos simples

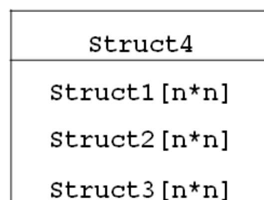

Figura 4. 2 Ejemplo de estructura de datos compleja

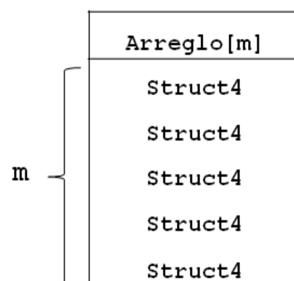

Figura 4. 3 Representación grafica del arreglo de estructuras de datos complejos

4.1.1 Estructuras de datos con memoria estática ANSI C

Las estructuras, en ocasiones conocidas como agregados, son colecciones de variables relacionadas bajo un nombre. Las estructuras pueden contener variables de diferentes tipos de datos, a diferencia de los arreglos, los cuales solo contienen elementos del mismo tipo. Las estructuras generalmente se utilizan para definir registros que van a almacenarse en archivos. Los apuntadores y las estructuras facilitan la formación de estructuras de datos más complejas. Las estructuras de datos son tipos de datos derivados, que se construyen por medio de objetos de otros tipos [Deitel *et al*, 2004]. Basándose en la descripción de la implementación, las estructuras de datos simples se declaran como se muestra en el código 4.1.

```
struct {                struct {                struct {
        int m1;                 int j;                  int m;
        int j1;                 int t;                  int t;
        int m2;                 int s;                  int s;
        int j2;                 int nop;                int nop;
} sxx;                  } maq;                  } trab;
```

Código 4. 1 Declaración de las estructuras de datos simples.

El código 4.1 declara las variables `sxx`, `maq` y `trab`, como estructuras o registros. La variable `sxx` comprende los campos `m1, j1, m2, j2` de tipo entero; La variable `maq` (máquina) comprende los campos `j, t, s y nop` que son de tipo de entero Y la variable `trab` (trabajo) comprende los campos `m, t, s, nop`.

4.1.1 Sintaxis de las estructuras de datos MPI

MPI_Type_struct es el constructor de datos más general en MPI y como consecuencia, provee al usuario una descripción completa de cada uno de los elementos de cada tipo. MPI_Type_struct se utiliza para definir el nuevo tipo de dato. Esta función tiene cinco argumentos. Los primeros cuatro son los parámetros de entrada, mientras que el último es el parámetro de salida [Pacheco, 1997].

Declaración de la estructura sxx

El primer elemento es un arreglo de tipo entero que contiene el número de copias que va a tener de cada tipo de dato en la estructura de datos (código 4.2).

```
int bloques1 [4] = {1, 1, 1, 1};
```

Código 4. 2 Declaración del arreglo de bloques para la estructura sxx

El segundo elemento es al igual que el primero un arreglo de tipo MPI_Aint (especifica un tipo de dato de *C* que contiene una dirección valida) que contiene los desplazamientos en bytes de los datos en la memoria.

El primer elemento no tiene desplazamiento de memoria aun, ya que en las estructuras los datos se almacenan en localidades contiguas, los desplazamientos empiezan a partir del segundo elemento. Se utilizo el operador `sizeof()` (sección 3.4.3.2) para conocer el tamaño del tipo de datos y así calcular dependiendo al sistema operativo, el correcto desplazamiento de un cierto tipo de dato Figura 4.4. El segundo elemento tiene un desplazamiento de memoria de un entero, el tercer elemento tiene un desplazamiento de dos enteros, que son los que le anteceden en la declaración de la estructuras y por último el cuarto elemento tiene tres enteros declarados anteriormente, por lo tanto tiene un desplazamiento de tres elementos (código 4.3).

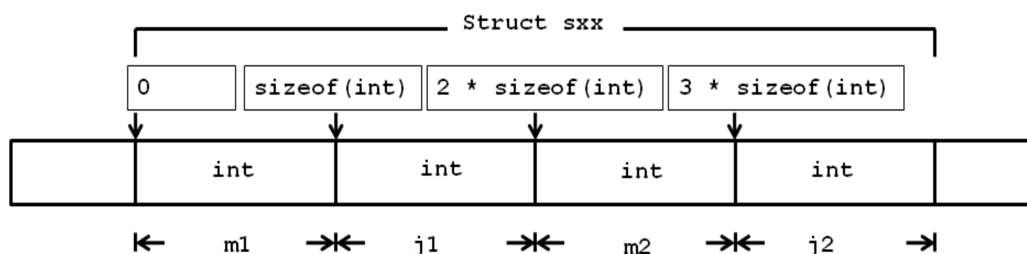

Figura 4. 4 Desplazamiento de memoria de la estructura sxx

```
MPI_Aint desplazamientos1 [4] = {0, sizeof (int), 2 *
sizeof (int), 3 * sizeof (int)};
```

Código 4. 3 Declaración del arreglo desplazamientos para la estructura sxx

El tercer elemento es otro arreglo de tipo MPI_Datatype (Tipo de datos MPI) que contiene los tipos de datos de cada elemento que son parte de la estructura de datos (código 4.4).

```
MPI_Datatype tipos1 [4] = {MPI_INT, MPI_INT, MPI_INT,
MPI_INT};
```

Código 4. 4 Declaración del arreglo tipos de datos para la estructura sxx

Una vez declarados los arreglos que contienen las copias de los elementos de cada bloque (bloques), los desplazamientos de memoria de cada uno de los tipos de datos (desplazamientos) y los tipos de datos de cada elemento en los bloques (tipos), se pasa a definir y crear el nuevo tipo de datos (código 4.5).

```
MPI_Datatype nuevotipo1;

MPI_Type_struct (4, bloques1, desplazamientos1, tipos1,
&nuevotipo1);
```

Código 4. 5 Declaración de la estructura sxx en MPI

Antes de que un tipo de dato derivado pueda ser utilizado, debe ser dado a conocer para el compilador MPI (código 4.6). Se puede pensar en esto como “compilar” el nuevo tipo de dato y se hace con la rutina MPI_Type_commit [Sloan, 2005].

```
MPI_Type_commit (&nuevotipo1);
```

Código 4. 6 Sentencia que da a conocer el nuevo tipo de dato al compilador MPI

Declaración de la estructura maq

El primer elemento, como en la sección anterior es un arreglo de tipo entero que contiene el número de copias que va a tener de cada tipo de dato en la estructura de datos (código 4.7)

```
int bloques2 [4] = {1, 1, 1, 1};
```

Código 4. 7 Declaración del arreglo de bloques para la estructura maq

El segundo elemento es un arreglo que contiene los desplazamientos en memoria. Se debe tener en cuenta los desplazamientos de la estructura anterior (Figura 4.4), la creación de nuevos tipos de datos como lo son las estructuras, los datos se guardan en localidades de memoria contigua, por lo tanto la declaración de los desplazamientos para esta estructura se pueden ver gráficamente en la figura 4.5 y en el código 4.8.

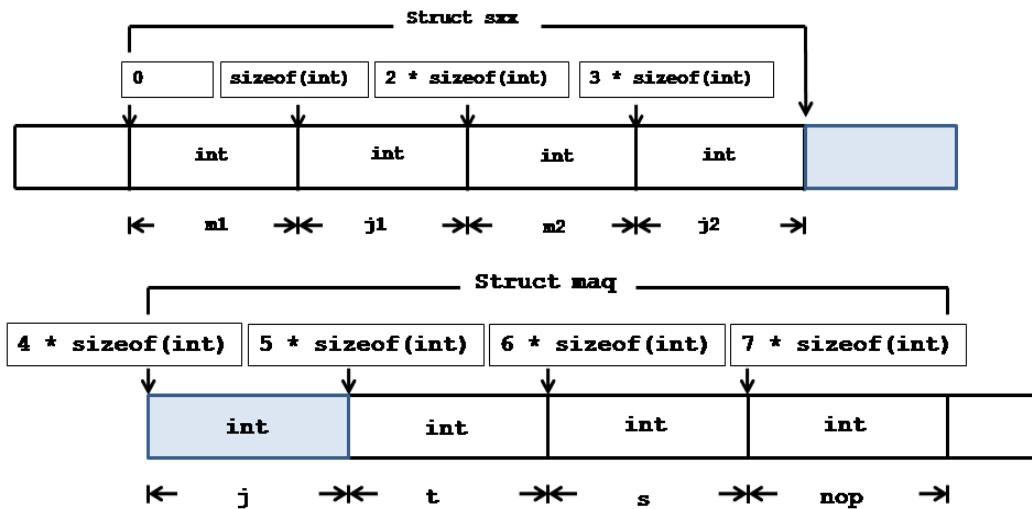

Figura 4. 5 Desplazamientos de memoria de la estructura maq

```
MPI_Aint desplazamientos2 [4] = {4 * sizeof (int), 5 *
sizeof (int), 6 * sizeof (int), 7 * sizeof (int)};
```

Código 4. 8 Declaración del arreglo desplazamientos para la estructura maq

El tercer elemento es otro arreglo del tipo MPI_Datatype, que tiene los tipos de datos de los elementos en la estructura (código 4.9).

```
MPI_Datatype tipos2 [4] = {MPI_INT, MPI_INT, MPI_INT,
MPI_INT};
```

Código 4. 9 Declaración del arreglo tipos de datos para la estructura maq

Se define y se da a conocer al compilador de MPI el nuevo tipo de dato derivado (código 4.10).

```
MPI_Datatype nuevotipo2;

MPI_Type_struct (4, bloques2, desplazamientos2, tipos2,
&nuevotipo2);

MPI_Type_commit (&nuevotipo2);
```

Código 4. 10 Declaración de la estructura maq en MPI

Declaración de la estructura trab

El primer elemento, el arreglo que tiene el número de elementos que tiene la estructura y cuantas copias de esos elementos se van a utilizar, se escribe como se muestra en el código 4.11.

```
int bloques3 [4] = {1, 1, 1, 1};
```

Código 4. 11 Declaración del arreglo de bloques para la estructura trab

Para los desplazamientos de la memoria de la estructura trab, se toman en cuenta los desplazamientos de las estructuras anteriores, por lo tanto, los desplazamientos no inician en cero, si no donde la estructura anterior termino (figura 4.6 y código 4.12).

