Centro de Investigación y Estudios
Avanzados del I.P.N.
Depto. Ing. Eléctrica
Sección Computación

Tesis de Maestría: *" Aplicaciones de Procesamiento en Paralelo Usando Transputers "*

Que para Obtener el Grado de
Maestro en Ciencias
Especialidad en Ingeniería Eléctrica

Presenta:

Julio César Gallardo[1]

Codirigida por:
Dr. José Luis Marroquín Zaleta (CIMAT)
M. en C. Feliú D. Sagols Troncoso (CINVESTAV)

México, D.F. Octubre de 1996

[1]Becario de CONACYT

Agradecimientos:

A mi familia, por su enorme apoyo moral y económico.

A CONACYT y a todos los mexicanos, quienes a través de sus impuestos y la canalización de éstos por dicha institución, me han proporcionado el medio económico apropiado para realizar mis estudios de posgrado.

A mi asesor de tesis José Luis Marroquín Z., por su valiosa asesoría y su gran apoyo económico.

Al profesor Feliú Sagols T., por la motivación académica que recibí de él a través de la impartición de sus cursos.

A los sinodales M. en. C. Feliú Sagols T., Dr. Guillermo Morales L., Dr. José Luis Marroquín Z. y Dr. Salvador Botello R., por aceptar formar parte del jurado y asumir lo que esto implica.

Dedicada:

A mi familia con mucho cariño y todo con todo mi ser,

A mis padres, Guillermo y Cholita.

A mis hermanos, Elias, Paty, Diana, Marco, Charly, Memo.

A mis sobrinos, Cristian y Niky.

A mi otra familia, la formada por los grandes amigos y cuates, a quienes no hace falta tener su sangre en común para llamarlos y considerarlos hermanos. A cada una de esas familias que me encuetro en cada ciudad que voy.

A mi hermoso y lindo Guanajuato encantado y a mi ranchito en el que viví mi enfancia: San Diego Quiriceo.

A "Jesús", quien es y seguirá siendo para mi el ideal más grande, al que todos los seres humanos debemos aspirar. Por ser "una incógnita" y la única razón para cambiar la dirección de mi vida. A él, aunque su historia no sea real y sólo exista en la imaginación, y aunque desee profundamente su existencia.

A todas aquellas personas que se han encontrado así mismas y han descubierto que es necesario cambiar para ser mejores. A todas estas personas que se esfuerzan por encontrar la verdad y tratan de vivir con ella. A aquellas personas que se han dado cuenta que es necesario vivir más allá de lo individual y que buscan una vida social más justa y sin conveniencias personales.

A la mujer, la flor con aroma más hermoso de este planeta, pero a la vez, la de mayor cuidado.

Aplicaciones de Procesamiento en Paralelo Usando Transputers

Julio Cesar Gallardo

Octubre de 1996

Resumen

En este trabajo se presenta la paralelización de dos aplicaciones usando una *red de transputers* que actúan como nodos de procesamiento, lenguaje C para su codificación y una computadora *PC 486* con *Windows95* actuando como "*host*". La primer aplicación consiste en optimizar una estructura de acero para una nave industrial sujeta a cargas laterales y utilizando como información un catálogo de secciones transversales; esto es conseguido mediante el uso de una familia de algoritmos paramétricos que permiten realizar búsquedas estocásticas. Esta familia de algoritmos es una generalización de los algoritmos genéticos, las estrategias evolutivas y el recocido simulado. La segunda aplicación consiste en construir un filtro para la eliminación de ruido en imágenes, usando un autómata celular síncrono determinístico. Además, en este trabajo se incluyen las experiencias adquiridas en el uso de sistemas paralelos basados en *transputers*.

Palabras Clave:
algoritmos genéticos, autómata celular, estrategias evolutivas, modelo CSP, occam, paralelismo, proceso, procesamiento de imágenes, tarjeta IMS B008, transputers, recocido simulado.

Contenido

1. Objetivos

- Implementar una familia de algoritmos de búsqueda estocástica que incluye a los algoritmos genéticos, estrategias evolutivas y recocido simulado como casos particulares; debiendo resolver el siguiente problema: "Optimización de estructuras de acero para una nave industrial sujeta a cargas laterales" Esta aplicación se realizará mediante una *red de transputers* que actúan como nodos de procesamiento, lenguaje C para su codificación y una computadora *PC 486* con *Windows95* actuando como computadora sede *(host)*.

- Construir e implementar un filtro para la eliminación de ruido en imágenes usando un autómata celular síncrono determinístico. También se realizará en el mismo equipo de cómputo y usando el mismo lenguaje que la aplicación anterior.

- Producir un texto que transmita las experiencias adquiridas en el uso de sistemas paralelos basados en *transputers*.

2. Introducción

En este capítulo se presenta una idea general de lo que es la computación en paralelo y sus principales aplicaciones. Además, se muestra una descripción del contenido y la organización de este trabajo.

2.1. Tendencia de la Computación

Muchos problemas de gran interés práctico requieren de un alto grado de procesamiento de cómputo, y frecuentemente no pueden ser resueltos en el lapso de tiempo que la aplicación exige, en una computadora común (secuencial), aún cuando mejore su rendimiento. Esto se debe a que la velocidad de las componentes computacionales está muy cercana a los límites impuestos por la velocidad de la luz. Así que el incremento en el rendimiento de las computadoras secuenciales va siendo cada vez menor. En tales problemas, solamente la computación en paralelo puede dar una solución viable.

2.2. ¿Qué es la Computación en Paralelo?

Una actividad que se realiza en paralelo es aquella que se efectúa por varias entidades (ya sean idénticas o heterogéneas), cada una desarrollando una parte de la actividad y trabajando juntas hacia un objetivo común. En el caso de paralelismo en computación, tales entidades pueden ser computadoras o microprocesadores. Esta idea de paralelismo no es nueva, ya que siempre ha sido una característica importante del desarrollo de todas las actividades humanas y sociales. Por ejemplo, en la organización y desarrollo del trabajo de una empresa, cada uno de sus grupos de trabajo, ya sea de una o varias personas (unidades de procesamiento en computación en paralelo), realiza una actividad (procesos subordinados) en forma coordinada (sincronización) con otros grupos de trabajo, o independientemente de los demás (asíncrona); con la finalidad de lograr un objetivo común. Para ésto, se tiene otro grupo de trabajo que se encarga de recoger, analizar, e integrar los resultados del trabajo de cada individuo o grupo (procesos de control, y su jerarquización), y así reportar el producto final.

2.3. Dificultades en el Área del Paralelismo

Además de tener el problema de crear máquinas con nuevas arquitecturas de hardware y un apropiado sistema de software, se tiene el problema de cambiar el enfoque, de secuencial a paralelo, que el programador le da a los algoritmos al resolver una aplicación en el momento de escribir el programa. Esto es, cuando el programador se enfrenta a un problema paralelizable (es decir, que el problema pueda ser dividido en varias partes y que éstas puedan resolverse concurrentemente) encuentra dificultades considerables en la definición de las actividades que pueden realizarse concurrentemente, lo cual se debe al hábito que se tiene de resolver los problemas de manera secuencial. La situación es aún más compleja por el hecho de que las técnicas de diseño y las herramientas de desarrollo para programas secuenciales no son los apropiados para la complejidad del software en paralelo. Sin embargo, es de esperarse que la programación paralela sea tan sencilla (o compleja) como la programación secuencial para aquellas personas quienes en el futuro lleguen a trabajar con computadoras paralelas sin antes usar la contraparte secuencial pura.

2.4. Aplicaciones del Paralelismo

Entre las principales aplicaciones del procesamiento en paralelo se tienen las siguientes:

1. Procesamiento de imágenes y reconocimiento de patrones.
2. Resolución de una gran variedad de problemas de optimización.
3. Simulación de sistemas biológicos y químicos.
4. Apoyo al diseño en campos como la tecnología mecánica, eléctrica y aereoespacial.
5. Control de estaciones de potencia nuclear.
6. Comprensión del lenguaje natural.

Sin el uso de la computación en paralelo, muchas de estas aplicaciones podrían difícilmente ser realizadas.

2.5. Contenido y Organización de este Trabajo de Tesis

En el capítulo 3 se presentan las herramientas que serán usadas para resolver las aplicaciones de optimización de estructuras y de proceso de imágenes de este trabajo sobre un sistema de transputers. También se presentan las características de los transputers y de la tarjeta B008; así como las instrucciones para la instalación de las herramientas que se usan para crear aplicaciones tanto en Occam como en lenguaje C, incluyendo una herramienta para la depuración desde Windows para ambos lenguajes.

En el capítulo 4, se presentan las principales fases requeridas para desarrollar aplicaciones sobre transputers, desde el análisis de requerimiento y especificación hasta su diseño, implementación y prueba. El objetivo de este capítulo es presentar las bases, tanto abstractas como concretas, para desarrollar aplicaciones en ambos lenguajes. Esto se logra mediante la implementación de una simple aplicación, la cual incluye desde los pasos necesarios para la configuración de la tarjeta B008 hasta la ejecución y depuración de la misma.

El capítulo 5 es la parte central de este trabajo, pues es aquí donde se desarrollan las aplicaciones de optimización de estructuras y de proceso de imágenes. Se muestra todo el proceso que cualquier aplicación requiere para su paralelización, desde el planteamiento del problema y su diseño hasta la presentación de una parte de su codificación y de los resultados. También se presentan los resultados obtenidos en cada una de las aplicaciones.

Por último, en el capítulo 6 se presentan las conclusiones obtenidas.

3. Presentación de los Transputers

En este capítulo se presentan las herramientas que serán usadas para resolver las aplicaciones de optimización de estructuras y de proceso de imágenes de este trabajo sobre un sistema de transputers. También se presentan las características de los transputers y de la tarjeta B008 sobre la que se montan; así como las instrucciones requeridas para la instalación de las herramientas que serán usadas para crear las aplicaciones tanto en Occam como en lenguaje C, incluyendo una herramienta para la depuración desde Windows para ambos lenguajes.

3.1. Los Transputers y la Tarjeta IMS B008

El transputer es un dispositivo VLSI que contiene: un chip-procesador, memoria y varias ligas de comunicación serial bi-direccionales e independientes. Su manufacturación la realiza INMOS, una división de SGS-Thomson Microeletronics. El transputer fue el primer microprocesador de 32 bits diseñado para operar en paralelo, y a través de sus ligas de comunicación y capacidades de multiprocesamiento, para facilitar el supercómputo en paralelo.

En los últimos modelos de transputers, cada transputer se coloca en una pequeña tarjeta, llamada TRAM (TRAnsputer Module), con un máximo de 16 megabytes de memoria adicional (dentro del TRAM y externa al transputer) y una unidad para realizar operaciones de punto flotante. La finalidad de poner un transputer sobre un TRAM es poder conectar este transputer con otros, a través de sus ligas, mediante un programa. Uno de los modelos más populares de TRAM tiene un procesador INMOS T-805 de 30MHz y 32 bits, 4 megabytes de RAM, unidad de punto flotante, 4 Kbytes de memoria interna y 4 ligas de comunicación (aproximadamente equivalente a un microprocesador INTEL 486 a 33 MHz).

Para obtener un sistema con más de un transputer, hay *tarjetas* que pueden integrar varios TRAMs; éstas se encuentran disponibles para PCs y estaciones de trabajo SUN. Por ejemplo, la tarjeta INMOS B008 es un tipo estándar para PC que agrupa un máximo de 10 TRAMs T-805. Además, existen cajas de expansión VMX en las que pueden ser integradas tantas tarjetas B008 como número de transputers se desee tener.

La arquitectura del transputer fue diseñada en base a los conceptos de concurrencia y comunicación de Occam, de manera que se puede establecer una corres-

pondencia entre el equipo de cómputo disponible y la estructura lógica de un programa. Occam es a su vez, un lenguage de procesamiento en paralelo basado en el modelo *CSP* (communicating secuencial processes) de Hoare [9]. En este modelo, cada programa está estructurado en un conjunto de procesos concurrentes que se comunican a través de instrucciones de Entrada/Salida, sobre canales punto-a-punto, en forma síncrona. Cada proceso, a su vez, puede estar estructurado como un conjunto de procesos comunicándose concurrentemente. Por lo tanto, cada programa *Occam* tiene una estructura jerárquica-paralela, tal como se muestra en la siguiente figura,

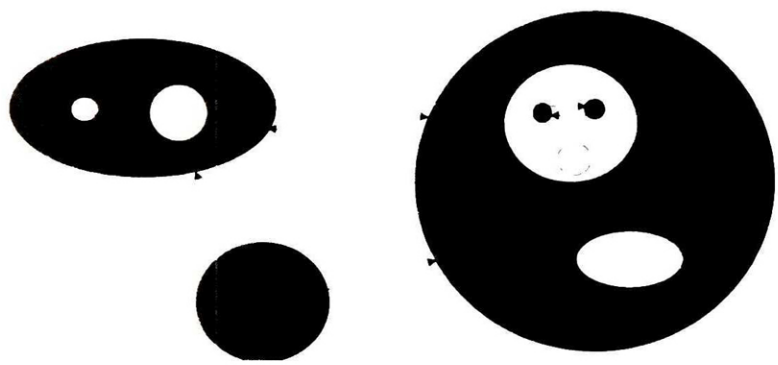


Figura 3.1: `Modelo CSP para procesamiento en paralelo.`

El modelo *CSP* es un método formal con el cual se pueden describir sistemas; un sistema está constituido por procesos. Cualquier proceso se puede descomponer en procesos base llamados primitivos (apéndice A).