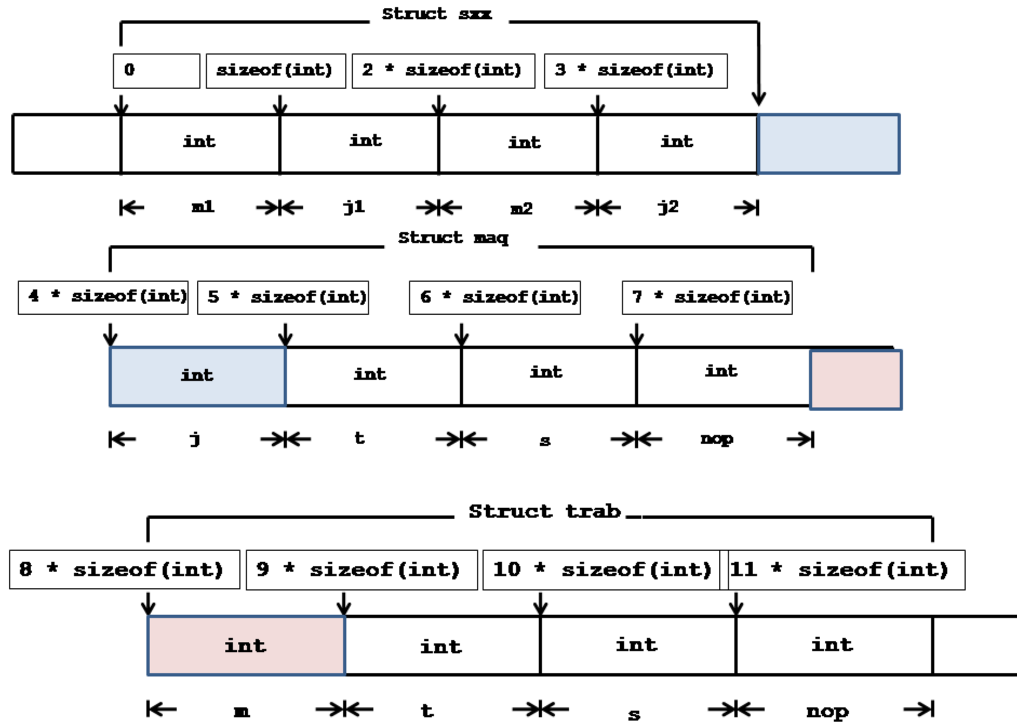

Figura 4. 6 Desplazamientos en memoria de la estructura de datos trab

```
MPI_Aint desplazamientos3 [4] = {8 * sizeof (int), 9 * sizeof (int), 10 * sizeof (int), 11 * sizeof (int)};
```

Código 4. 12 Declaración del arreglo desplazamientos para la estructura trab

El arreglo de tipos de datos es el elemento número tres en la declaración de la estructura, define los tipos de datos de cada bloque y se escribe como se muestra en el código 4.13.

```
MPI_Datatype tipos3 [4] = {MPI_INT, MPI_INT, MPI_INT, MPI_INT};
```

Código 4. 13 Declaración del arreglo tipos de datos para la estructura trab

Se define y se da a conocer al compilador de MPI el nuevo tipo de dato derivado (código 4.14).

```
MPI_Datatype nuevotipo3;
MPI_Type_struct (4, bloques3, desplazamientos3, tipos3,
&nuevotipo3);
MPI_Type_commit (&nuevotipo3);
```

Código 4. 14 Declaración de la estructura trab en MPI

4.1.3 Sintaxis de estructuras de datos complejos C, memoria estática

Las estructuras de datos con memoria estática son aquellas en las que el tamaño ocupado en memoria se define antes de que el programa se ejecute y no puede modificarse dicho tamaño durante la ejecución del programa. Estas estructuras de datos están implementadas en casi todos los lenguajes. Su principal característica es que ocupan solo una casilla de memoria, por lo tanto una variable simple hace referencia a un único valor a la vez, dentro de este grupo de datos se basan en los datos primitivos: enteros, reales, caracteres, boléanos [Deitel *et al*, 2004].

En el código 4.15 se define la variable *sched* (calendario), en la que los miembros `ops, maq y trab` son estructuras, que a su vez son un arreglo de dos dimensiones `n * n`.

```
struct {
     sxx ops [n][n];
     maq opm[n][n];
     trab opj[n][n];
}sched;
```

Código 4. 15 Declaración de la estructura de datos compleja para ANSI C

4.1.4 Sintaxis de estructuras de datos complejos MPI

Teniendo en cuenta las declaraciones de las estructuras de datos que se necesitan para crear la estructura compleja (`sxx, maq y trab`), que han sido declaradas para los lenguajes *C* (código 4.1), MPI (Sección 4.1.1) y la declaración de la estructura compleja para el lenguaje *C* (código 4.15), la declaración de la estructura compleja para MPI queda como se muestra en la siguiente sección.

Declaración de la estructura sched con memoria estática

Para la declaración de la estructura de datos anidada, que contiene como miembros a las estructuras de datos creadas con anterioridad, teniendo en cuenta que para el compilador de MPI, estas estructuras de datos son vistas como nuevos tipos de datos derivados y creados por el usuario, como se ha visto de igual forma la declaración de las estructuras se compone de cinco elementos. El primero de ellos es el arreglo de tipo entero que tiene el número de copias que va a haber de cado tipo de dato. En este caso se utiliza la variable `n` que se refiere al número de maquinas que se pueden simular o poner en un *Cluster*. Tomando en cuenta la sintaxis de la estructura para el lenguaje *C* en la sección 4.1.3, la estructura de datos va a estar compuesta por un arreglo de `n * n` copias de cada nuevo tipo de dato (código 4.16).

```
int bloques [3] = {n * n, n * n, n * n};
```

Código 4. 16 Declaración del arreglo de bloques para la estructura sched

En la declaración del arreglo de tipo `MPI_Aint` para la estructura de datos compleja, es necesario tener en cuenta que ya existen desplazamientos en memoria desde que se empezó a declarar las estructuras de datos simples. Por lo tanto aunque para la primera estructura, el primer miembro de dicha estructura no tiene un desplazamiento de memoria si no hasta el segundo elemento de la primera estructura empiezan los desplazamientos. De igual manera la estructura compleja empieza en ese primer desplazamiento. En este caso el primero elemento de la estructura es un entero por lo tanto los desplazamientos empiezan en la posición donde termina el primer dato. Los siguientes desplazamientos se calculan con el operador `sizeof()`, pero se obtiene el tamaño de las estructuras completas (figura 4.7 y código 4.17).

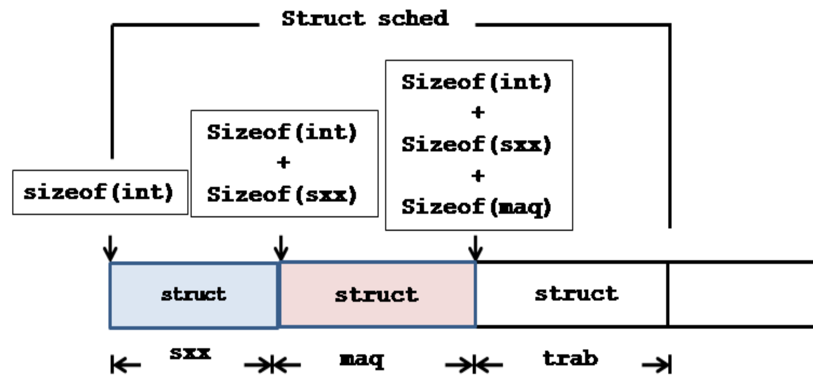

Figura 4. 7 Desplazamiento en memoria de la estructura de datos compleja (sched)

```
MPI_Aint desplazamientos [3] = {sizeof (int), sizeof
(int) + sizeof (sxx), sizeof (int) + sizeof (maq)};
```

Código 4. 17 Declaración del arreglo desplazamientos para la estructura sched

El arreglo de tipos de datos es el elemento número tres en la declaración de la estructura, define los tipos de datos de cada bloque, es importante señalar que los tipos de datos en esta estructura ya no son primitivos, es decir son tipos de datos derivados que se dieron a conocer al compilador MPI con anterioridad y se escribe como lo muestra el código 4.18.

```
MPI_Datatype tipos [4] = {nuevotipo1, nuevotipo2,
nuevotipo3};
```

Código 4. 18 Declaración del arreglo tipos de datos para la estructura sched

Se define y se da a conocer al compilador de MPI el nuevo tipo de dato complejo, ya que es una estructura de datos que tiene como miembros otras estructuras de datos (código 4.19).

```
MPI_Datatype nuevotipoX;
MPI_Type_struct (3, bloques, desplazamientos, tipos,
&nuevotipoX);
MPI_Type_commit (&nuevotipoX);
```

Código 4. 19 Declaración de la estructura sched en MPI

4.1.5 Estructuras de datos complejos con memoria dinámica

Las estructuras de datos dinámicas nos permiten crear estructuras de datos que se adapten a las necesidades reales a las que suelen enfrentarse nuestros programas. Pero no sólo eso, también nos permitirá crear estructuras de datos muy flexibles, ya sea en cuanto al orden, la estructura interna o las relaciones entre los elementos que las componen. Dichas estructuras están compuestas de otras pequeñas estructuras de datos a las que llamaremos nodos o elementos, que agrupan los datos con los que trabajará nuestro programa y además uno o más punteros autorreferenciales, es decir, punteros a objetos del mismo tipo nodo.