El Transputer fue diseñado de manera que:

- Los procesos pudieran fácilmente ser cargados e inicializados independientemente del nodo en donde residan sobre la red.
- Los mensajes puedan fácilmente ser enviados y recibidos:
  - entre dos procesos que se encuentran sobre el mismo nodo de procesamiento.

- entre un proceso de algún nodo de procesamiento y el *programa supervisor central*[1]
- entre el programa supervisor central y la computadora sede (la PC).

- Fuera posible parar los procesos de cada nodo, con lo cuál se pudieran realizar, principalmente, las tareas de depuración.

Con este diseño, los transputers pueden ser elegidos de entre otros sistemas paralelos para realizar la aplicación de interés, de acuerdo a sus características, las cuales se enuncian enseguida.

Características de los transputers

Los transputers poseen una serie de características relacionadas específicamente con su arquitectura de diseño; éstas se presentan a continuación:

- Da un rendimiento de *30 MIPS* (millones de intrucciones por segundo).
- Tiene una unidad para realizar operaciones de punto flotante, llamada *FPU*, de 64 bits, logrando *4.3 MFLOPS* (millones de operaciones de punto flotante por segundo).
- Maneja una cola de instrucciones capaz de procesar 4 instrucciones simultáneamente.
- Tiene una memoria de 4 kilobytes interna al transputer para una rápida ejecución de las instrucciones.
- Permite usar memoria externa al transputer, pero interna al TRAM, hasta de 4 gigabytes.
- Usa dos relojes, uno de 1 μs (microsegundos) y otro de 64 μs, para manejar dos niveles de prioridad para la ejecución de procesos.
- Cambio de contexto entre procesos de *30 ns (*nanosegundos*)*.
- Reloj externo de 5MHz para reiniciar el transputer y/o sus ligas.
- 4 ligas bidireccionales seriales de 20 Mbits/s.

[1] Se le llama *programa supervisor central* al programa correspondiente al proceso que establece la comunicación entre la computadora sede y el resto de los procesos.

- Ruteador "*IMS COO4*" programable de 32 entradas en 32 salidas para configurar por software la red de transputers.

Con estas características, un sistema de transputers puede ser comparado con otros sistemas paralelos para tomar la decisión sobre cual sistema adquirir y/o usar.

Otra cuestión importante en la elección de un sistema paralelo, es el modelo de programación que éste usa. En el caso particular de un sistema de transputers, el modelo de programación usado se muestra a continuación.

3.2. Modelo de Programación

Las aplicaciones para transputers se codifican usando procesos de acuerdo al modelo CSP; principalmente en los lenguages *ANSI C*, *C++* y *OCCAM*, los cuales tienen facilidades para mezclado de lenguajes e inserción de instrucciones en ensamblador.

Los archivos que representan a los procesos, se compilan y enlazan con las funciones de las bibliotecas que se usan, para formar archivos llamados *unidades enlazadas*. Sobre un transputer pueden ser ejecutadas una o más unidades enlazadas.

Una vez que se tienen todas las unidades enlazadas, es necesario especificar cuales unidades serán ejecutadas sobre tal transputer, lo cual se consigue mediante un archivo, llamado *descripción de configuración*, que consta de tres partes: *descripción de hardware* (aquí se describe el número de transputers y la topología en la que se conectan mediante sus ligas), *descripción de software* (aquí se especifican los procesos y los canales que usados por la aplicación) y *asignación de unidades lógicas a físicas* (aquí se asignan los procesos a los procesadores y los canales a las ligas de los tranputers).

Las unidades enlazadas y el archivo de descripción de configuración compilado, se ponen en un solo archivo, el cual representa a la aplicación, y puede ser ejecutado sobre los transputers.

Cada una de estas etapas requeridas para la creación de una aplicación sobre los transputers, pueden ser fácilmente visualizadas en la figura 3.2.

Una vez que se tienen ubicadas las herramientas que serán usadas para desarrollar aplicaciones sobre transputers, es necesario instalar cada una de éstas, lo cual se cual se puede lograr siguiendo las instrucciones de la siguiente sección.

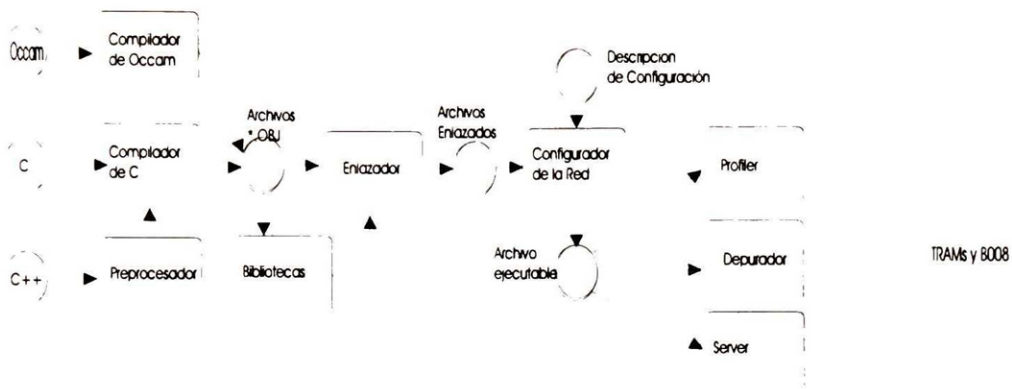


Figura 3.2: `Etapas necesarias para la creación de aplicaciones sobre transputers.`

3.3. Instalación del Software

En esta sección se dan los pasos necesarios para realizar una instalación completa de todas las herramientas disponibles para desarrollar aplicaciones sobre los transputers.

Las herramientas que se tienen disponibles son las siguientes:

- *ANSI C Toolset.* Serie de cuatro discos enumerados y etiquetados como *IMS D7314A.*
- *Occam Toolset.* Serie de cuatro discos enumerados y etiquetados como *IMS D7305A.*
- *PC INQUEST.* Serie de cuatro discos enumerados y etiquetados como *IMS D7300A.*
- *Software MMS para Configurar la Tarjeta B008.* Consta de un solo disco etiquetado como *IMS S708B.*

Para instalar estas herramientas es necesario tener un equipo de cómputo con ciertas características que a continuación se enuncian.

Prerrequisitos de Instalación

Los siguientes prerrequisitos son necesarios para la instalación de cada sistema de herramientas:

- Una IBM PC 386, 486 ó 586 (o compatible) con MS-DOS (ver. 5.0 o posterior) y Windows 3.1 (Windows para grupos o Windows95) en modo mejorado.
- Acceso a drive de $3\frac{1}{2}$ de alta densidad.
- Al menos 4 MBytes y preferiblemente 8 MBytes de memoria.
- Al menos 20 MBytes de espacio de disco: 8.0 MBytes para el compilador de C (herramientas D7314A), 8.3 para Occam (herramientas D7305A), 3.0 para INQUEST (herramientas D7300A) y 0.7 para el software MMS (herramientas S708B).
- Para instalar INMOS INQUEST es necesario haber instalado previamente INMOS ANSI C toolset IMS D7305 ó INMOS OCCAM2 toolset IMS D7305.
- Una tarjeta IMS B004 ó IMS B008 y TRAMs ó una tarjeta compatible conteniendo (o conectada a) transputers IMS T225, T400, T425, T426, T801, ó T805.

Una vez que se tiene un equipo de cómputo con estas características, cada sistema de herramientas puede ser instalada. Los pasos necesarios se presentan a continuación:

Instalando el Software: *D7314A ANSI C Toolset*

Para instalar ANSI C, insertar el disco 1 en el *drive*. Enseguida, ejecutar el archivo *install.bat* dando como parámetros la letra del *drive* sobre el cual está ubicado el disco 1, y la letra del drive sobre el cual se desea instalar ANSI C.

Por ejemplo, si el drive fuente es A y el drive destino es C, escribir:

install a c

posteriormente seguir las indicaciones que este programa de instalación requiera.

La instalación crea el directorio *\D7314A*, en el cual todas las componentes se copian en subdirectorios.

Si se tiene algún problema con la instalación, consultar en [14] el cap. 2.

Instalando el Software: *D7305A Occam Toolset*

Para instalar Occam Toolset, se sigue el mismo procedimiento dado para instalar C Toolset.

La instalación crea el directorio *\D7305A*. Para mayores detalles acerca de la instalación, consultar en [18] el cap. 2.

Instalando el software: *D7300A INQUEST*

Para instalar INQUEST sobre una PC se elige la opción "Run..." en el menu "File" desde el "Program Manager" en Windows. Introducir lo siguiente:

a:setup

seguir las instrucciones de este programa de instalación durante su ejecución.

Esta instalación crea el directorio *\D7300A* y una estructura de subdirectorios que a continuación se presenta en la siguiente tabla:

Directorio	**Contenido**
\BIN	Herramientas (ejecutables)
\LIBS	Bibliotecas
\EXAMPLES\APP_C	Ejem. en C para depurar
\EXAMPLES\APP_OCC	Ejem. en Occam para depurar
\EXAMPLES\ASERV	Ejem. para ASERV
\EXAMPLES\PIPE	Ejems. para usar *monitor* y *profiler*
\EXAMPLES\DYNLOAD	Ejems. de carga dinámica

Información más detallada puede ser encontrada en [24], en la sección de instalación en los cap. 1 y 2.

Instalando el Sofware MMS (S708) para la Configurar la Tarjeta B008

Copiar toda la estructura de archivos y directorios contenida en el disco S708 a un directorio creado en el disco duro local. Si S708 es este directorio, y si además se está posicionado en él (desde MS DOS), entonces se ejecuta el siguiente comando:

c:\s708\ xcopy a:*.* /s

El contenido del directorio S708 después de la instalación es la siguiente:

Archivos	Contenido o uso
B008	archivo *hardwire*
ISERVER.EXE	copia ejecutable de *iserver*
ISERVER	Directorio que contiene fuentes para *iserver*
MMS2.B4	Software para configurar la tarjeta B008
PCMMS.ITM	Archivo que facilita el uso de MMS
S708DRIV.SYS	Driver en DOS para la tarjeta B008
SOFTWIRE	Ejemplo de una configuración *softwire*

Una vez que se tienen instaladas las herramientas, es necesario realizar ciertas operaciones, algunas de las cuales no se describen en los manuales (por ejemplo, las modificaciones a los archivos *autoexec.bat* y *config.sys*). Tales modificaciones se enuncian en la siguiente sección.

3.4. Notas Importantes Acerca de la Instalación

Para lograr que todos los sistemas funcionen apropiadamente es muy importante realizar las siguientes operaciones:

1. Agregar las siguientes líneas a los archivos:

   - *CONFIG.SYS*

- FILES=30
- SHELL=C:\COMMAND.COM C: /E:1024 /P

- *AUTOEXEC.BAT*
  - REM ********** TRANSPUTERS **********
  - PATH %PATH%;C:\D7300A;C:\D7305A\TOOLS; C:\D7314A\TOOLS
  - SET ICONDB=C:\D7314A\CONNECT\B008M.DAT
  - SET TRANSPUTER=B008
  - SET IBOARDSIZE=#400000
  - SET IDEBUGSIZE=#400000
  - SET ISEARCH=C:\D7305A\LIBS\;C:\D7314A\LIBS\; C:\D7300A\LIBS\
  - SET ITERM=C:\D7314A\ITERMS\PCANSI.ITM
  - SET DOS4G=quiet
  - REM **

2. Copiar ISERVER.NFS a ISERVER.EXE desde DOS ejecutando los siguientes comandos:

   - cd \D7314A\TOOLS
   - copy iserver.nfs iserver.exe

3. Copiar el archivo ISERVER.EXE del directorio \D7314A\TOOLS al directorio \S708, pues el programa contenido en el directorio S708 es una versión no actual.

4. Modificar el archivo B008M.DAT ubicado en el directorio C:\D7314A\CONNECT. Deberá ser cambiada la línea:

   - |B008 |T|localhost|Link1|b004|||Description of board configuration|

   cambiando sólo el campo *link1* por *#150*, quedando la línea de la siguiente manera,

- |B008 |T|localhost|#150|b004|||Description of board configuration|

Una vez instaladas las herramientas que serán usadas en el desarrollo de aplicaciones para transputers, es necesario saber cuales herramientas son las más útiles y cual es la función principal de cada una; ésto será tratado en la siguiente sección.

3.5. Herramientas para Desarrollar Aplicaciones en una Red de Transputers

Esta sección describe brevemente el uso de cada una de las herramientas disponibles para la creación de aplicaciones sobre transputers. Algunas de estas herramientas pueden solamente ser usadas desde Windows. Las restantes desde el sistema operativo DOS.

Herramientas en el Sistema Operativo DOS

Primero serán presentadas las herramientas de cada sistema de programas para sistema operativo DOS.

Herramientas para Configurar la Tarjeta B008 (S708B)

Las siguientes herramientas son usadas para fijar y reconocer la configuración de la tarjeta B008.

- *ISERVER.*- Herramienta usada para cargar y ejecutar programas sobre transputers; además permite la comunicación con la computadora sede (*host*).
- *MMS2.B4.*- Módulo de software de la tarjeta B008 que se utiliza para configurar la topología de conexión de los procesadores.

El uso de estas herramientas se mostra en detalle en [22] y [23].

Herramientas para ANSI C Toolset (D7314A)

Las siguientes herramientas son usadas para desarrollar aplicaciones sobre transputers usando lenguaje C:

- *ICC.*- Compilador estándar de ANSI C con soporte para concurrencia. Genera código objeto para cualquier tipo específico de transputer o clases de transputers.
- *ICCONF.*- Compilador del archivo que contiene la descripción de configuración.
- *ICOLLECT.*- Colector de código, el cual genera archivos de código ejecutables, sobre los transputers. Con esta herramienta, todas las unidades enlazadas que forman la aplicación que se desea implementar, son unidas y puestas en un solo archivo para ser posteriormente cargado y ejecutado mediante *ISERVER.*

- *IDEBUG.*- Depurador[2] de red, que proporciona *depuración interactiva*[3] o *post-mortem*[4] de programas para transputer en ambiente DOS.

- *ILIBR.*- Herramienta usada por el programador para crear bibliotecas de funciones a partir de archivos compilados.

- *ILIST.*- Despliega archivos binarios ejecutables en una forma leíble (es decir, en ensamblador).

- *IMAKEF.*- Herramienta para automatizar la compilación, enlazamiento, configuración y recolección de programas a un ejecutable.

- *IMAP.*- Genera un mapa de la memoria que un programa ejecutable (sobre los transputers) usa.

- *ISERVER.*- Herramienta usada para cargar y ejecutar programas sobre hardware para transputers. Además permite la comunicación con la computadora sede.

- *ISIM.*- Permite simular la ejecución de programas sobre transputers sin que estén presentes.

- *SKIP.* Herramienta que permite ejecutar el *proceso principal*[5] en un procesador diferente del transputer raíz.

---

[2]Un *depurador* para los transputers es un programa que sirve para observar el comportamiento de una aplicación en los transputers y dejarla libre de errores. Esto es, ejecutando instrucción por instrucción de cada proceso que forma parte de la aplicación, y así poder saber los valores de cualquier estructura de datos (o variable) en cualquier momento.

[3]Depuración *Interactiva.* Es una de las dos formas de depurar un programa sobre los transputers, iniciando cada uno de los procesos y deteniéndolos en la primera instrucción. Ejecutando instrucción por instrucción de cada proceso que forma parte de la aplicación, se puede conocer el contenido de cualquier estructura de datos (o variable) en cualquier momento; permitiendo también realizar las operaciones clásicas de un depurador y algunas otras más: *break, watch, print, interrupt*, manipulación de procesos, hilos, canales, visualización del *stack*, etc.

[4]Depuración *postmortem.* Es otra de las formas de depurar un programa sobre los transputers. Después de que un programa termina debido a alguna falla de programación, son guardados ciertos datos que sostienen la información necesaria para realizar la depuración *interactiva* en el estado antes de haber ocurrido el error.

[5]Un *proceso principal* es aquel que establece la comunicación entre la computadora huésped y el resto de los procesos que forman la aplicación sobre los transputers. Este proceso, si existe, debe ser ejecutado en el transputer raíz (el transputer que está directamente conectado a la computadora sede a través de su liga 0).

La descripción completa de cada una de estas herramientas se encuentra en [19].

Herramientas para D7305A Occam Toolset

Para occam existen exactamente las mismas herramientas que existen para C, inclusive con el mismo nombre, a excepción de IC e ICCONF, el compilador y el configurador respectivamente, que ahora tienen una pequeña variación en su nombre: OC y OCCONF. Cada una de estas herramientas se describe en [15].

Herramientas en Windows: *D7300A INQUEST*

Las siguientes herramientas son auxiliares, pero muy utiles en el desarrollo de aplicaciones sobre transputers (principalmente la herramienta para depuración):

- *INQUEST*.- Esta herramienta permite la realizar una depuración post-mortem.
- *IMON*.- Programa para monitorear el tiempo de uso de los procesadores.
- *PROFILER*.-Esta herramienta sirve para realizar un análisis estadístico de tiempo y memoria para cada función, procedimiento y proceso.
- *IRUN*. Programa (en windows) que puede realizar las mismas operaciones que *ISERVER* (en *DOS)*. Además, permite realizar la depuración interactiva.
- *RSPY*.- Analizador de la red mediante la conexión de los switches del croosbar *COO4*. También permite crear la descripción de hardware (recordemos que ésta es una de las tres partes que debe contener el archivo de configuración) para *Occam* y *C* .

Información más detallada de cada una de estas herramientas de INQUEST puede ser revisada en [24].

De todas las herramientas presentes, hay una de mayor interés, ésta es *ISERVER*, programa que se ejecuta sobre la computadora sede para establecer la comunicación con el transputer raíz, y a través de éste, con el resto de la red de transputers. Esta comunicación se consegue asignando dos canales sobre la liga 0 del transputer raíz, la cuál está conectada a la computadora sede. El programa

ISERVER se usa, principalmente, para cargar y ejecutar programas-transputer a una red (de transputers). Además, se usa conjuntamente con el software *MMS* para configurar dicha red. Este programa mantiene su ejecución sobre la computadora sede mientras el programa-transputer esté activo en los transputers. La relación entre el programa-transputer y el *ISERVER* es siempre de tipo amo-esclavo. El uso de esta herramienta se ilustra en la figura 3.3.

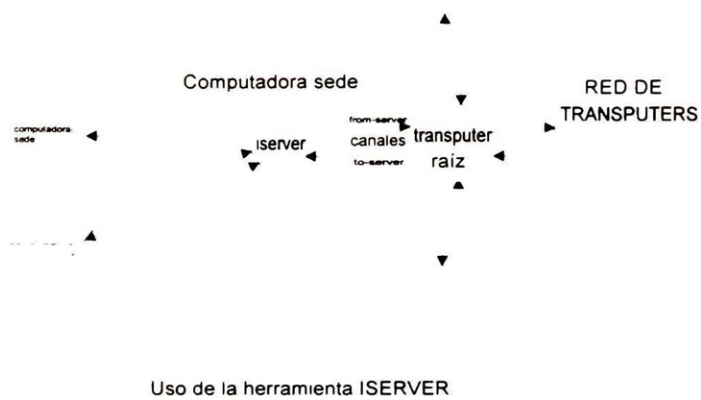


Figura 3.3: `Uso de la herramienta ISERVER.`

El uso de cada una de las herramientas quedará completamente claro mediante la creación de una aplicación en ambos lenguajes, Occam y C, para resolver el mismo problema (que consiste en realizar una simple operación aritmética en cada proceso, a partir de dos números recibidos), a través de las secciones 4.3 y 4.4.

En las proximas dos secciones se presentan las principales características tanto de C como de Occam, con la finalidad de orientar al programador en la selección del lenguaje para la creación de su aplicación de interés.

3.6. Características Generales de *Occam*

En esta sección se presentan las características de Occam, con el fin de que el programador interesado en desarrollar aplicaciones sobre transputers tenga una idea de lo que es Occam y lo compare con el lenguaje C, a través de la siguiente sección, y que con ésto se pueda elegir el lenguaje más conveniente.

El lenguaje Occam tiene las siguientes características:

- Es el lenguaje óptimo para desarrollar aplicaciones sobre transputers desde el punto de vista del hardware.
- Es un lenguaje de alto nivel usado en aplicaciones de control en tiempo real, procesamiento de imágenes, comunicaciones, etc.
- Fue diseñado para expresar algoritmos concurrentes y ejecutarlos en uno o varios procesadores.
- Describe una aplicación como una serie de procesos que se comunican vía canales (modelo *CSP*). Ésto facilita la definición de un programa, y permite que un sistema con multiples funciones sea construido en base a un diseño modular. Un conjunto de procesos interactuando puede ser agrupado como un solo programa para propósitos de desarrollo, permitiendo a los programadores trabajar sobre pequeñas secciones manejables de un sistema.
- Proporciona un alto grado de seguridad (no disponible en C), debido a su diseño basado en métodos formales y técnicas de prueba matemática. Occam soporta programación concurrente estructurada e incluye una sofisticada verificación de errores al tiempo de compilación, reduciendo el tiempo de ejecución.
- Permite combinar código de *C* y ensamblador.
- No tiene reglas de precedencia, por lo cuál se tienen que usar paréntesis para definir el orden en que se realizará cada operación dentro de una expresión. Obviamente resulta muy engorroso escribir expresiones.
- No se tienen estructuras de datos equivalentes a las existentes en C.

La descripción de Occam será tratada en el apéndice A.

Para que el programador pueda elegir entre Occam y C durante la etapa de implementación de la aplicación de interés, es necesario comparar estos lenguajes, lo cual puede ser conseguido mediante la presente sección y la siguiente.

3.7. Características Generales de *C*

En esta breve sección se presentan las características del lenguaje C para que, en conjunción con la sección anterior, pueda ser comparado con Occam.

El lenguaje C tiene las siguientes características:

- Totalmente compatible con ANSI C.
- El procesamiento en paralelo está disponible mediante el uso de bibliotecas extras. Sus funciones permiten definir y crear procesos, comunicar uno con otro vía canales y su sincronización usando semáforos.
- Permite reusar código, reestructurándolo en procesos.
- Permite combinar código de Occam y ensamblador.
- Se tiene la ventaja de aprovechar toda la potencialidad de C.
- No se tiene tanta seguridad, como en Occam, para la verificación de errores durante la compilación del programa; así como tampoco se tiene en el manejo de errores durante su ejecución.

La descripción del lenguaje C, específicamente lo relacionado con el paralelismo, será tratada en el apéndice B.

En estas últimas secciones se presentaron las principales características tanto de C como de Occam, con lo cual el programador puede elegir el lenguaje más conveniente para la creación de su aplicación de interés.

Además de codificar la aplicación en términos de procesos, es necesario definir la forma en la que éstos serán ejecutados en la red de transputers mediante una serie de instrucciones especiales dadas a través de un archivo llamado *descripción de configuración*, el cual se describe brevemente en la siguiente sección.

3.8. Descripción de la Configuración Adherida a toda Aplicación

Independientemente del lenguaje de programación, en toda aplicación sobre una red de transputers existe una parte de configuración, en la cual, se debe especificar el número de procesadores a usar y la topología en la que serán conectados (esta debe ser la misma red, o una subred, correspondiente a la red dada por la configuración de la tarjeta usando el software *MMS*); en esta configuración se debe especificar también los procesos y canales con los cuales está diseñada la aplicación; y por último, estos procesos y canales deben ser asignados a la red de procesadores y ligas. Así, toda especificación de configuración esta formada por tres partes:

1. Descripción de *Hardware*.
2. Descripción de *Software*.
3. Asignación de procesos y canales a procesadores y ligas.

Por razones de espacio, no incluiremos el detalle de este lenguaje especial (el cual tiene la misma sintaxis del lenguaje en el que se codifica la aplicación. Por ejemplo, si la aplicación se codifica en C entonces la descripción de configuración se escribe de acuerdo a la sintaxis de C). Los detalles completos de la descripción de la configuración pueden ser consultados en el capítulo 6 de [21] y [17] para Occam y C respectivamente. Únicamente nos limitaremos a mostrar la descripción de configuración de una misma aplicación para ambos lenguajes, C y Occam, que será presentado en secciones posteriores 4.3 y 4.4.

Durante la creación de una aplicación varios lenguajes de programación pueden ser combinados. Los métodos necesarios para realizar dicha combinación se presentan en la siguiente sección.

3.9. Combinando Varios Lenguajes en una Aplicación

Para muchas aplicaciones es necesario crear programas usando más de un lenguaje de programación. Por ejemplo, un algoritmo puede ser expresado de una forma más simple en un lenguaje específico, o usando módulos de aplicación ya existentes.

Cada conjunto de herramientas proporciona una base simple y clara para combinar los diferentes lenguajes (C, Occam y ensamblador). En particular, las herramientas para C y para Occam permiten esta combinación. Los procesos independientes (es decir, procesos correspondientes a unidades enlazadas), pueden ser

escritos ya sea, en diferentes lenguajes, y ser compilados y enlazados usando un conjunto de herramientas común; o bien, usando módulos enlazados ya existentes. Estos procesos independientes pueden ser puestos (y ejecutados posteriormente) en cualquier parte de la red de transputers a través de la descripción de configuración.