4.1.6 Asignación de memoria y estructuras de datos dinámicas

La creación y el mantenimiento de estructuras de datos dinámicas requiere de la asignación dinámica de memoria, que permite a un programa obtener más memoria en tiempo de ejecución para el almacenar nuevos nodos. Cuando esa memoria ya no es requerida por el programa, se libera de manera que pueda reutilizarse para asignar otros objetos en el futuro. El límite para la asignación dinámica de memoria puede ser tan grande como la cantidad de memoria física disponible en la computadora ó la cantidad de memoria virtual disponible en un sistema de memoria virtual.

Los operadores `nuevo (new)` y `borrar (delete)` son esenciales para la asignación dinámica de memoria. El operador `new` toma como argumento el tipo del objeto que se asignará en forma dinámica y devuelve un apuntador a un objeto de ese tipo. Por ejemplo, la instrucción del código 4.20.

```
nodo *identificador = new Nodo (10);
```

Código 4. 20 Sintaxis del operador new

Asigna el número de bytes determinado por `sizeof(nodo)`, ejecuta el constructor de `nodo` y almacena un apuntador a esta memoria en identificador.

El operador `delete` ejecuta el destructor de `nodo` y libera la memoria asignada con `new`; la memoria se devuelve al sistema para que pueda volver a asignarse en el futuro. Para la liberar la memoria asignada en forma dinámica por el operador `new` anterior, use la instrucción (código 4.21).

```
delete identificador;
```

Código 4. 21 Sintaxis del operador delete

Observe que identificador no es el que se elimina; el espacio al que este apunta es el que se elimina [Deitel *et al*, 2004].

La declaración de la estructura de datos compleja, y tomando en cuenta las mismas declaraciones de las estructuras de datos simples creados en la sección 4.1.1, la estructura se escribe como se muestra en el código 4.22.

```
typedef struct {
     maq **opm;
     maq **MOPM;
     maq **mopm;
     maq **opm_tmp;
     trab **opj;
     trab **MOPJ;
     trab **mopj;
     trab **opj_tmp;
}sched;
```

Código 4. 22 Sintaxis de la estructura de datos compleja con memoria dinámica

Arreglo de estructuras de datos dinámico

Cuando los elementos de un arreglo son estructuras, el arreglo recibe el nombre de arreglo de estructuras o arreglo de registros. Esta es una construcción muy útil y potente [Ceballos, 1997] (código 4.23).

```
sched poblacion [m]
```

Código 4. 23 Arreglo de estructuras complejas

Una vez declarado el arreglo de las estructuras de datos dinámicas, se procede a asignar la memoria a estas estructuras. Es necesario conocer el la variable n que se refiere al número de maquinas dispuestas, ya que es la base de la creación de la memoria dinámica. La Asignación se hace como se muestra en el código 4.24.

```
for(h = 0; h < TAMPOBLACION; h++){
    poblacion[h].opm = new maq* [nmaq];
    poblacion[h].opj = new trab* [nmaq];
    poblacion[h].MOPM = new maq* [nmaq];
    poblacion[h].MOPJ = new trab* [nmaq];
    poblacion[h].mopm = new maq* [nmaq];
    poblacion[h].mopj = new trab* [nmaq];
    poblacion[h].opm_tmp = new maq* [nmaq];
    poblacion[h].opj_tmp = new trab* [nmaq];
    for(i = 0; i < nmaq; i++){
    for(j = 0; j < nmaq; j++){
         poblacion[h].opm[i] = new maq [nmaq];
         poblacion[h].opj[i] = new trab [nmaq];
         poblacion[h].MOPM[i] = new maq [nmaq];
         poblacion[h].MOPJ[i] = new trab [nmaq];
         poblacion[h].mopm[i] = new maq [nmaq];
         poblacion[h].mopj[i] = new trab [nmaq];
         poblacion[h].opm_tmp[i] = new maq [nmaq];
         poblacion[h].opj_tmp[i] = new trab [nmaq];
    }
    }
}
```

Código 4. 24 Asignación de memoria dinámica a las estructuras de datos complejas

4.1.7 Barreras

El modelo de comunicación tiene que ver con el tiempo que un proceso pasa bloqueado tras llamar a una función de comunicación. Existen dos tipos: comunicación bloqueante y comunicación no bloqueante.

Barrera en la comunicación bloqueante

Una barrera es una operación especial colectiva que no deja al procesador continuar hasta que todos los procesos en el comunicador (`MPI_COMM_WORLD`) hallan llamado la rutina MPI_Barrier. Sirve para sincronizar la comunicación y estar seguros de que todos los procesadores han hecho los envíos y/o recepciones programados [Gropp *et al,* 1999]. La sintaxis de esta rutina se muestra en el código 4.25.

```
MPI_Barrier (MPI_Comm comm)

[entrada] comm-comunicador
```

Código 4. 25 Sintaxis de la barrera (MPI_Barrier)

Barreras en la comunicación no bloqueante

Hay una variedad de funciones que MPI utiliza para completar las operaciones sin bloqueo, el más simple de ellos es `MPI_Wait`. `MPI_Wait` bloquea el proceso hasta que la operación identificada por "request" termina: Sí se trata de un envío, ya sea que el mensaje se halla enviado o se copia en el buffer del sistema: Sí es una recepción, el mensaje se halla copiado en el buffer de recepción [Pacheco, 1997] (código 4.26).

```
MPI_Wait (MPI_Request request, MPI_Status status)

[entrada] request-identificador

[salida]  status-estado de la operación
```

Código 4. 26 Sintaxis de la barrera (MPI_Wait)

CAPÍTULO V PRUEBAS Y RESULTADOS

5.1 Pruebas

Probar un programa es la forma más común de comprobar que satisface su especificación y hace lo que el cliente quiere que haga. Sin embargo, las pruebas sólo son una de las técnicas de verificación y validación [Somerville, 2006]. La siguiente sección proporciona información sobre el plan de pruebas que se define en el estándar IEEE 829, debido al hecho de que este plan se presenta como una norma que cumple con casi todos los requisitos que los administradores puedan tener en el plan de pruebas [Leithold, 2007].

La necesidad de comprobar el correcto funcionamiento del algoritmo hace que sea indispensable un plan de pruebas, con el cual se procederá una serie de ensayos que permitan obtener resultados correctos y erróneos con el fin de analizar el proceso de ejecución. Con este conjunto de pruebas seremos capaces de determinar si nuestro programa es erróneo sobre todo en casos extremos y particulares, tanto si estos fallos se producen por la una mala implementación del programa o bien por un uso especifico que realiza el usuario.

5.1.1 Plan de pruebas

El propósito del plan de pruebas es explicar el alcance, enfoque, recursos requeridos, calendario, responsables y manejo de riesgos de un proceso de pruebas, esto con la intención de reunir toda la información necesaria para planear y controlar el desempeño de esta fase.

El aspecto más importante para realizar la planificación de este conjunto de pruebas es abarcar con ellas todos los requisitos que debe cumplir el algoritmo y que por lo tanto responda correctamente a las funcionalidades que se solicitan inicialmente. El plan de pruebas se describe de la tabla 5.1 a la tabla 5.5.

ID	Programación paralela.
Tipo de prueba	Prueba de funcionalidad.
Objetivo	Verificar que el programa corra en paralelo en los nodos del *Cluster.*
Descripción	El nodo principal manda mensajes a los nodos y estos regresan la confirmación de la recepción del mensaje.

Tabla 5. 1 Prueba para verificar la paralización del programa

ID	Comunicación bloqueante, memoria estática.
Tipo de prueba	Prueba de funcionalidad.
Objetivo	Verificar que la información transferida en la comunicación con los nodos del *Cluster* no se pierda.
Descripción	El nodo principal manda estructuras de datos complejas a los nodos del *Cluster*, dichos nodos modifican la información recibida y envían de regreso la información modificada.

Tabla 5. 2 Prueba que verifica la no pérdida de datos en la comunicación bloqueante

ID	Comunicación no bloqueante, memoria estática.
Tipo de prueba	Prueba de funcionalidad.
Objetivo	Verificar que la información transferida en la comunicación con los nodos del *Cluster* no se pierda.
Descripción	El nodo principal manda estructuras de datos complejas a los nodos del *Cluster*, dichos nodos modifican la información recibida y envían de regreso la información modificada.

Tabla 5. 3 Prueba que verifica la no pérdida de datos en la comunicación no bloqueante con memoria estática

ID	Comunicación bloqueante, memoria dinámica.
Tipo de prueba	Prueba de funcionalidad.
Objetivo	Verificar que el programa no pierda información en la comunicación con los nodos del *Cluster.*
Descripción	El nodo principal asigna memoria dinámica a las estructuras de datos complejas y las envía a los nodos del *Cluster*, dichos nodos modifican la información recibida y envían de regreso la información modificada.

Tabla 5. 4 Prueba que verifica la no pérdida de datos en la comunicación bloqueante con memoria dinámica

ID	Comunicación no bloqueante, memoria dinámica.
Tipo de prueba	Prueba de funcionalidad.
Objetivo	Verificar que el programa no pierda información en la comunicación con los nodos del *Cluster.*
Descripción	El nodo principal asigna memoria dinámica a las estructuras de datos complejas y las envía a los nodos del *Cluster*, dichos nodos modifican la información recibida y envían de regreso la información modificada.

Tabla 5. 5 Prueba que verifica la no pérdida de datos en la comunicación no bloqueante con memoria dinámica

5.1.2 Elementos necesarios para pruebas

La siguiente tabla contiene los elementos requeridos para realizar las pruebas del algoritmo (Tabla 5.6).

Elemento	Tipo y notas
SSH	Sirve para acceder a máquinas remotas a través de una red.
Cluster	Sección 1.7. Tabla 1.1 y Tabla 1.2.

Tabla 5. 6 Elementos requeridos para realizar las pruebas

5.1.3 Criterios de entrada

Es importante tener en cuenta que para la aplicación de las pruebas, existen tres parámetros para el algoritmo, que son los siguientes:

- Tamaño de la población (*m*): Se refiere al número de elementos que va a tener el arreglo de estructuras de datos complejas.
- Número de máquinas (*n*): Es el valor del número de elementos que va a tener el arreglo de estructuras de datos simples
- Cantidad de Iteraciones (ITER): Cantidad de repeticiones del algoritmo.

5.1.4 Ejecución del plan de pruebas

Durante la ejecución de pruebas pueden apreciarse los resultados obtenidos.

- Ejecución de la prueba “Programación paralela” (ver Tabla 5. 7)
- Ejecución de la prueba “Comunicación bloqueante, memoria estática” (ver Tabla 5. 8)
- Ejecución de la prueba “Comunicación no bloqueante, memoria estática” (ver Tabla 5. 9)
- Ejecución de la prueba “Comunicación bloqueante, memoria dinámica” (ver Tabla 5. 10)
- Ejecución de la prueba “Comunicación no bloqueante, memoria dinámica” (ver Tabla 5. 11)

ID: Programación paralela		**Evaluadores:** René López Ruiz. **Ejecutores:** René López Ruiz.
Fecha y Hora: 10/ Diciembre/ 2010; 10:00		
Entradas suministradas	**Resultados esperados**	**Resultados Obtenidos**
Ninguno	Que los datos enviados a los nodos del *Cluster*, se reciban exitosamente y sean capaces de regresar la información al nodo principal.	Los nodos del Cluster reciben y envían exitosamente los datos (Ver Figura 5.1).

Tabla 5. 7 Prueba ejecutada "Programación paralela"

```
Archivo  Editar  Ver  Buscar  Terminal  Ayuda
[rene@clusterciicap ~]$ mpirun -np 4 ./hola
Tenemos 4 procesadores, yo soy el proceso maestro:0 en maquina:clusterciicap.uaem.mx
¡Holaaa 1! Procesador 1 en maquina:clusterciicap.uaem.mx reportandose
¡Holaaa 2! Procesador 2 en maquina:clusterciicap.uaem.mx reportandose
¡Holaaa 3! Procesador 3 en maquina:clusterciicap.uaem.mx reportandose
--------------------------------------------------------------------------
mpirun noticed that process rank 2 with PID 18884 on node clusterciicap.uaem.mx exited on signal 11 (Segmentation fault).
--------------------------------------------------------------------------
[rene@clusterciicap ~]$
```

Figura 5. 1 Visualización de la prueba ejecutada "Programación paralela"

<table>
<tr><td colspan="2">ID: Comunicación bloqueante, memoria estática</td><td rowspan="2">Evaluadores: Wendy Torres Manjarrez, René López Ruiz
Ejecutores: Wendy Torres Manjarrez</td></tr>
<tr><td colspan="2">Fecha y Hora: 15/ Diciembre/ 2010; 16:30</td></tr>
<tr><td>Entradas suministradas</td><td>Resultados esperados</td><td>Resultados Obtenidos</td></tr>
<tr><td>Tamaño población (m): 6
Numero máquinas (n): 6
Numero de iteraciones (ITER): 30</td><td>Las estructuras complejas enviadas a los nodos del Cluster, sean recibidas correctamente. En los nodos las estructuras se modifiquen y regresen los datos al nodo maestro.</td><td>Los datos se enviaron y se recibieron satisfactoriamente, con la asignación de la memoria para las estructuras de manera estática (Ver Figura 5.2).</td></tr>
</table>

Tabla 5. 8 Prueba ejecutada “Comunicación bloqueante, memoria estática”

```
Archivo  Editar  Ver  Buscar  Terminal  Ayuda
Poblacion: 5  T 4 2 - TR1 = 35  Poblacion: 5  T 4 2 - TR2 = 35  Poblacion: 5  T 4 2 - TR3 = 35  Poblacion: 5  T 4 2 - TR4 = 35

Poblacion: 5  S 4 3 - SX1 = 35  Poblacion: 5  S 4 3 - SX2 = 35  Poblacion: 5  S 4 3 - SX3 = 35  Poblacion: 5  S 4 3 - SX4 = 35
Poblacion: 5  M 4 3 - MA1 = 35  Poblacion: 5  M 4 3 - MA2 = 35  Poblacion: 5  M 4 3 - MA3 = 35  Poblacion: 5  M 4 3 - MA4 = 35
Poblacion: 5  T 4 3 - TR1 = 35  Poblacion: 5  T 4 3 - TR2 = 35  Poblacion: 5  T 4 3 - TR3 = 35  Poblacion: 5  T 4 3 - TR4 = 35

Poblacion: 5  S 4 4 - SX1 = 35  Poblacion: 5  S 4 4 - SX2 = 35  Poblacion: 5  S 4 4 - SX3 = 35  Poblacion: 5  S 4 4 - SX4 = 35
Poblacion: 5  M 4 4 - MA1 = 35  Poblacion: 5  M 4 4 - MA2 = 35  Poblacion: 5  M 4 4 - MA3 = 35  Poblacion: 5  M 4 4 - MA4 = 35
Poblacion: 5  T 4 4 - TR1 = 35  Poblacion: 5  T 4 4 - TR2 = 35  Poblacion: 5  T 4 4 - TR3 = 35  Poblacion: 5  T 4 4 - TR4 = 35

Poblacion: 5  S 4 5 - SX1 = 35  Poblacion: 5  S 4 5 - SX2 = 35  Poblacion: 5  S 4 5 - SX3 = 35  Poblacion: 5  S 4 5 - SX4 = 35
Poblacion: 5  M 4 5 - MA1 = 35  Poblacion: 5  M 4 5 - MA2 = 35  Poblacion: 5  M 4 5 - MA3 = 35  Poblacion: 5  M 4 5 - MA4 = 35
Poblacion: 5  T 4 5 - TR1 = 35  Poblacion: 5  T 4 5 - TR2 = 35  Poblacion: 5  T 4 5 - TR3 = 35  Poblacion: 5  T 4 5 - TR4 = 35

Poblacion: 5  S 5 0 - SX1 = 35  Poblacion: 5  S 5 0 - SX2 = 35  Poblacion: 5  S 5 0 - SX3 = 35  Poblacion: 5  S 5 0 - SX4 = 35
Poblacion: 5  M 5 0 - MA1 = 35  Poblacion: 5  M 5 0 - MA2 = 35  Poblacion: 5  M 5 0 - MA3 = 35  Poblacion: 5  M 5 0 - MA4 = 35
Poblacion: 5  T 5 0 - TR1 = 35  Poblacion: 5  T 5 0 - TR2 = 35  Poblacion: 5  T 5 0 - TR3 = 35  Poblacion: 5  T 5 0 - TR4 = 35

Poblacion: 5  S 5 1 - SX1 = 35  Poblacion: 5  S 5 1 - SX2 = 35  Poblacion: 5  S 5 1 - SX3 = 35  Poblacion: 5  S 5 1 - SX4 = 35
Poblacion: 5  M 5 1 - MA1 = 35  Poblacion: 5  M 5 1 - MA2 = 35  Poblacion: 5  M 5 1 - MA3 = 35  Poblacion: 5  M 5 1 - MA4 = 35
Poblacion: 5  T 5 1 - TR1 = 35  Poblacion: 5  T 5 1 - TR2 = 35  Poblacion: 5  T 5 1 - TR3 = 35  Poblacion: 5  T 5 1 - TR4 = 35

Poblacion: 5  S 5 2 - SX1 = 35  Poblacion: 5  S 5 2 - SX2 = 35  Poblacion: 5  S 5 2 - SX3 = 35  Poblacion: 5  S 5 2 - SX4 = 35
Poblacion: 5  M 5 2 - MA1 = 35  Poblacion: 5  M 5 2 - MA2 = 35  Poblacion: 5  M 5 2 - MA3 = 35  Poblacion: 5  M 5 2 - MA4 = 35
Poblacion: 5  T 5 2 - TR1 = 35  Poblacion: 5  T 5 2 - TR2 = 35  Poblacion: 5  T 5 2 - TR3 = 35  Poblacion: 5  T 5 2 - TR4 = 35

Poblacion: 5  S 5 3 - SX1 = 35  Poblacion: 5  S 5 3 - SX2 = 35  Poblacion: 5  S 5 3 - SX3 = 35  Poblacion: 5  S 5 3 - SX4 = 35
Poblacion: 5  M 5 3 - MA1 = 35  Poblacion: 5  M 5 3 - MA2 = 35  Poblacion: 5  M 5 3 - MA3 = 35  Poblacion: 5  M 5 3 - MA4 = 35
Poblacion: 5  T 5 3 - TR1 = 35  Poblacion: 5  T 5 3 - TR2 = 35  Poblacion: 5  T 5 3 - TR3 = 35  Poblacion: 5  T 5 3 - TR4 = 35

Poblacion: 5  S 5 4 - SX1 = 35  Poblacion: 5  S 5 4 - SX2 = 35  Poblacion: 5  S 5 4 - SX3 = 35  Poblacion: 5  S 5 4 - SX4 = 35
Poblacion: 5  M 5 4 - MA1 = 35  Poblacion: 5  M 5 4 - MA2 = 35  Poblacion: 5  M 5 4 - MA3 = 35  Poblacion: 5  M 5 4 - MA4 = 35
Poblacion: 5  T 5 4 - TR1 = 35  Poblacion: 5  T 5 4 - TR2 = 35  Poblacion: 5  T 5 4 - TR3 = 35  Poblacion: 5  T 5 4 - TR4 = 35

Poblacion: 5  S 5 5 - SX1 = 35  Poblacion: 5  S 5 5 - SX2 = 35  Poblacion: 5  S 5 5 - SX3 = 35  Poblacion: 5  S 5 5 - SX4 = 35
Poblacion: 5  M 5 5 - MA1 = 35  Poblacion: 5  M 5 5 - MA2 = 35  Poblacion: 5  M 5 5 - MA3 = 35  Poblacion: 5  M 5 5 - MA4 = 35
Poblacion: 5  T 5 5 - TR1 = 35  Poblacion: 5  T 5 5 - TR2 = 35  Poblacion: 5  T 5 5 - TR3 = 35  Poblacion: 5  T 5 5 - TR4 = 35

 Tiempo total: 7.810000  Tiempo total: 38.080000  Tiempo total: 37.420000  Tiempo total: 39.080000 [rene@clusterciicap ~]$
```