Para combinar *Occam* y *C* existen ciertas consideraciones que deben ser tomadas en cuenta. Éstas son las siguientes:

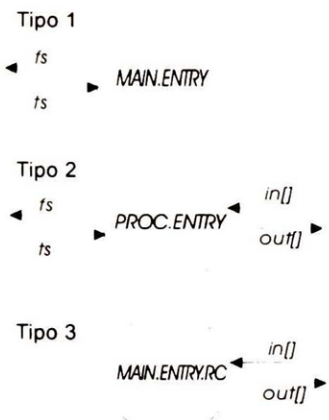
- Declarar rutinas externas.
- Traducir nombres de acuerdo a la sintaxis del lenguaje, con la finalidad de preservar la consistencia de nombres. Por ejemplo, el símbolo "_" se usa por C (y no por Occam) en nombres de variables y funciones (la función *Calcula_f( )* usa este símbolo en C); mientras que en Occam (y no en C) el símbolo "." se usa para lo mismo (la función *Calcula.f( )* usa este símbolo en Occam).
- Seguir las convenciones de llamada a rutinas o funciones del otro lenguage; esto es, realizar el paso de parámetros por valor, especificar la dirección de memoria estática y hacer que los tipos de parámetros coincidan.
- Cuidar la compatibilidad del tipo de dato regresado por la función o rutina.
- Realizar adecuadamente el manejo de la memria estática (*C* usa un área de memoria llamada *GSB (Global Static Base)* para datos estáticos).
- Especificar de manera adecuada las bibliotecas durante la fase de enlazamiento.

La combinación de lenguajes más común en el desarrollo de aplicaciones se da entre C y Occam. Y particularmente entre estas dos combinaciones, la más apropiada se da cuando el lenguaje C se usa desde Occam para implementar el código de los procesos (por la razón de que C es el lenguaje de mayor dominio público), y cuando Occam se usa solamente para realizar las tareas específicas de paralelismo. A continuación se comentan a nivel muy general los métodos para lograr esta combinación.

Usando C desde Occam

Existen dos métodos para tal efecto:

1. Usando procesos (mediante la biblioteca *entry.lib*) de tipo 1, 2 y 3:



fs, ts (from server, to server) son los canales de comunicación con la computadora sede
in[ ], out[ ] son arreglos de canales que el proceso en C usa para comunicarse con otros procesos en el programa

Figura 3.4: `Tipos de procesos.`

- Tipo 1.- Este tipo de proceso se usa cuando la aplicación completa, que incluye a este proceso, se diseña para que su ejecución sólo sea posible en un solo transputer. En este caso, este proceso sólo puede comunicarse con la computadora sede.
- Tipo 2.- Este tipo de proceso se usa cuando se necesita de la comunicación con otros procesos, así como con la computadora sede.
- Tipo 3.- Este tipo de proceso es similar al del tipo 2 excepto que no tiene acceso a la computadora sede.

2. Usando solamente funciones (a través de la biblioteca *callc.lib*).
   Los programas en C hacen uso de la memoria estática, y de la memoria *heap*. Sin embargo, Occam no hace uso de este tipo de memoria, de manera que debe ser explícitamente reservada por el programa en Occam antes de las llamadas de las funciones en C. Para ésto, existen cuatro rutinas en la biblioteca *callc.lib* de Occam que se usan para fijar y terminar las áreas de memoria estática y *heap* que las funciones de C requieren.

Una vez instaladas todas las herramientas disponibles, y descritas en términos generales, es necesario pasar al uso de éstas, lo cual se llevará a cabo con el desarrollo del siguiente capítulo.

4. Desarrollando Software en Paralelo

Como en su contraparte secuencial, el software en paralelo se desarrolla en una serie de etapas que comprenden desde el análisis de requerimiento y especificación hasta su diseño, implementación y prueba. Sin embargo, aunque estas fases tienen los mismos objetivos correspondientes a las fases del software secuencial. éstas tienen características nuevas, pues se presentan otros problemas que no se encuentran en el desarrollo del software secuencial. En este capítulo, se presentan estas fases.

El objetivo de este capítulo es presentar las bases, tanto abstractas como concretas, para desarrollar aplicaciones en ambos lenguajes. Esto se logra mediante la implementación de una simple aplicación, la cual incluye desde los pasos necesarios para la configuración de la tarjeta B008 hasta la ejecución y depuración de la misma.

4.1. Introducción a la Computación en Paralelo

En esta sección se presenta una introducción a nivel general de lo que es la computación en paralelo, la cual servirá de base para desarrollar las aplicaciones de interés y realizar también su respectivo análisis de tiempo.

4.1.1. Fases de Diseño

El objetivo básico del diseño de un sistema de software es la identificación de sus módulos lógicos, las funcionalidades que cada módulo debe tener, los datos que cada módulo debe manejar, la forma en la cual los módulos interactúan y la interfaz de uno con otro. En sistemas de software en paralelo cada módulo se asigna a un proceso que a su vez puede ser asignado con uno de los procesadores del sistema. Así, el diseño de software en paralelo se realiza de acuerdo a las siguientes fases:

1. *Descomposición funcional.*- En esta fase es necesario encontrar las diferentes funcionalidades del problema e identificar el conjunto de módulos lógicos y los datos que cada uno requiere.

2. *Partición.*- En esta parte se define el mapeo de los módulos lógicos y datos (los cuales reflejan el punto de vista de la aplicación) sobre un conjunto de procesos y archivos de datos (los cuales reflejan el punto de vista del diseño).

3. *Localización.*- Una vez que la parte lógica ha sido definida, cada proceso y archivo de datos (unidades lógicas) deben ser asignados a uno o más procesadores del sistema (unidades físicas). Para resolver este problema de localización es recomendable realizar las siguientes fases:

    1. *Construcción de la gráfica de comunicación,* en donde los nodos son procesadores y los arcos son las líneas de comunicación entre los procesadores.
    2. *Elección de la topología de interconexión de red.*
    3. *Asignación de los elementos lógicos de la gráfica de comunicación sobre el sistema de procesadores.*

4. *Escalabilidad.*- Esta fase tiene la finalidad de encontrar el número óptimo de procesadores para obtener el mejor rendimiento de la aplicación. Es importante para ello, definir la aplicación en términos del número de procesadores, sólo así tiene sentido hablar de esta fase.

Es importante notar que cada una de estas fases están estrechamente relacionadas, por lo que cualquier variación en una de ellas afectará a las otras tres, y al rendimiento de la aplicación.

4.1.2. Estrategias para Explotar el Paralelismo

Durante la fase de partición pueden ser adoptadas diferentes estrategias para extraer el paralelismo del algoritmo con el que se pretende resolver la aplicación de interés. A continuación se presentan cuatro de ellas.

1. Paralelismo *amo-esclavo.* En la mayoría de los problemas reales un mismo tipo de proceso realiza una tarea sobre un conjunto de datos diferente. Si cada proceso se asigna a un solo procesador, cada procesador ejecuta el mismo programa (aunque con diferentes ramas de datos), con una actividad de comunicación solamente requerida para recoger los resultados del procesamiento. En una estructura *amo-esclavo*, un proceso *amo* envía conjuntos de datos independientes a los procesos restantes (*esclavos*). La ventaja

fundamental de esta estructura es su simplicidad y su equilibrio de carga computacional. Sin embargo, si el número de procesos esclavos es muy grande, la sobrecarga del proceso *amo* puede incrementarse de manera que el rendimiento total del sistema puede disminuir.

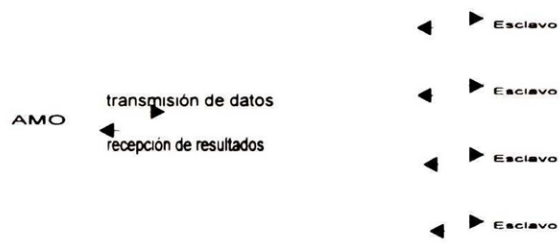


Figura 4.1: `Paralelismo maestro-esclavo`

2. Paralelismo *geométrico*. Muchos de los problemas físicos tienen una estructura geométrica regular (por ejemplo, en el problema de procesamiento de imágenes, autómatas celulares, redes neuronales, simulación de conducción de calor, etc.), en la cual cada elemento (o dato) del dominio del problema interacciona con sus vecinos. En estos problemas, su dominio puede ser dividido en varias regiones (lo cual define el *grano* del sistema), las cuales pueden ser procesadas independientemente una de otra, siendo cada una asignada a un procesador diferente de acuerdo a la fase de la descomposición funcional. Así, cada procesador actualiza su región e intercambia los valores de los datos en sus bordes para ser procesados (recordemos que estos dependen de sus vecinos). El paralelismo geométrico es uno de los más simples y una de las técnicas más eficientes para desarrollar aplicaciones de procesamiento en paralelo. Por razones de eficiencia, la elección de la topología de la red de cómputo debe reflejar los patrones de comunicación entre los elementos

(datos) asignados a cada procesador. Por ejemplo, si en el problema de procesamiento de imágenes se elige la vecindad de los cuatro vecinos más cercanos entonces una topología de tipo malla es la más apropiada para transmitir y recibir información entre los procesadores.

3. Paralelismo *algorítmico*. Este tipo de paralelismo puede ser explotado por la partición del algoritmo en un cierto número de procesos. Cada uno de estos procesos se ejecuta completamente en paralelo sobre sus propios datos o sobre datos recibidos de otros procesos. La estructura de los procesadores se define por el flujo de datos entre los procesos, y obviamente, depende altamente del algoritmo usado en particular.

4. Métodos *híbridos*. Los tres tipos de algoritmos anteriores no son mutuamente excluyentes y pueden ser combinados para lograr un paralelismo óptimo.

Descripciones más detalladas de estos métodos o estrategias para explotar el paralelismo pueden ser encontrados en [8] cap. 3.1. También pueden ser revisados algunos ejemplos para cada método en esta referencia.

Mas adelante veremos como el paralelismo *amo-esclavo* se usa en el diseño de la primera de la aplicaciones de esta tesis (optimización de estructuras). También veremos como el paralelismo *geométrico* y el paralelismo *amo-esclavo* se combinan para explotar el paralelismo en la segunda aplicación (un filtro para eliminar ruido en imágenes).

4.1.3. Fase de Prueba

Un punto crucial en el desarrollo de aplicaciones de procesamiento en paralelo es la fase de prueba. En esta fase, el software implementado se analiza con la finalidad de verificar si cumple con los requerimientos, tanto de ausencia de errores como con otras características que este debe poseer (rendimiento, confiabilidad, crecimiento, etc.).

Los dos enfoques fundamentales para la prueba de software son el estático y el dinámico. En el primer caso, el análisis no require de la ejecución del programa, en lugar de ello todas las trayectorias de ejecución se examinan para detectar posibles causas de error, tales como abrazos mortales (deadlocks). En la literatura existen varios métodos para realizar este análisis, entre los cuales se incluye la construcción de una red de Petri. Esta prueba estática de software es actualmente un tema de investigación.

En la prueba dinámica del software, el programa se ejecuta bajo un número de entradas elegidas cuidadosamente de manera que representen casos de prueba relevantes que deberán ser capaces de activar todas las posibles interacciones entre los procesos. Durante la ejecución de prueba, el programa se monitorea con la finalidad de detectar anomalías y de obtener índices de medición del rendimiento. Sin embargo, el objetivo principal de esta prueba dinámica es detectar los posibles errores del programa, los cuales hacen que el sistema tenga un comportamiento diferente del que se supone debe tener.

4.1.4. Análisis de Rendimiento

El análisis del rendimiento de un sistema paralelo es actualmente uno de los muchos problemas abiertos en las ciencias de la computación, y además, uno de los problemas más complejos. Aún en esta situación, pueden ser evaluados algunos parámetros que miden el rendimiento total del sistema. Los índices que se usan más ampliamente son la *velocidad* y la *eficiencia*. La *velocidad* o *razón de cambio* de un sistema secuencial a un sistema en paralelo, se define como la razón entre el tiempo de respuesta para un problema usando un solo procesador y el tiempo de respuesta cuando se usan N procesadores para resolver el mismo problema. Si denotamos como T_{Sec} el primero de estos dos tiempos, y como $T_{Par}(N)$ para el segundo tiempo, dependiendo este último del número de procesadores N. Obviamente,

$$T_{Sec} = T_{Par}(1)$$

Usando esta notación, la *velocidad* $S(N)$ para un sistema en paralelo puede ser definida como

$$S(N) = \frac{T_{Sec}}{T_{Par}(N)}$$

La eficiencia ε se define en términos de $S(N)$ como

$$\varepsilon = \frac{S(N)}{N}$$

Esta definición presupone que los procesadores son el recurso crítico del sistema en paralelo (ignorando por supuesto la memoria, los canales de comunicación, etc.).

Cuando se desea obtener un sistema en paralelo con un gran número de procesadores, lo que realmente importa no es que el sistema deba ser *eficiente* sino que

sea capaz de obtener los resultados propuestos: esto es, que el sistema sea *efectivo* según la siguiente definición:

Un sistema se dice que es *efectivo* si es capaz de proporcionar un tiempo de respuesta aceptable usando un número de procesadores menor o igual a aquellos disponibles o a aquellos que pueden ser adquiridos sin exceder los límites de costo establecidos.

Una descripción más detallada de este término (*sistema paralelo efectivo*) puede ser encontrada en [8] cap. 3.4.

Existen varias causas que pueden contribuir a la eficiencia de un sistema; esencialmente éstas son:

1. *Sobrecarga algorítmica.* En algunos problemas, el algoritmo usado para encontrar la solución es óptimo para medios secuenciales, no siendo así para sistemas paralelos, afectando directamente su eficiencia.