Figura 5. 2 Visualización de la prueba ejecutada “Comunicación bloqueante, memoria estática”

<table>
<tr><td colspan="2">ID: Comunicación no bloqueante, memoria estática</td><td rowspan="2">Evaluadores: Wendy Torres Manjarrez, René López Ruiz
Ejecutores: Wendy Torres Manjarrez</td></tr>
<tr><td colspan="2">Fecha y Hora: 15/ Diciembre/ 2010; 17:00</td></tr>
<tr><td>Entradas suministradas</td><td>Resultados esperados</td><td>Resultados Obtenidos</td></tr>
<tr><td>Tamaño población (m): 6
Numero máquinas (n): 6
Numero de iteraciones (ITER): 30</td><td>Las estructuras complejas enviadas a los nodos del Cluster, sean recibidas correctamente. En los nodos las estructuras se modifiquen y regresen los datos al nodo maestro.</td><td>Los datos no se enviaron y se recibieron, las barreras fueron ignoradas, por lo tanto no hubo consistencia en los datos (Ver Figura 5.3).</td></tr>
</table>

Tabla 5. 9 Prueba ejecutada "Comunicación no bloqueante, memoria estática"

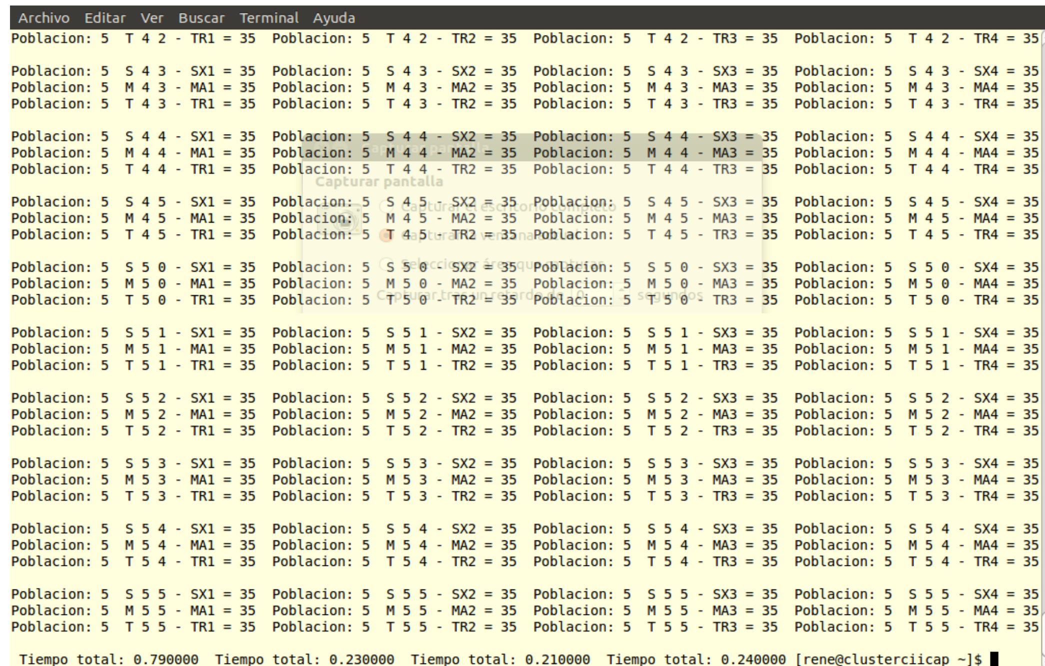

Figura 5. 3 Visualización de la prueba ejecutada "Comunicación no bloqueante, memoria estática"

ID: Comunicación bloqueante, memoria dinámica		**Evaluadores:** Ángel Felipe Lara Valladares, René López Ruiz **Ejecutores:** Ángel Felipe Lara Valladares
Fecha y Hora: 16/ Diciembre/ 2010; 19:30		
Entradas suministradas	**Resultados esperados**	**Resultados Obtenidos**
Tamaño población (m): 2 Numero máquinas (n): 2 Numero de iteraciones (ITER): 2	Las estructuras complejas enviadas a los nodos del *Cluster*, sean recibidas correctamente. En los nodos las estructuras se modifiquen y regresen los datos al nodo maestro.	Los datos se enviaron y se recibieron satisfactoriamente, con la asignación de la memoria para las estructuras de manera dinámica. Los nodos esclavos modificaron la información recibida, pero no pudieron regresar los datos, por lo tanto los datos se pierden (Ver Figura 5.4).

Tabla 5. 10 Prueba ejecutada “Comunicación bloqueante, memoria dinámica”

```
Archivo  Editar  Ver  Buscar  Terminal  Ayuda
Envio para bandera3 = 5025317
 --- Recv de proceso: 1 --- en NoMa
Poblacion 1 mopm 0 1 = 4        Poblacion 1 mopj 0 1 = 4        Poblacion 1 opm_tmp 0 1 = 4     Poblacion 1 opm_tmp 0 1 = 4


Estructura Recibida del proceso --> Check 0 en NM
Poblacion 0 omp 0 0 = 0         Poblacion 0 omj 0 0 = 0         Poblacion 0 MOPM 0 0 = 0        Poblacion 0 MOPJ 0 0 = 0
Poblacion 0 momp 0 0 = 0        Poblacion 0 momj 0 0 = 0        Poblacion 0 omp_tmp 0 0 = 0     Poblacion 0 omj_tmp 0 0 = 0
Poblacion 0 omp 0 1 = 0         Poblacion 0 omj 0 1 = 0         Poblacion 0 MOPM 0 1 = 0        Poblacion 0 MOPJ 0 1 = 0
Poblacion 1 opm 1 0 = 4         Poblacion 1 opj 1 0 = 4         Poblacion 1 MOPM 1 0 = 4        Poblacion 1 MOPJ 1 0 = 4
Poblacion 0 momp 0 1 = 0        Poblacion 0 momj 0 1 = 0        Poblacion 0 omp_tmp 0 1 = 0     Poblacion 0 omj_tmp 0 1 = 0

Poblacion 0 omp 1 0 = 0         Poblacion 0 omj 1 0 = 0         Poblacion 0 MOPM 1 0 = 0        Poblacion 0 MOPJ 1 0 = 0
Poblacion 0 momp 1 0 = 0        Poblacion 0 momj 1 0 = 0        Poblacion 0 omp_tmp 1 0 = 0     Poblacion 0 omj_tmp 1 0 = 0
Poblacion 1 mopm 1 0 = 4        Poblacion 1 mopj 1 0 = 4        Poblacion 1 opm_tmp 1 0 = 4     Poblacion 1 opm_tmp 1 0 = 4
Poblacion 0 omp 1 1 = 0         Poblacion 0 omj 1 1 = 0         Poblacion 0 MOPM 1 1 = 0        Poblacion 0 MOPJ 1 1 = 0
Poblacion 0 momp 1 1 = 0        Poblacion 0 momj 1 1 = 0        Poblacion 0 omp_tmp 1 1 = 0     Poblacion 0 omj_tmp 1 1 = 0

Poblacion 1 omp 0 0 = 0         Poblacion 1 omj 0 0 = 0         Poblacion 1 MOPM 0 0 = 0        Poblacion 1 MOPJ 0 0 = 0
Poblacion 1 momp 0 0 = 0        Poblacion 1 momj 0 0 = 0        Poblacion 1 omp_tmp 0 0 = 0     Poblacion 1 omj_tmp 0 0 = 0
Poblacion 1 omp 0 1 = 0         Poblacion 1 omj 0 1 = 0         Poblacion 1 MOPM 0 1 = 0        Poblacion 1 MOPJ 0 1 = 0
Poblacion 1 momp 0 1 = 0        Poblacion 1 momj 0 1 = 0        Poblacion 1 omp_tmp 0 1 = 0     Poblacion 1 omj_tmp 0 1 = 0

Poblacion 1 omp 1 0 = 0         Poblacion 1 omj 1 0 = 0         Poblacion 1 MOPM 1 0 = 0        Poblacion 1 MOPJ 1 0 = 0
Poblacion 1 momp 1 0 = 0        Poblacion 1 momj 1 0 = 0        Poblacion 1 omp_tmp 1 0 = 0     Poblacion 1 omj_tmp 1 0 = 0
Poblacion 1 omp 1 1 = 0         Poblacion 1 omj 1 1 = 0         Poblacion 1 MOPM 1 1 = 0        Poblacion 1 MOPJ 1 1 = 0
Poblacion 1 momp 1 1 = 0        Poblacion 1 momj 1 1 = 0        Poblacion 1 omp_tmp 1 1 = 0     Poblacion 1 omj_tmp 1 1 = 0

 --- Recv de proceso: 1 --- en NoMa
Poblacion 1 opm 1 1 = 4         Poblacion 1 opj 1 1 = 4         Poblacion 1 MOPM 1 1 = 4        Poblacion 1 MOPJ 1 1 = 4
Poblacion 1 mopm 1 1 = 4        Poblacion 1 mopj 1 1 = 4        Poblacion 1 opm_tmp 1 1 = 4     Poblacion 1 opm_tmp 1 1 = 4

*** Regresando de proceso 0 maquina clusterciicap.uaem.mx a NoMa ***

--- Estructura eviada en el nodo esclavo ---

Envio para bandera3 = 5025317
 --- Recv de proceso: 1 --- en NoMa
```