2. *Sobrecarga de software.* Cuando en una aplicacion se repiten algunas estructuras de datos, o se introducen mecanismos que aseguran la consistencia de los datos, o se duplican algunas operaciones, la eficiencia del sistema disminuye.

3. *Equilibrio de cargas.* Cuando la carga computacional se subdivide de manera no equitativa y, por consecuencia, en algunos momentos un subconjunto de procesadores pueden estar inactivos, la eficiencia del sistema paralelo disminuye.

4. *Sobrecarga de comunicación.* Uno de los problemas más importantes en un sistema paralelo es la comunicación debido a que ésta puede contribuir considerablemente en el tiempo necesario en encontrar la solución de la aplicación de interés.

5. *Sobrecarga de comunicación externa.* Algunas aplicaciones requieren de comunicaciones externas tales como operaciones de lectura/escritura a disco, despliegues gráficos, lecturas del teclado, u operaciones de lectura/escritura hacia algún dsipositivo de entrada/salida en general. La falta de un canal de entrada/salida para cada nodo de procesamiento puede disminuir la eficiencia del sistema paralelo si esta comunicación es frecuente y con gran flujo de información.

Hasta ahora se han presentado algunas sugerencias, a nivel general, acerca de la creación de aplicaciones en sistemas paralelos que van desde las fases de diseño

hasta la fase de prueba. Todos estos conceptos e ideas serán usados en la creación de las dos aplicaciones de este trabajo. En la siguiente sección de este capítulo, se pasará a la parte práctica y concreta. En primer lugar, se darán los pasos necesarios para fijar o reconocer la topología de conexión entre los transputers de la tarjeta B008.

4.2. Configuración de la Tarjeta *IMS B008*

En esta sección se inicia la parte concreta y práctica de este cápitulo con la configuración de la tarjeta B008.

Hace algunos años, los transputers se montaban sobre tarjetas que se interconectaban unas con otras, manualmente, por medio de ligas de tranputer, y por tanto, toda modificación de la topología requería de nuevas interconexiones. Tan pronto como el elemento *C004* fue introducido, *INMOS* definió una única arquitectura para sus tarjetas, llamada *Module Motherboard Arquitecture (MMA)*, para ser usada por todos los productos de la siguiente generación. La característica fundamental de esta arquitectura es que tienen un soporte de conexión y conmutación de las ligas en todas las tarjetas de esta familia. Así, es posible hacer modificaciones dinámicas para establecer las interconexiones entre los transputers contenidos en cualquier número de tarjetas encadenadas. En la práctica ésto se consegue usando un software especial llamado *MMS (Module Motherboard Software)* capaz de reconocer y fijar la configuración especificada por el usuario a través de un lenguaje especial[6]. En esta especificación de la configuración, sólo pueden ser manejadas las ligas de los transputers conectadas al *C004* (la liga 0 y la liga 3) tal como se muestra en la figura 4.2.

En teoría es posible cambiar la configuración de los IMS C004s mientras un programa está siendo ejecutado sobre un arreglo de TRAMs. La reconfiguración puede ser lograda mediante un protocolo que asegura que ningún mensaje esté siendo transmitido en ese momento. Esto puede ser útil, por ejemplo, en un sistema en el cual se necesita una red específica durante la fase de carga de datos y otra red completamente diferente durante la fase de procesamiento de datos. Aunque esto es posible, su organización no es fácil y deberá ser solamente intentado por usuarios expertos. Además, se tiene muy poca información disponible; ésta se encuentra en [23] pag. 21.

Una consecuencia importante de estas tarjetas que usan el dispositivo *C004*, es que no son capaces de fijar cualquier topología. Su diseño inclinó la balanza a favor de la opción de habilitar tanto como fuera posible una reconfiguración,

[6]En este reporte de tesis sólo se presentarán ejemplos, bien documentados, de lo que es este lenguaje especial, y que corresponden a las configuraciones de las topologías más importantes. Tales ejemplos se muestran en los archivos *hardwire* y *softwire*. La sintaxis de este lenguaje puede ser revisado en [22] a través de los apéndices C y D, y las secciones 4, 5 y 6 del capítulo 4.

de manera que ésta fuera sencilla; por la opción de mantener la complejidad y el costo que implica un nuevo cableado de reconexión.

En esta arquitectura, *MMA*, todos los transputers se conectan directamente por dos de sus ligas en un *pipeline* y las otras dos se conectan al *IMS C004* para ser conmutadas por software (ver la figura 4.2). Los *C004s* contenidos en cada tarjeta (típicamente uno o dos) se controlan por transputers *T225* (también, típicamente uno o dos), dedicados exclusivamente a ésto.

Para fijar o reconocer la topología de conexión de los transputers instalados sobre una tarjeta *B008* es necesario configurar dicha tarjeta. La configuración de esta tarjeta se crea por el software *MMS*, que se ejecuta sobre el transputer *T225* a través del uso del programa *ISERVER*. Esta configuración puede ser guardada en un archivo para después ser cargada antes de la ejecución de un programa. El software *MMS* permite interactuar con la tarjeta *B008* a través de menus. Para esto, son necesarios dos archivos: *"hardwire"* y *"softwire"*; el primero contiene las conexiones físicas entre los *slots*[7] de la tarjeta *B008* y la descripción de cada uno de sus componentes; el segundo contiene las conexiones necesarias para la topología requerida. Estos archivos se encuentran en el directorio *c:\s708* del software *S708* para *MMS*.

Para obtener información más detallada, pueden ser consultados [22] y [23].

4.2.1. Hardwire

Este archivo, conocido como descripción de *hardwire*, describe como definir la configuración de la tarjeta IMS B008. Mediante este archivo se especifica el número y tipo de componentes que la tarjeta IMS B008 tiene. También se especifica como los slots, C004s y edges[8] se conectan sobre la tarjeta. A continuación se muestra un diagrama de lo que representa la configuración descrita en este archivo hardwire, que ahora tiene el nombre de *B008*.

El archivo *B008*[9] correspondiente a la descripción de hardwire se muestra a continuación:

---

[7] Un *slot* es una ranura sobre la tarjeta B008 para colocar un TRAM. La tarjeta B008 tiene diez slots.

[8] Un *edge* es un puerto de comunicación de la tarjeta B008 que tiene como finalidad, establecer una conexión con otra tarjeta B008 o con dispositivos de entrada/salida. La tarjeta B008 tiene diez edges.

[9] La numeración de las líneas del archivo *B008*, incluyendo el punto después del número, no forman parte de este archivo. Éstos sólo fueron colocados para su referencia posterior.

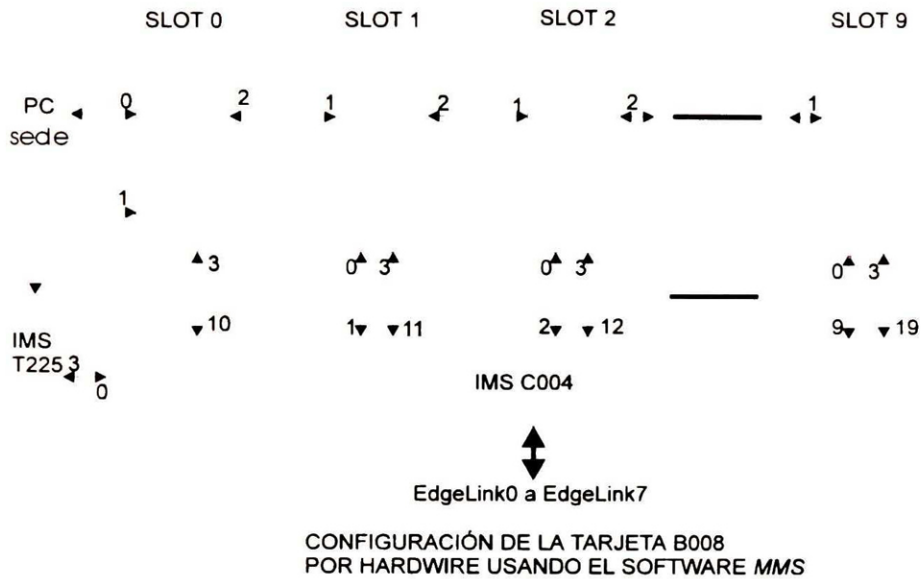


Figura 4.2: Conexión por hardwire

```
01.-/*****************************************/
02.Archivo: B008
03.-/*****************************************/
04.
05.-{{{ Esta instrucción define la tarjeta B008
06.DEF B008
07.
08.  -{{{ Define el número de: transputers T225, elementos
09.  -{{{ de conmutación C004, slots y edges.
10.  SIZES
11.     T2 1 - Un transputer T225
12.     C4 1 - Un croosbar C004
13.     SLOT 10 - Diez ranuras para colocar TRAMs
14.     EDGE 10 - Diez conexiones externas, ya sea a otra tarjeta B008
15.               - o hacia dispositivos de entrada/salida
16.  END
17.  -}}}
```

```
18.
19.  -{{{ Define la conexión entre el transputer T225 y el elemento de
20.  - conmutación C004.
21.  T2CHAIN
22.    T2 0, LINK 3 C4 0 - El transputer 0 ( el único ) T225 se conecta
23.  END                    - a través de su liga 3 al C004 por la liga 0.
24.  -}}}
25.
26.  -{{{ Esta instrucción define las conexiones entre cada uno de los
27.  -{{{ slots que componen la tarjeta B008.
28.  HARDWIRE
29.    -{{{ Estas instrucciones definen las conexiones slot a slot
30.    -usando ligas 2 y 1.
31.    SLOT 0, LINK 2 TO SLOT 1, LINK 1
32.    SLOT 1, LINK 2 TO SLOT 2, LINK 1
33.    SLOT 2, LINK 2 TO SLOT 3, LINK 1
34.    SLOT 3, LINK 2 TO SLOT 4, LINK 1
35.    SLOT 4, LINK 2 TO SLOT 5, LINK 1
36.    SLOT 5, LINK 2 TO SLOT 6, LINK 1
37.    SLOT 6, LINK 2 TO SLOT 7, LINK 1
38.    SLOT 7, LINK 2 TO SLOT 8, LINK 1
39.    SLOT 8, LINK 2 TO SLOT 9, LINK 1
40.    -}}}
41.
42.    -{{{ conexiones de los slots al C004
43.    -{{{ slot 0
44.    C4 0, LINK 10 TO SLOT 0, LINK 3
45.    -}}}
46.    -{{{ slot 1
47.    C4 0, LINK 1 TO SLOT 1, LINK 0
48.    C4 0, LINK 11 TO SLOT 1, LINK 3
49.    -}}}
50.    -{{{ slot 2
51.    C4 0, LINK 2 TO SLOT 2, LINK 0
52.    C4 0, LINK 12 TO SLOT 2, LINK 3
53.    -}}}
54.    -{{{ slot 3
```

```
55.    C4 0, LINK 3 TO SLOT 3, LINK 0
56.    C4 0, LINK 13 TO SLOT 3, LINK 3
57.    -}}}
58.    -{{{ slot 4
59.    C4 0, LINK 4 TO SLOT 4, LINK 0
60.    C4 0, LINK 14 TO SLOT 4, LINK 3
70.    -}}}
71.    -{{{ slot 5
72.    C4 0, LINK 5 TO SLOT 5, LINK 0
73.    C4 0, LINK 15 TO SLOT 5, LINK 3
74.    -}}}
75.    -{{{ slot 6
76.    C4 0, LINK 6 TO SLOT 6, LINK 0
77.    C4 0, LINK 16 TO SLOT 6, LINK 3
78.    -}}}
79.    -{{{ slot 7
80.    C4 0, LINK 7 TO SLOT 7, LINK 0
81.    C4 0, LINK 17 TO SLOT 7, LINK 3
82.    -}}}
83.    -{{{ slot 8
84.    C4 0, LINK 8 TO SLOT 8, LINK 0
85.    C4 0, LINK 18 TO SLOT 8, LINK 3
86.    -}}}
87.    -{{{ slot 9
88.    C4 0, LINK 9 TO SLOT 9, LINK 0
89.    C4 0, LINK 19 TO SLOT 9, LINK 3
90.    -}}}
91.    -}}}
92.
93.    -{{{ Conexiones de los edges del 0 al 9 en el C004
94.    -{{{ a través de las ligas de la 20 a la 29.
95.    C4 0, LINK 20 TO EDGE 0
96.    C4 0, LINK 21 TO EDGE 1
97.    C4 0, LINK 22 TO EDGE 2
98.    C4 0, LINK 23 TO EDGE 3
99.    C4 0, LINK 24 TO EDGE 4
100.    C4 0, LINK 25 TO EDGE 5
```

```
101.      C4 0, LINK 26 TO EDGE 6
102.      C4 0, LINK 27 TO EDGE 7
103.      C40,LINK 28 TO EDGE 8
104.      C40,LINK 29 TO EDGE 9
105.   END
106.   -}}}
107.-}}}
108.PIPE B008 END
```

Los comentarios dentro de este archivo se ponen en líneas después de colocar el signo menos dos veces consecutivas (*-)*. Como se puede observar, la sintaxis de este archivo es muy similar a la de Occam, aunque con instrucciones diferentes. Veamos paso a paso cada una de las instrucciones usadas en este archivo.