Figura 5. 4 Visualización de la prueba ejecutada “Comunicación bloqueante, memoria dinámica”

<table>
<tr><td colspan="2">ID: Comunicación bloqueante, memoria dinámica</td><td rowspan="2">Evaluadores: Ángel Felipe Lara Valladares, René López Ruiz
Ejecutores: Ángel Felipe Lara Valladares</td></tr>
<tr><td colspan="2">Fecha y Hora: 16/ Diciembre/ 2010; 20:00</td></tr>
<tr><td>Entradas suministradas</td><td>Resultados esperados</td><td>Resultados Obtenidos</td></tr>
<tr><td>Tamaño población (m): 2
Numero máquinas (n): 2
Numero de iteraciones (ITER): 2</td><td>Las estructuras complejas enviadas a los nodos del Cluster, sean recibidas correctamente. En los nodos las estructuras se modifiquen y regresen los datos al nodo maestro.</td><td>Los datos se enviaron y se recibieron satisfactoriamente, con la asignación de la memoria para las estructuras de manera dinámica. Los nodos esclavos modificaron la información recibida, pero no pudieron regresar los datos, por lo tanto los datos se pierden (Ver Figura 5.5).</td></tr>
</table>

Tabla 5. 11 Prueba ejecutada “Comunicación no bloqueante, memoria dinámica”

```
Archivo  Editar  Ver  Buscar  Terminal  Ayuda

Poblacion 0 omp 1 0 = 0        Poblacion 0 omj 1 0 = 0        Poblacion 0 MOPM 1 0 = 0       Poblacion 0 MOPJ 1 0 = 0
Poblacion 0 momp 1 0 = 0       Poblacion 0 momj 1 0 = 0       Poblacion 0 omp_tmp 1 0 = 0    Poblacion 0 omj_tmp 1 0 = 0
Poblacion 0 omp 1 1 = 0        Poblacion 0 omj 1 1 = 0        Poblacion 0 MOPM 1 1 = 0       Poblacion 0 MOPJ 1 1 = 0
Poblacion 0 momp 1 1 = 0       Poblacion 0 momj 1 1 = 0       Poblacion 0 omp_tmp 1 1 = 0    Poblacion 0 omj_tmp 1 1 = 0

Poblacion 1 omp 0 0 = 0        Poblacion 1 omj 0 0 = 0        Poblacion 1 MOPM 0 0 = 0       Poblacion 1 MOPJ 0 0 = 0
Poblacion 1 momp 0 0 = 0       Poblacion 1 momj 0 0 = 0       Poblacion 1 omp_tmp 0 0 = 0    Poblacion 1 omj_tmp 0 0 = 0
Poblacion 1 omp 0 1 = 0        Poblacion 1 omj 0 1 = 0        Poblacion 1 MOPM 0 1 = 0       Poblacion 1 MOPJ 0 1 = 0
Poblacion 1 momp 0 1 = 0       Poblacion 1 momj 0 1 = 0       Poblacion 1 omp_tmp 0 1 = 0    Poblacion 1 omj_tmp 0 1 = 0

Poblacion 1 omp 1 0 = 0        Poblacion 1 omj 1 0 = 0        Poblacion 1 MOPM 1 0 = 0       Poblacion 1 MOPJ 1 0 = 0
Poblacion 1 momp 1 0 = 0       Poblacion 1 momj 1 0 = 0       Poblacion 1 omp_tmp 1 0 = 0    Poblacion 1 omj_tmp 1 0 = 0
Poblacion 1 omp 1 1 = 0        Poblacion 1 omj 1 1 = 0        Poblacion 1 MOPM 1 1 = 0       Poblacion 1 MOPJ 1 1 = 0
Poblacion 1 momp 1 1 = 0       Poblacion 1 momj 1 1 = 0       Poblacion 1 omp_tmp 1 1 = 0    Poblacion 1 omj_tmp 1 1 = 0

 --- Recv de proceso: 3 --- en NoMa

Estructura Recibida del proceso --> Check 3 en NM
Poblacion 0 omp 0 0 = 0        Poblacion 0 omj 0 0 = 0        Poblacion 0 MOPM 0 0 = 0       Poblacion 0 MOPJ 0 0 = 0
Poblacion 0 momp 0 0 = 0       Poblacion 0 momj 0 0 = 0       Poblacion 0 omp_tmp 0 0 = 0    Poblacion 0 omj_tmp 0 0 = 0
Poblacion 0 omp 0 1 = 0        Poblacion 0 omj 0 1 = 0        Poblacion 0 MOPM 0 1 = 0       Poblacion 0 MOPJ 0 1 = 0
Poblacion 0 momp 0 1 = 0       Poblacion 0 momj 0 1 = 0       Poblacion 0 omp_tmp 0 1 = 0    Poblacion 0 omj_tmp 0 1 = 0

Poblacion 0 omp 1 0 = 0        Poblacion 0 omj 1 0 = 0        Poblacion 0 MOPM 1 0 = 0       Poblacion 0 MOPJ 1 0 = 0
Poblacion 0 momp 1 0 = 0       Poblacion 0 momj 1 0 = 0       Poblacion 0 omp_tmp 1 0 = 0    Poblacion 0 omj_tmp 1 0 = 0
Poblacion 0 omp 1 1 = 0        Poblacion 0 omj 1 1 = 0        Poblacion 0 MOPM 1 1 = 0       Poblacion 0 MOPJ 1 1 = 0
Poblacion 0 momp 1 1 = 0       Poblacion 0 momj 1 1 = 0       Poblacion 0 omp_tmp 1 1 = 0    Poblacion 0 omj_tmp 1 1 = 0

Poblacion 1 omp 0 0 = 0        Poblacion 1 omj 0 0 = 0        Poblacion 1 MOPM 0 0 = 0       Poblacion 1 MOPJ 0 0 = 0
Poblacion 1 momp 0 0 = 0       Poblacion 1 momj 0 0 = 0       Poblacion 1 omp_tmp 0 0 = 0    Poblacion 1 omj_tmp 0 0 = 0
Poblacion 1 omp 0 1 = 0        Poblacion 1 omj 0 1 = 0        Poblacion 1 MOPM 0 1 = 0       Poblacion 1 MOPJ 0 1 = 0
Poblacion 1 momp 0 1 = 0       Poblacion 1 momj 0 1 = 0       Poblacion 1 omp_tmp 0 1 = 0    Poblacion 1 omj_tmp 0 1 = 0

Poblacion 1 omp 1 0 = 0        Poblacion 1 omj 1 0 = 0        Poblacion 1 MOPM 1 0 = 0       Poblacion 1 MOPJ 1 0 = 0
Poblacion 1 momp 1 0 = 0       Poblacion 1 momj 1 0 = 0       Poblacion 1 omp_tmp 1 0 = 0    Poblacion 1 omj_tmp 1 0 = 0
Poblacion 1 omp 1 1 = 0        Poblacion 1 omj 1 1 = 0        Poblacion 1 MOPM 1 1 = 0       Poblacion 1 MOPJ 1 1 = 0
Poblacion 1 momp 1 1 = 0       Poblacion 1 momj 1 1 = 0       Poblacion 1 omp_tmp 1 1 = 0    Poblacion 1 omj_tmp 1 1 = 0
[rene@clusterciicap ~]$
```

Figura 5. 5 Visualización de la prueba ejecutada "Comunicación no bloqueante, memoria dinámica"

5.2 Análisis de resultados

En esta sección se presentan los resultados de rendimiento obtenidos de la evaluación en base al plan de pruebas mencionadas en la sección anterior.