La instrucción *DEF* (línea 02) define las componentes de la tarjeta B008 y sus conexiones. Esta instrucción comprende las instrucciones *SIZES*, *T2CHAIN* y *HARDWIRE* (líneas 10, 21 y 28 resp.), cada una de las cuales definen su alcance hasta donde se encuentre una instrucción *END*.

La instrucción *SIZES* (línea 10) define el número de: transputers T225, elementos de conmutación C004, slots y edges a través de las instrucciones *T2*, *C4*, *SLOT* y *EDGE* respectivamente, especificando enseguida el número de cada uno de éstos (líneas 11, 12, 13 y 14 resp.).

Mientras que la instrucción *T2CHAIN* define la conexión entre los transputers T225 y los elementos de conmutación C004. En nuestro caso, la tarjeta B008 sólo tiene un T225 y un C004, los cuales se conectan a través de la liga 3 del transputer y la liga 0 del C004 (línea 22).

La instrucción *HARDWIRE* (línea 28) define las conexiones entre cada uno de los slots, y entre los slots y el elemento de conmutación C004 usando instruciones *SLOT* y *C4*. Por ejemplo, la línea 31 conecta la liga 2 del slot 0 a la liga 1 del slot 1. Y la línea 44 conecta la liga 10 del C004 a la liga 0 del slot 0. Las líneas 31-39 definen la conexión en línea de los slots a través de las ligas 1 y 2, tal como se muestra en la figura 4.2. Las líneas 44-80 definen la conexión entre los slots (a través de las ligas 0 y 3) y el C004.

4.2.2. Softwire

Mediante las órdenes de este archivo, conocido como *softwire*, se define la configuración requerida, usando solamente las ligas 0 y 3 de cada slot, pues las ligas 1

y 2 se usan en el archivo *hardwire* para definir la conexión de los slots en línea (esta forma de conexión se conoce también como *pipe*). Las conexiones de softwire permiten unir un slot a otro (y mediante ésto, se logra también la conexión de las ligas de un transputer con otro al poner los TRAMs sobre los slots), ya sea, dentro de la misma tarjeta o con otra, mediante el uso del software MMS, el cuál conmuta los canales vía el croosbar C004. No es posible hacer cualquier conexión deseada, ya que esto depende de como los slots están físicamente conectados uno a otro, según el archivo hardwire.

La instrucción *SOFTWIRE* se usa para definir la configuración softwire. Mientras que la instrucción PIPE 0 define la configuración de softwire para el pipe 0, es decir, el primer pipe definido por hardwire (en nuestro caso sólo hay uno, pues solamente se tiene una tarjeta B008). La instrucción *SLOT* se usa de la misma forma que en la descripción de hardwire, a excepción de que se usen solamente las ligas 0 y 3 (las conectadas al C004).

A continuación se presentan ejemplos de algunas configuraciones obtenidas cambiando sólo las ordenes dadas en este archivo softwire, es decir, sólo se conmutan las ligas 0 y 3 de cada transputer (asociado a un SLOT).

ESTRELLA (figura 4.3)
SOFTWIRE
PIPE 0
  SLOT 1, LINK 0 TO SLOT 3, LINK 0
  SLOT 1, LINK 3 TO SLOT 0, LINK 3
END

PIPELINE (figura 4.4)
SOFTWIRE
PIPE 0
  - *No es necesaria ninguna especificación pues el pipe*
  - *está dado por hardwire*
END

ANILLO (figura 4.5)
SOFTWIRE
PIPE 0
  SLOT 0, LINK 3 TO SLOT 3, LINK 0

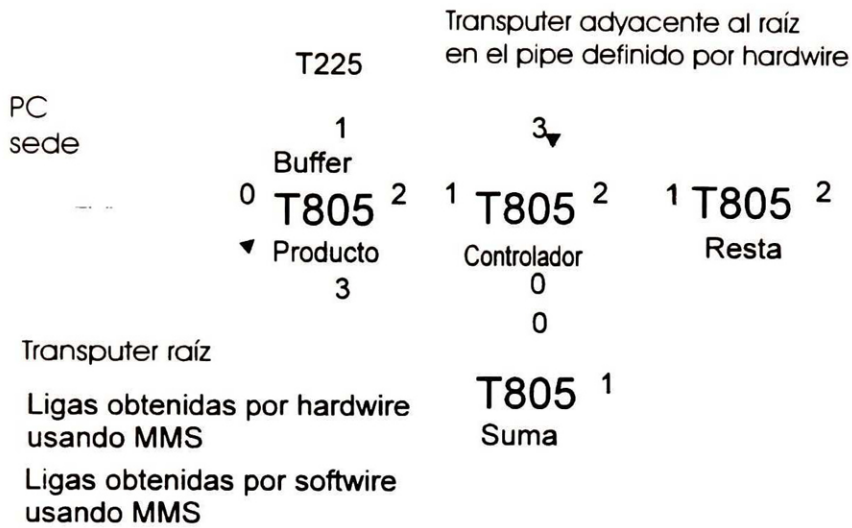


Figura 4.3: `Topología estrella`

END

DOBLE LIGA (figura 4.6)
SOFTWIRE
PIPE 0
SLOT 0, LINK 3 TO SLOT 3, LINK 0
SLOT 1, LINK 0 TO SLOT 2, LINK 0
SLOT 2, LINK 3 TO SLOT 3, LINK 3
END

SEMICOMPLETA1 (figura 4.7)
SOFTWIRE
PIPE 0
SLOT 0, LINK 3 TO SLOT 2, LINK 0
SLOT 1, LINK 0 TO SLOT 2, LINK 3
SLOT 1, LINK 3 TO SLOT 3, LINK 0
END

SEMICOMPLETA2 (figura 4.8)

Figura 4.4: `Topología pipeline`

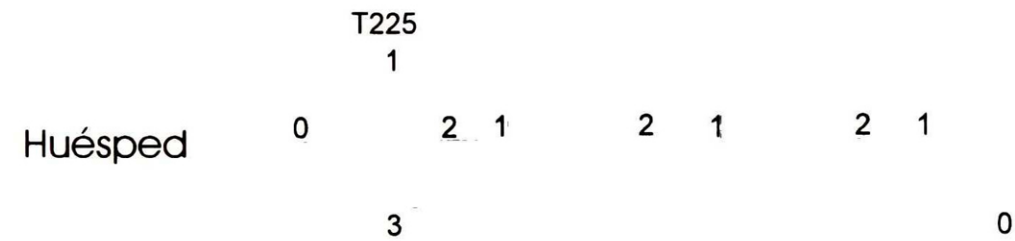


Figura 4.5: `Topología anillo`

```
SOFTWIRE
PIPE 0
  SLOT 0, LINK 3 TO SLOT 2, LINK 0
  SLOT 1, LINK 0 TO SLOT 3, LINK 0
  SLOT 2, LINK 3 TO SLOT 3, LINK 3
END
```

En esta sección se dió una descripción de la tarjeta B008 mediante la cual puede ser configurada. Dos archivos son necesarios para fijar la configuración deseada, *hardwire* y *softwire*, los cuales se pasan como parámetros al programa MMS para que lleve a cabo las instrucciones contenidas en estos archivos.

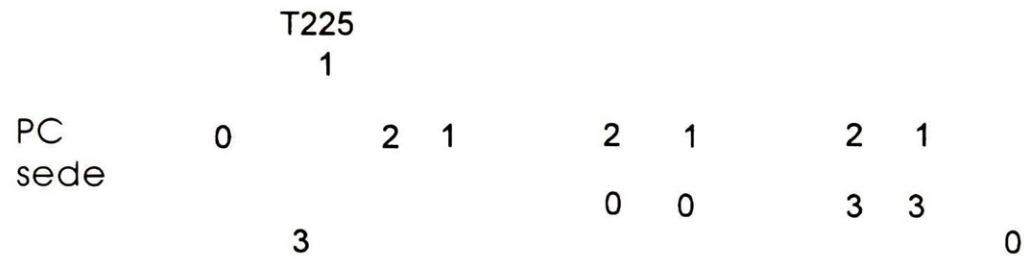


Figura 4.6: `Topología doble liga`

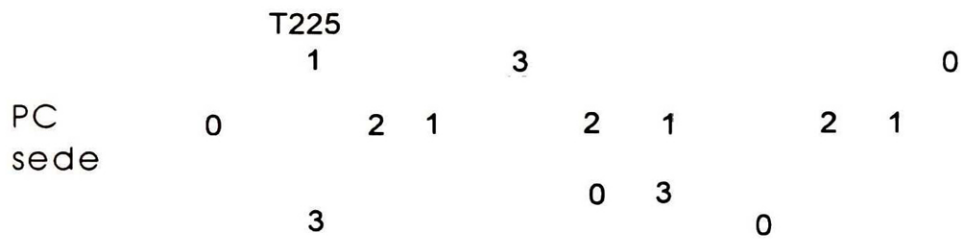


Figura 4.7: `Topología semicompleta 1`

Los pasos concretos y necesarios para fijar una configuración serán dados explícitamente durante la creación de una aplicación-ejemplo para los transputers en la siguiente sección, usando Occam para su codificación. Por esta razón, es indispensable definir los conceptos fundamentales de la programación en Occam, lo cual se lleva a cabo en el apéndice A.

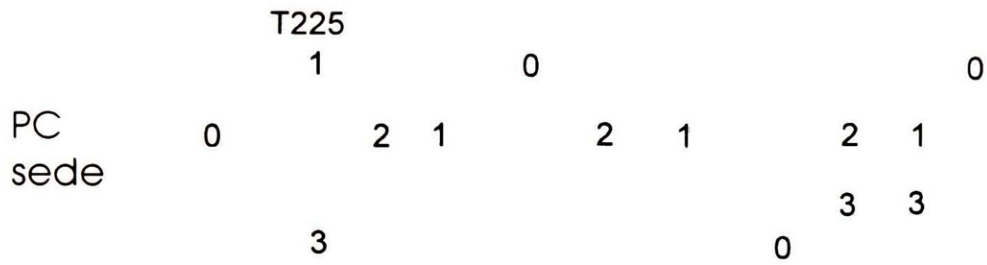


Figura 4.8: `Topología semicompleta 2`

4.3. Desarrollando una Aplicación-Ejemplo en Occam

Esta sección contiene un ejemplo muy sencillo de como paralelizar una aplicación sobre la topología estrella. En este caso, la aplicación consiste en realizar una operación aritmetica sobre cada procesador. Obviamente, esto no reduce el tiempo de cálculo, que es el principal objetivo del paralelismo, pero en cambio, por su simplicidad, muestra claramente cada una de las etapas desde su diseño hasta su compilación y ejecución. Esta es precisamente la finalidad del presente ejemplo.

La aplicación la formarán cuatro procesos: *controlador*, *suma*, *resta* y *producto*. El primero recibe dos enteros como entrada, los cuales se envían a cada uno de los otros tres procesos para que realizen la suma, la resta y el producto de estos enteros respectivamente, y a su vez, regresen el resultado. Por lo tanto, el proceso controlador necesita dos canales, uno para enviar y otro para recibir, para establecer la comunicación con cada uno de los otros procesos.

Además, como veremos a continuación, un quinto proceso se requiere para lograr la configuración *estrella*.

La topología *estrella* se consigue usando el transputer adyacente al raíz, en el *pipe* que se define por el archivo *hardwire* mediante el uso de *MMS*; esto se debe a que este último sólo tiene una liga libre, pues la liga 0 está conectada a la computadora sede, la liga 1 al transputer *T225* (el cual controla la conectividad de la red) y la liga 2 al otro transputer adyacente. Así, el segundo transputer en el *pipe* es conectado al cuarto mediante la liga 0 de ambos; esto se consigue a través del archivo *softwire* mediante *MMS*. La figura 4.9 lo muestra más claramente.

Como deseamos asignar cada par de canales que comunica a cada par de procesos sobre una liga y, a su vez, cada proceso asignarlo sobre cada transputer (es decir, el proceso central que forman todos los procesos con sus ligas deseamos

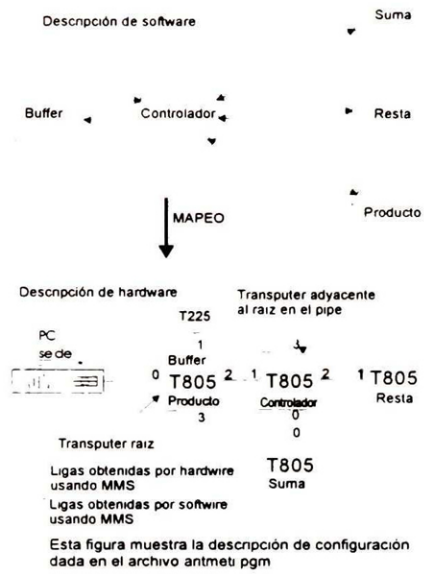


Figura 4.9: `Descripción de la configuración a través del archivo aritmeti.pgm`

asignarlo sobre el procesador central de la topología estrella). Esto es, el proceso *controlador* será asignado sobre el procesador adyacente (o central) y los procesos *suma, resta y producto* sobre los restantes procesadores, tal como se indica en la figura 4.9. Por lo tanto, el proceso *controlador*, que es el que establece la comunicación con la computadora sede (y mediante este, se establece la comunicación con el usuario) necesita de un proceso intermedio para re-transmitir estos mensajes. De aquí la necesidad de este nuevo proceso denominado *buffer*.

El código de este ejemplo, en su totalidad, se encuentra en el directorio

\jcg\maestria\tesis\occam\ejemplo

del disco auxilar de esta tesis. Este directorio contiene los siguientes archivos: *suma.occ, resta.occ, prod.occ, control.occ, buff.occ, aritmeti.pgm* e *info.txt*. Con la finalidad de lograr una mayor claridad, cada proceso fue ubicado en un archivo, y correspodiente al nombre de este. La descripción de configuración se encuentra en el archivo *aritmeti.pgm*.

Para construir la aplicación deben ser ejecutados los siguientes comandos[10]:

imakef aritmeti.btl /y
maker /f aritmeti.mak

Esto producirá varios archivos, entre los cuales está *aritmeti.btl* que es la aplicación que ya está lista para ser ejecutada sobre los transputers. Obviamente, su ejecución presupone que la topología de conexión de los transputers ha sido fijada a una de tipo *estrella*. Esta conexión puede ser fijada posicionándose en el directorio (desde DOS),

\S708

y ejecutando el siguiente comando

iserver /sb mms2.b4 estrella b008

entonces aparecerá un sistema de menus en el cual debe ser elegida la opción '*s*' correspondiente a la acción "*set C004*". Una vez fijada la configuración, elegir la opción '*q*' para salir del software *MMS*.

Recordemos que el archivo correspondiente a esta topología estrella, se encuentran en el directorio *c:\jcg\maestria\tesis\config* del disco auxiliar de esta tesis, el cual debe ser copiado al directorio *c:\s708* correspondiente al software *MMS*. El archivo *B008* correspondiente a las instrucciones hardwire se encuentra en el directorio *c:\s708*. Este archivo *B008* no debe ser cambiado mientras la tarjeta B008 no cambie. Es decir, mientras no le sean agregados nuevos componentes, tales como transputers, C004, T225, etc. Así, para las aplicaciones que siguen este archivo *B008* permanecerá sin cambios.

Los archivos *B008* y *estrella* se presentan en la sección 4.2.

El programa *aritmeti.btl* puede ser entonces ejecutado regresando al directorio del ejemplo y ejecutando a su vez el siguiente comando:

iserver /sb aritmeti.btl

[10]Si se desea hacer una depuración interactiva usando la herramienta INQUEST para Windows, se deberá ejecutar dicho comando quitando la opción */y*. Esto dará varios avisos referentes al ruteo virtual de los canales, que pueden ser ignorados sin ningún problema.

Para hacer una depuración interactiva de este ejemplo se sugiere realizar primero esta misma depuración con otro ejemplo dado en el capítulo 5 de [24]. En este capítulo se describe, a través de un ejemplo, cada una de las operaciones que pueden ser realizadas para esta depuración.

Análogamente, si se desea realizar una depuración postmortem, se sugiere seguir paso a paso las indicaciones dadas en el capítulo 6 para el mismo ejemplo del capítulo 5, de esta misma referencia.

Estos capítulos 5 y 6 de [24], son guías tutoriales muy breves para realizar ambas formas de depuración. Su lectura y su puesta en práctica, son necesarios para aprender a realizar estas tareas.

La codificación completa se incluye a continuación.

Archivo: **ARITMETI.PGM**

```
- Archivo de configuración para usar una topología "ESTRELLA"
- con 4 procesadores de 4 MBytes de Memoria. Este archivo
- contiene una sección donde se asignan explícitamente las unidades
- lógicas a las unidades físicas, definida por la instrucción MAP


- DESCRIPCIÓN DE HARDWARE

- Estas instrucciones definen la constante K y M
VAL K IS 1024:
VAL M IS K * K:
- Declaración de los procesadores
NODE p.ctrl.fis, p.suma.fis, p.resta.fis, p.prod.fis:
- Declaración de las ligas de comunicación usadas
- por los procesadores.
ARC hostlink, ctrl.prod.link1, ctrl.prod.link2:
ARC ctrl.suma.link, ctrl.resta.link:
- Define la red de procesadores llamada red.cuatro.procesadores
NETWORK red.cuatro.procesadores
   - DO agrupa instruciones SET y CONNECT
   DO
      - SET define las características de los procesadores
      SET p.ctrl.fis (type, memsize := "T805", 4*M)
      SET p.prod.fis (type, memsize := "T805", 4*M)
```

```
SET p.suma.fis (type, memsize := "T805", 4*M)
SET p.resta.fis (type, memsize := "T805", 4*M)

- CONNECT define la conexión entre los procesadores a través
- de sus ligas usando WITH
CONNECT p.prod.fis[link][0] TO HOST WITH hostlink
CONNECT p.prod.fis[link][2] TO p.ctrl.fis[link][1] WITH ctrl.prod.link1
CONNECT p.ctrl.fis[link][2] TO p.suma.fis[link][1] WITH ctrl.suma.link
CONNECT p.ctrl.fis[link][0] TO p.resta.fis[link][0] WITH ctrl.resta.link
CONNECT p.ctrl.fis[link][3] TO p.prod.fis[link][3] WITH ctrl.prod.link2


- DESCRIPCIÓN DE SOFTWARE

- La biblioteca hostio.inc contiene el protocolo SP (Server Protocol)
#INCLUDE "hostio.inc"

- Los archivos "*.c9h" dados en las instrucciones USE contienen a
- cada uno de los procesos que serán usados en esta aplicación.
- La extensión "*.c9h" se usada como notación para crear el archivo
- "*.mak" a través de la herramienta 'imakef' para crear automáticamente
- el ejecutable. Cada letra o número de la extensión "*.c9h" tiene un·
- significado:
- la "c" significa que se crearán las instrucciones necesarias para obtener
- unidades encadenadas a partir del código fuente "*.occ"
- el "9" significa que éstas serán para el tipo de transputer T805
- la "h" significa su ejecución será en modo HALT (en caso de error
- se detiene todo el sistema). El significado de la notación para esta
- extensión y el uso de la herramienta 'imakef' puede ser consultado
- en [15] cap. 11.
#USE "control.c9h"
#USE "suma.c9h"
#USE "resta.c9h"
#USE "prod.c9h"
#USE "buff.c9h"
```

```
- Declaración de procesos
NODE p.suma.log, p.resta.log, p.prod.log, p.ctrl.log:

- Declaración de canales
CHAN OF SP fs, ts, fs0, ts0:
CHAN OF INT ctrl.a.sum, sum.a.ctrl:
CHAN OF INT ctrl.a.rest, rest.a.ctrl:
CHAN OF INT ctrl.a.prod, prod.a.ctrl:

- CONFIG define la sección de descripción de software.
CONFIG
  PAR
    - PROCESSOR asocia un proceso a un procesador
    PROCESSOR p.suma.log
      suma ( ctrl.a.sum, sum.a.ctrl )
    PROCESSOR p.resta.log
      resta ( ctrl.a.rest, rest.a.ctrl )
    PROCESSOR p.prod.log
      - Sobre este procesador se ejecutan, concurrentemente, los
      - procesos "buffer" y "producto".
      PAR
        buffer( fs, ts, ts0, fs0 )
        producto ( ctrl.a.prod, prod.a.ctrl )
    PROCESSOR p.ctrl.log
      controlador ( fs0, ts0, ctrl.a.sum, sum.a.ctrl,
                    ctrl.a.rest, rest.a.ctrl, ctrl.a.prod, prod.a.ctrl )
  :


- ASIGNACIÓN de procesos y canales a procesadores y ligas resp.
- MAPPING define la sección de asignación
MAPPING
  DO
    - Asignación de procesos a procesadores.
    - MAP en combinación con ONTO asigna procesos
    - a procesadores. También se usa para asignar un
    - máximo de dos canales sobre una liga.
```

```
MAP p.suma.log ONTO p.suma.fis
MAP p.resta.log ONTO p.resta.fis
MAP p.prod.log ONTO p.prod.fis
MAP p.ctrl.log ONTO p.ctrl.fis
- Asignación de canales a ligas.
MAP fs, ts ONTO hostlink
MAP fs0, ts0 ONTO ctrl.prod.link1
MAP ctrl.a.prod, prod.a.ctrl ONTO ctrl.prod.link2
MAP ctrl.a.sum, sum.a.ctrl ONTO ctrl.suma.link
MAP ctrl.a.rest, rest.a.ctrl ONTO ctrl.resta.link
```

Archivo: **CONTROL.OCC**

```
- Este archivo contiene el proceso "controlador", el cuál recibe los
- valores de las variables 'n1' y 'n2' desde la PC sede
- para realizar las operaciones aritméticas suma, resta y producto
- contenidas en sus respectivos procesos. Luego, n1 y n2 se envían
- a cada uno de estos procesos; para finalmente recibir el resultado
- de las operaciones y enviarlos a la PC sede.

- Uso de la biblioteca 'hostio' que contiene varios procesos de
- entrada/salida, constantes y variables globales.
#INCLUDE "hostio.inc"
#USE "hostio.lib" - Código de los procesos

- Definición del proceso controlador con dos canales para cada
- proceso suma, resta y producto; y con dos canales para
- establecer la comunicación con la PC sede.
PROC controlador ( CHAN OF SP fs, ts,
                   CHAN OF INT ctrl.a.sum, sum.a.ctrl,
                   ctrl.a.rest, rest.a.ctrl, ctrl.a.prod, prod.a.ctrl )
INT n1, n2, suma, prod, resta:
BOOL error:
SEQ
```

```
- se reciben los valores de 'n1' y 'n2'
  so.write.string despliega un mensaje en el monitor de
- la PC sede.
so.write.string( fs, ts, " Da el valor de n1: " )
- so.read.echo.int lee 'n1' desde el teclado de la PC sede
- y lo despliega en el monitor de la PC sede.
so.read.echo.int( fs, ts, n1, error )
- so.write.nl cambia de línea en el despliegue de la PC sede.
so.write.nl( fs, ts )
so.write.string( fs, ts," Da el valor de n2: " )
so.read.echo.int( fs, ts, n2, error )
so.write.nl( fs, ts )

- se envían 'n1' y 'n2' a cada proceso
ctrl.a.sum ! n1
ctrl.a.sum ! n2
ctrl.a.rest ! n1
ctrl.a.rest ! n2
ctrl.a.prod ! n1
ctrl.a.prod ! n2

- se reciben los resultados de las operaciones
sum.a.ctrl ? suma
rest.a.ctrl ? resta
prod.a.ctrl ? prod

- se envían estos resultados a la PC sede
so.write.string( fs, ts," la suma es: " )
so.write.int( fs, ts, suma, 0 )
so.write.nl( fs, ts )
so.write.string( fs, ts," la resta es: " )
so.write.int( fs, ts, resta, 0 )
so.write.nl( fs, ts )
so.write.string( fs, ts," el prod es: " )
so.write.int( fs, ts, prod, 0 )

- fin de este proceso
```

```
  so.exit( fs, ts, sps.success )
:
```

Archivo: **SUMA.OCC**

```
-Contiene el proceso 'suma'
PROC suma (CHAN OF INT ctrl.a.suma, suma.a.ctrl)
INT n1, n2, n3:
SEQ
  ctrl.a.suma ? n1
  ctrl.a.suma ? n2
  n3 := n1 + n2
  suma.a.ctrl ! n3
:
```

Archivo: **BUFF.OCC**

```
- Este archivo contiene el proceso auxiliar 'buffer', el cual
- sirve para retransmitir los mensajes a PC sede, cuando el
- proceso que estable la comunicacion con la PC sede, en este caso
- el proceso 'controlador', no se ejecuta en el proceso raíz,
- sino en uno adyacente a él. Pero este proceso 'buffer' si
- debe ser ejecutado sobre el transputer raíz.
- Esto se consigue mediante el uso del proceso 'so.buffer'
- definido en la bibioteca 'hostio'.

#INCLUDE "hostio.inc"
#USE "hostio.lib"
- buffer transmite información entre la PC sede y el resto de
- los procesos; y termina cuando recibe TRUE por el canal
- stopper.
PROC buffer( CHAN OF SP fs, ts, de.apli, a.apli )
- Canal usado para finalizar este proceso.
```

CHAN OF BOOL stopper:
- buffer nunca termina, pues nunca se le pasa nada
- por el canal stopper.
so.buffer(fs, ts, de.apli, a.apli, stopper)

4.4. Desarrollando una Aplicación-Ejemplo en C

En esta sección, será implementado en lenguaje C el mismo ejemplo que fue implementado en la sección 4.3 usando el compilador de Occam. Recordemos que la aplicación consiste en realizar una operación aritmética sobre cada uno de los procesadores (transputers) de una red de cuatro, en una topología estrella. El planteamiento del problema y su diseño ya fueron presentados en la sección anterior, por lo que sólo mostraremos como crear dicha aplicación (es decir, un programa ejecutable sobre los transputers), además de los comentarios necesarios para explicar los cambios hechos al ejemplo al pasar de Occam a C.