Los parámetros de entradas se determinaron en base a las instancias del problema en el capítulo 3, esto porque en este capítulo el objetivo principal es analizar el impacto sobre el desempeño del algoritmo propuesto mientras no exista pérdida de datos en la comunicación paralela.

Para comprobar que los datos en la comunicación no se pierden, en los nodos la información se imprime y se verifica el estado de las rutinas de envío y recepción, dando como resultado lo siguiente:

En la comunicación bloqueante (Figura 5.2), la transferencia fue exitosa. Cada envío y recepción es verificado en los nodos del Cluster. Además que una vez modificada la información se imprime para verificar la consistencia de los

datos. Gracias a que MPI provee sincronización en los nodos por medio de las rutinas de tipo bloqueante, se tiene la certeza que los datos no se cruzan, es decir que no recibe mientras envía o viceversa (Figura 5.6). Lo cual retomando la hipótesis I, las pruebas realizadas proporcionan evidencia que demuestra que la hipótesis I, es verdadera. Utilizando el paso de mensajes, al enviar estructuras de datos complejas estáticas, la información no se pierde.

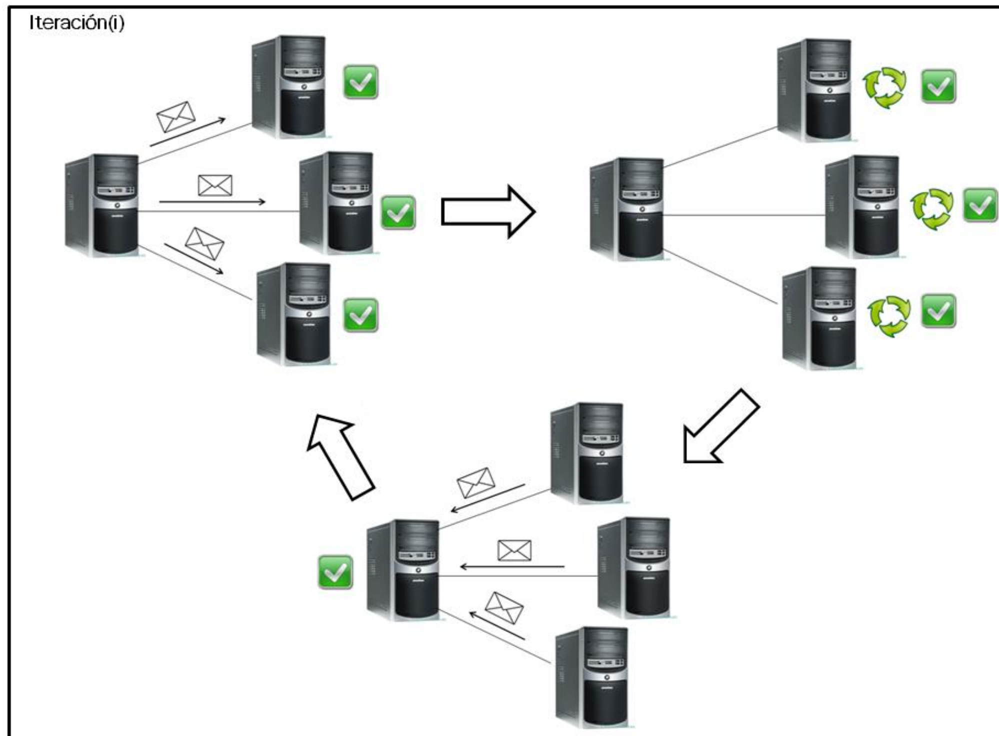

Figura 5. 6 Comunicación con memoria estática no bloqueante

Las rutinas que MPI tiene de comunicación no bloqueante, nos ayudan cuando se necesita que el procesador regrese al poner el mensaje en el buffer de envío para seguir con la ejecución del código. Pero en este caso y para las pruebas específicas, no sirve de mucho este tipo de comunicación, pero la comunicación fue exitosa (ver Figura 5.3), a reserva de que se tiene que hacer énfasis en el tipo barreras que se ponen (Sección 4.1.7) y en donde se ponen, ya que el procesador puede comenzar a recibir mientras está enviando y la información se perdería (figura 5.7).

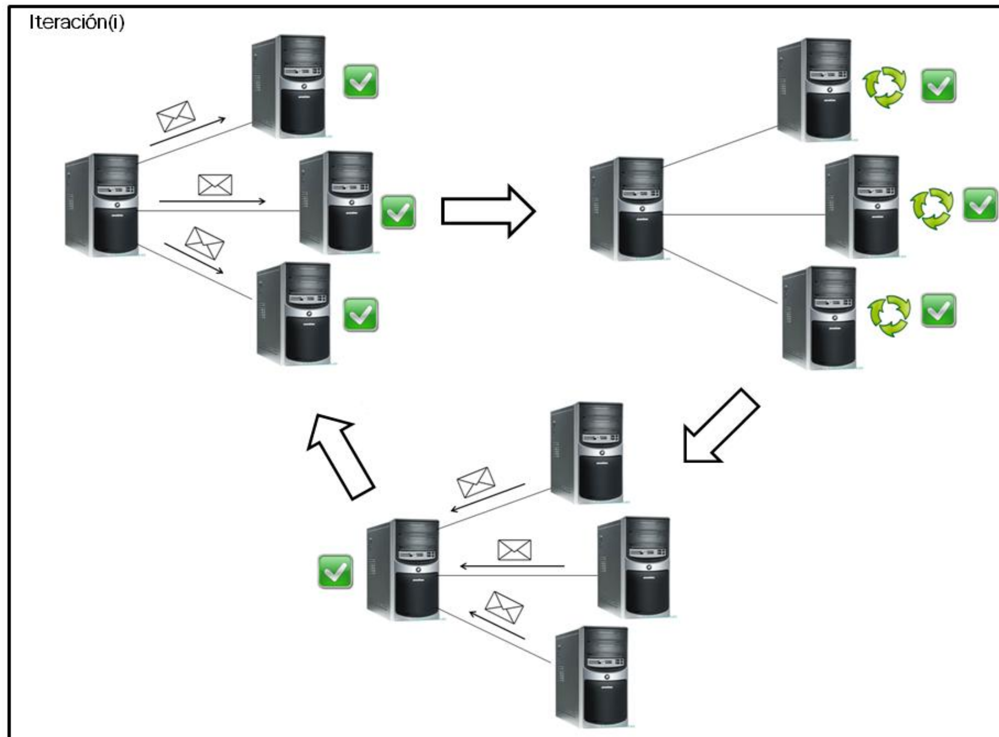

Figura 5. 7 Comunicación con memoria estática no bloqueante

La complejidad del algoritmo aumenta con la creación y asignación de memoria para las estructuras de datos complejas. Para el compilador del lenguaje *C*, esto no significa ningún problema, se puede crear y liberar memoria como se disponga en los recursos. A diferencia del lenguaje de programación *C*, en MPI las declaraciones de los nuevos tipos de datos se hacen de manera estática, es decir, cuando el programador necesita de la creación de tipos de datos como lo son las estructuras, MPI provee las herramientas necesarias para poder hacer este trabajo, con la desventaja de que su declaración se debe hacer antes de la ejecución del programa. Por tal motivo hay una discrepancia, mientras para el compilador C es posible crear y liberar memoria en las estructuras, MPI es fijo y una vez compilado el programa no se puede modificar las rutinas propias de MPI, por lo tanto los desplazamientos de las estructuras tampoco se pueden modificar. En las pruebas realizadas se muestra (ver Figuras 5.3 y 5.4) que, los datos son enviados exitosamente del nodo principal a los nodos del Cluster, los nodos esclavos modifican los datos recibidos pero no son capaces de crear los

mensajes para enviar de regreso al nodo principal. La razón es que en la ejecución se creó memoria que los nodos esclavos desconocen, ya que la rutina de envió de MPI provee la dirección del buffer donde empieza el mensaje y esta dirección no coincide con la que intenta acceder el nodo esclavo. Por tal motivo la comunicación no se realiza con éxito (Figura 5.8) y retomando la hipótesis planteada “Utilizando el paso de mensajes, al enviar estructuras de datos complejas dinámicas, la información no se pierde”, las pruebas realizadas no demuestran que dicha comunicación no es posible, si no que solo no se puedo demostrar de la manera con la que se trabajo en esta tesis. Con la implementación de algún otro método o con las actualizaciones que se hacen al estándar de MPI, se pueda sanar este problema.

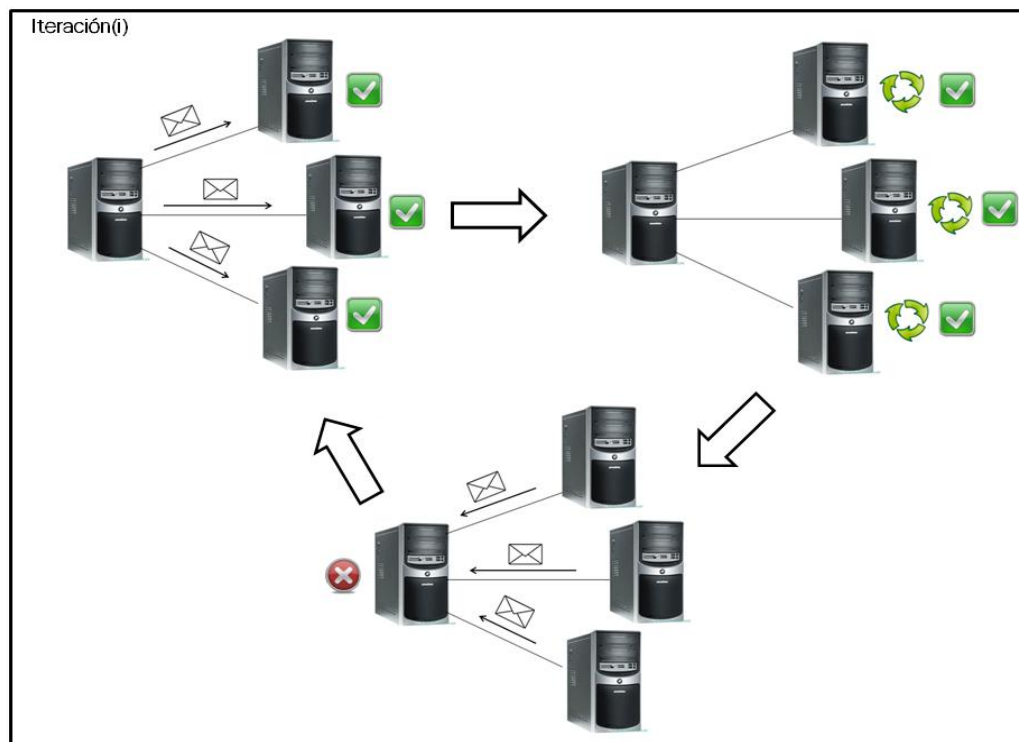

Figura 5. 8 Comunicación con memoria dinámica

CAPÍTULO VI CONCLUSIONES

6.1 Conclusiones

De acuerdo a las pruebas experimentales realizadas al algoritmo propuesto, utilizando el paradigma del paso de mensajes, se puede concluir que el propuesto es eficiente, cuando se trabaja con asignación de memoria de forma estática, se tiene la certeza que no hay pérdida de los datos enviados en todas las comunicaciones. Además el mismo algoritmo se probó bajo comunicaciones bloqueantes y no bloqueantes, concluyendo que la mejor manera de enviar datos con paso mensajes es la comunicación bloqueante, ya que esto permite la sincronización entre los nodos del Cluster. Si se requiere utilizar la comunicación no bloqueante, es importante señalar que el uso de barreras como los son MPI_Wait, proveen tiempos en la ejecución del programa para poder lograr sincronizar nodos del Cluster, parecido a la comunicación bloqueante.

Por otro lado el algoritmo también muestra cómo se trabajó con la asignación de la memoria dinámica, en este algoritmo la transferencia de datos no es posible, con el método propuesto en esta tesina no se logro evitar la pérdida de datos. Posiblemente con las nuevas versiones de MPI que vengan en el futuro, se logre la comunicación con memoria estática o algún otro método puedan sanar este problema.

La programación con paso de mensajes es simple, cuando se trabaja con tipos de datos conocidos por el compilador. La complejidad comienza cuando se requiere trabajar con múltiples datos de tipo complejo, es decir datos creados por el programador. Para futuros trabajos que necesiten basarse en el algoritmo propuesto, el uso de las comunicaciones bloqueantes es la mejor opción de transferencia de datos, ya que es necesario que haya una sincronización de trabajo entre los nodos y las rutinas de este tipo no proveen de control sobre los procesadores.

6.2 Trabajos futuros

La investigación propuesta ayudará a la implementación del paso de mensajes en cualquier algoritmo que se necesite paralelizar, especialmente en aquellos que manejan estructuras de datos complejas en la implementación de sus algoritmos. Este procedimiento ayudará al entendimiento de la declaración y creación de tipos de datos creados por el usuario para el compilador MPI.

Además servirá como base a futuras investigaciones que intenten resolver el problema de la asignación de memoria dinámica a estructuras de datos complejas o investigaciones de otro tipo de ambientes, como puede ser un ambiente GRID.

REFERENCIAS

[Alonso *et al*., 1998]. Alonso Jordá Pedro, García Granada Fernando, Onaindía de la Rivaherrera Eva. *Diseño e implementación de programas en lenguaje C.* Valencia: Universidad Politécnica del Valencia.

[Álvarez, 1996]. Álvarez Cáceres Rafael. *El método científico en las ciencias de la salud.* Madrid: Díaz de Santos.

[Cairo, 2006]. Cairo Osvaldo. *Fundamentos de programación. Piensa en C.* México: PEARSON EDUCACION.

[Ceballos, 1997]. Ceballos Francisco. Javier. *Enciclopedia del lenguaje C.* Madrid: RA-MA Editorial.

[Cegarra, 2004]. Cegarra José, *Metodología de la investigación científica y tecnológica.* Madrid: Diaz de Santos.

[Coulouris *et al*., 2001]. Coulouris, G., Dollimore, J., Kingberg, T. *Distribuited System.*China: Pearson Education Limited.

[Deitel *et al*, 2004]. Deitel, Harvey M., Paul J. *Como programar en C/C++ y java.* México: Pearson Educación.

[Fernández, 2004] Fernández, M. *Nuevas tendencias en la informática.*España: COMPOBELL.

[Garrido y Fernández, 2006]. Garrido Carrillo Antonio, Fernández Valdivia Joaquín. *Abstracción y estructuras de datos en C++.* Madrid: DELTA publicaciones.

[Grama *et al*., 2003] Grama, A., Gupta, A., Karypis, G., Kumar, V. *Introduction to Parallel Computing.* England: Pearson.

[Geist *et al*, 2000]Geist Al, Buguelin Adam, Dongarra Jack, Jiang Weicheng, Mancheck Robert, Sunderam Vaidy. *Parallel Virtual Machine*. United States of America: Massachusetts Institute of Technology.

[Gropp *et al*., 2005] Gropp, W., Huss-Lederman, S., Lumsdaine, A., Lusk, E., Nitzberg, B., Saphir, W.*MPI.*Netherlands: IOS Amsterdan.

[Gropp *et al,* 1999] Gropp William, Lusk Ewing, Skjellum Anthony. *Using MPI: Portable parallel programming with the Message-Passing Interface.* Estados Unidos de America: Massachusetts Institute of Technology.

[Jacob *et al*., 2005] Jacob Bart, Brown Michael, Fukui Ken taro, Trivedi Nihar. *Introduction to GRID computing.* United States of America: IBM.

[Karniadakis *et al*, 2007]. Em Karniadakis, G., Kirby II, R. *Parallel Scientifc Computing in C++ and MPI.*United States of America: Cambridge.

[Kernighan y Ritchie, 1991]. Kernighan W. Brian, Ritchie M. Dennis. *El lenguaje de programación C. México*: Pearson Educación.

[Kopper, 2005] Kooper Karl. *Linux Enterprise Cluster.* United States of America: No Start Press, Inc.

[Leithold, 2007]. Leithold Alfred. *Structured testing in practice.* Alemania: 2007.

[Lence, 2005] Lence, P. *Técnicas paralelas aplicadas a optimización no lineal en sistemas de memoria distribuida.*España: USC.

[Maozhen y Baker, 2005] Maozhen Li, Baker Mark. *The Grid core technologies.* Inglaterra: Wiley.

[Martínez, 2006] Martínez M. Santiago. *Difusión automático de datos bajo cómputo paralelo en clusters.* México D.F.: IPN

[Martínez y Martín, 2003]. Martínez Gil Francisco, Martin Quetglás Gregorio. *Introducción a la programación estructurada en C.* España: Maite Simon.

[Morton y Pentico, 1993] Morton E. Thomas, David W. Pentico. *Heuristic Scheduling Systems.* Canadá: ETA.

[Munilla y García, 2003] Munilla Calvo Eduardo, García Valcárcel Ignacio. *Cómo implantar Software libre, Servicios web y el GRID computing para ahorrar costes y mejorar las comunicaciones en su empresa.* Madrid: FC Editorial.

[Pacheco, 1997] Pacheco, P. *Parallel Programming with MPI.*United States of America: Morgan Kauímann Publisher, Inc.

[Prabhu, 2008] Prabhu, C.S.R. *Grid and Cluster Computing.* India: Easter PHI Learning Pvt. Ltd

[Rosano, 1998]. Rosano, F. L. *Tecnologia conceptos, problemas y perspectivas.* Mexico D.F.: Siglo veitiuno editores.

[Sedgewick, 1992]. Sedgewick Robert. *Algoritmos en C++.* Massachusetts: Addison-Wesley Publishing Company, Inc.

[Sloan, 2005]. Sloan D. Joseph. *High performance LINUX Clusters with OSCAR, Rocks, openMosix & MPI.* Estados Unidos de América: O'Reilly.

[Somerville, 2006]. Somerville Ian. *Ingeniería del Software.* Madrid: Pearson Educación, S.A.

[Wang *et al*., 2006] Wang Jun, Yi Zhang, Zurda M. Jacke, Lu Bao-Liang, Yin Hujun. *Advances in Neuronal Networks – ISNN 2006.* China: Springer.

[Zhang, 2001]. Zhang Tony. *Aprendiendo C en 24 horas.* Indiana: Pearson Education.