El código se encuentra en el directorio

\jcg\maestria\tesis\c\ejemplo

del disco auxilar de esta tesis. Este directorio contiene los siguientes archivos: *suma.c, suma.lnk, resta.c, resta.lnk, prod.c, prod.lnk, control.c, control.lnk, aritmeti.cfs e info.txt.* Notemos que el archivo correspondiente al código del proceso *buffer* del mismo ejemplo para *Occam* ya no se encuentra. Esto se debe a que en el lenguaje *C* no existe una función o proceso equivalente a este proceso *buffer*, el cual sirva para retransmitir los mensajes entre el transputer raíz y la computadora sede. Recordemos también, que en el ejemplo de *Occam* los procesos *controlador* y *producto* se asignan sobre el procesador adyacente y el procesador raíz respectivamente; ahora, en el caso de la implentación de este ejemplo en C, se asignan al contrario sobre estos mismos procesadores. Los canales se asignan, de manera virtual, sobre las ligas de los procesadores por el configurador a través de ligas virtuales. Así, el proceso *buffer* no es necesario, y es por tanto suprimido. El diseño de los procesos y sus canales tienen un pequeño cambio, el cuál se ve reflejado en la figura 4.10.

Otra cosa que podemos observar, es que se incluyen otros archivos "**.lnk*". Estos archivos se usan para hacer el encadenamiento indirecto; esto es, la herramienta "*link*" usa como parámetros una lista de todas las bibliotecas usadas por la aplicación, las cuales corresponden a los nombres de estos archivos dados en el archivo "**.lnk*". A diferencia de *Occam*, estos archivos son necesarios para crear el archivo "**.mak*" por medio de la herramienta "*imakef*" (en Occam, estos archivos "**.lnk*" se crean por "*imakef*"). Los archivos "**.lnk*" deben incluir, a su vez, uno de los siguientes dos archivos:

- *cstartup.lnk.* Cuando se desea crear un proceso que tenga comunicación

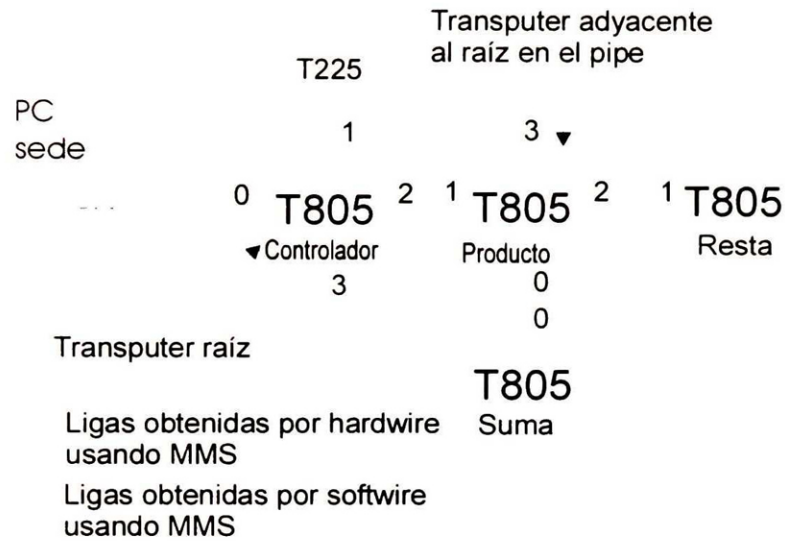


Figura 4.10: `Descripción de la configuración a través del archivo aritmeti.cfs.`

con la computadora sede (si se desea depurar el proceso, se debe usar en su lugar, el archivo *cdebug.lnk*). En el presente ejemplo, el archivo "*control.lnk*" correspondiente al proceso *controlador* debe usar este archivo, *cstartup.lnk*, en una de sus líneas, pues este proceso es el que establece la comunicación con la computadora sede.

- *cstartrd.lnk*. Se usa en caso contrario al anterior (usar *cdebugrd.lnk*, en su lugar para realizar una depuración con este proceso). Los archivos "**.lnk*" correspondientes a los procesos suma, resta y porducto deben usar este archivo, *cstartrd.lnk*, en una de sus líneas.

Información más detallada puede ser encontrada en [17] sección 11 del capítulo 3, y en [15] capítulo 11.

Para construir la aplicación deben ser ejecutados los siguientes comandos[11]:

[11] Si se desea hacer una depuración interactiva usando la herramienta INQUEST para Windows se deberá ejecutar dicho comando quitando la opción */y*. Esto producirá varios avisos referentes al ruteo virtual de los canales, que pueden ser ignorados sin ningún problema.

imakef aritmeti.btl /y /c
maker /f aritmeti.mak

Esto producirá varios archivos, entre los cuales está "*aritmeti.btl*" que es la aplicación que ya está lista para ser ejecutada sobre los transputers. El programa *aritmeti.btl* puede ser montado sobre la red de transputers y ejecutado mediante el siguiente comando:

iserver /sb aritmeti.btl

Obviamente, su ejecución presupone que la topología de conexión de los transputers ha sido fijada a una de tipo *estrella*, lo cual puede ser realizado usando el software MMS y el archivo B008 tal como se indicó en el ejemplo de Occam en la sección 4.3.

Los archivos *B008* y *estrella* se presentan en la sección 4.2.

La codificación completa se incluye a continuación.

Archivo: **ARITMETI.PGM**

```
/*Archivo de configuración para usar una topología "ESTRELLA"
  con 4 procesadores de 4 MBytes de Memoria.
*/


/* DESCRIPCIÓN DE HARDWARE */

/* Definición de los procesadores a usar */
/* T805 define un procesador de tipo T805
  con memoria memory ( de 4 megabytes ) */
T805 (memory = 4M) p_ctrl;
T805 (memory = 4M) p_suma;
T805 (memory = 4M) p_resta;
T805 (memory = 4M) p_prod;

/* Definiendo las conexiones entre ellos, a través de sus ligas */
/* 'host' es un tipo de liga predefinido por la herramienta de
```

```
   configuración ICCONF. host representa a la única liga
   de un procesador que comunica la red de procesadores con
   la computadora sede. connect une dos ligas de dos procesadores */
connect host to p_ctrl.link[0];
connect p_ctrl.link[2] to p_prod.link[1];
connect p_prod.link[2] to p_suma.link[1];
connect p_prod.link[0] to p_resta.link[0];
connect p_prod.link[3] to p_ctrl.link[3];


/* DESCRIPCIÓN DE SOFTWARE */

/* Definición de los procesos */
/* Nodo principal (establece la comunicación con la
   computadora sede) raíz */
/* process define un proceso con espacios de
   memoria stacksize y heapsize para el stack y el heap
   respectivamente, con una interfaz definida por interface
   y sus canales de entrada y salida: input y output resp.*/
process (stacksize = 100K, heapsize = 1000K,
          interface ( input host_in, output host_out,
          input de_suma, output a_suma,
          input de_resta, output a_resta,
          input de_prod. output a_prod)
          ) controlador;

/* Nodos esclavos */
process (stacksize = 5K, heapsize = 50K,
          interface (input de_ctrl, output a_ctrl) ) suma;
process (stacksize = 5K, heapsize = 50K,
          interface (input de_ctrl, output a_ctrl) ) resta;
process (stacksize = 5K, heapsize = 50K,
          interface (input de_ctrl, output a_ctrl) ) prod;

/* Declaración de los canales que comunican a la
   computadora sede con la red de procesadores */
input HostInput;
```

```
output HostOutput;

/* Conexión de canales entre los procesos */
/* connect establece la relación de conexión entre
   los canales de los diferentes procesos. */
connect HostInput to controlador.host_in;
connect HostOutput to controlador.host_out;
connect controlador.a_suma to suma.de_ctrl;
connect controlador.de_suma to suma.a_ctrl;
connect controlador.a_resta to resta.de_ctrl;
connect controlador.de_resta to resta.a_ctrl;
connect controlador.a_prod to prod.de_ctrl;
connect controlador.de_prod to prod.a_ctrl;

/* ASIGNACIÓN */
/* Asignación de procesos a procesadores y de canales a ligas */
/* place asigna uno o más procesos a un procesador y dos canales como
   máximo sobre cada una liga. */
place controlador on p_ctrl;
place suma on p_suma;
place resta on p_resta;
place prod on p_prod;
place HostInput on host;
place HostOutput on host;

/* Los archivos "*.c9h" dados en las instrucciones use contienen a
   cada uno de los procesos (funciones) que serán usados en esta
   aplicación.
   La extensión "*.c9h" se usa como notación para crear el archivo
   "*.mak" a través de la herramienta 'imakef' para crear automáticamente
   el ejecutable. Cada letra o número de la extensión "*.c9h" tiene un
   significado:
   la "c" significa que se crearán las instrucciones necesarias para obtener
   unidades encadenadas a partir del código fuente "*.c"
   el "9" significa que éstas serán para el tipo de transputer T805
   la "h" significa que su ejecución será en modo HALT (en caso de error
   se detiene todo el sistema). El significado de la notación para esta
```

```
   extensión y el uso de la herramienta 'imakef' puede ser consultado
   en [15] cap. 11.
*/
use "control.c9h" for controlador;
use "suma.c9h" for suma;
use "resta.c9h" for resta;
use "prod.c9h" for prod;
```

Archivo: **CONTROL.C**

```
/* Este archivo contiene el proceso "controlador", el cuál recibe los
   valores de las variables 'n1' y 'n2' desde la PC sede
   correspondientes a las operaciones aritméticas suma,
   resta y producto contenidas en sus respectivos procesos. Luego, envía
   estos operandos a cada proceso; para finalmente recibir el resultado
   de las operaciones y enviarlas a la computadora sede.
*/

#include <stdio.h> /* Uso de la biblioteca 'stdio.h' que contiene varios
                      procesos de entrada/salida, constantes y variables
                      globales. */
#include <channel.h> /* Biblioteca que contiene la función 'ChanInInt'
                        y 'ChanOutInt' entre otras. */
#include <misc.h> /* Biblioteca que contiene la función 'get_param' */

int main( )
{
   /* Declaración de canales */
   Channel * de_prod, * a_prod,
           * de_suma, * a_suma,
           * de_resta, * a_resta;

   /* Declaración de variables internas */
   int n1, n2, suma, prod, resta;
```

```
/* Obtención de los canales definidos para el proceso
   'controlador' a través del archivo "aritmeti.cfs".
   Los canales se obtienen en el mismo orden en el que
   se definen en la configuración.
        Los dos primeros canales se reservan por el
   compilador para este proceso 'controlador', pues es el
   que establece la comunicación con la computadora sede.
   La función void * get_param ( int n ) se usa para obtener
   los canales de un proceso, pues éstos no pueden ser pasados
   directamente como parámetros a la función main( )
   correspondiente al proceso, debido a que sus parámetros
   se definen por el lenguaje C, los cuales se adquieren
   desde la línea de comandos de la computadora sede.
*/
de_suma = (Channel *) get_param (3);
a_suma = (Channel *) get_param (4);
de_resta = (Channel *) get_param (5);
a_resta = (Channel *) get_param (6);
de_prod = (Channel *) get_param (7);
a_prod = (Channel *) get_param (8);

/* se reciben los valores de 'n1' y 'n2' */
printf(" Da el valor de n1: " );
scanf("%d", &n1 );
printf("\n Da el valor de n2: " );
scanf("%d", &n2 );

/* se envían 'n1' y 'n2' a cada proceso */
ChanOutInt( a_suma, n1 );
ChanOutInt( a_suma, n2 );
ChanOutInt( a_resta, n1 );
ChanOutInt( a_resta, n2 );
ChanOutInt( a_prod, n1 );
ChanOutInt( a_prod, n2 );

/* se reciben los resultados de las operaciones */
suma = ChanInInt( de_suma );
```

```
   resta = ChanInInt( de_resta );
   prod = ChanInInt( de_prod );

   /* se envían estos resultados a la computadora sede */
   printf(" la suma es: %d\n", suma );
   printf(" la resta es: %d\n", resta );
   printf(" el prod es: %d\n", prod );
   return 1;
}
```

Archivo: **CONTROL.LNK**

```
- Archivo de enlazamiento indirecto
control.t9h
#include cstartup.lnk -Elegir esta opción cuando no se desea depurar
-#include cdebug.lnk -y esta para depurar
```

Archivo: **SUMA.C**

```
/* Función 'main( )' correspondiente al proceso 'suma' */
#include <channel.h> /* Biblioteca que contiene la función 'ChanInInt'
                        y 'ChanOutInt' entre otras. */
#include <misc.h> /* Biblioteca que contiene la función 'get_param' */

int main( )
{
   int n1, n2;

   /* Declaración de canales */
   Channel * de_ctrl, * a_ctrl;

   de_ctrl = (Channel *) get_param (1);
   a_ctrl = (Channel *) get_param (2);
```

```
    n1 = ChanInInt( de_ctrl );
    n2 = ChanInInt( de_ctrl );
    ChanOutInt( a_ctrl, n1+n2 );
    return 1;
}
```

Archivo: **SUMA.LNK**

```
- Archivo de enlazamiento indirecto
suma.t9h
#include cstartrd.lnk -Elegir esta opción cuando no se desea depurar
-#include cdebugrd.lnk -y esta para depurar
```

5. Paralelización de las Aplicaciones de Interés

El presente capítulo es la parte central de este trabajo de tesis, pues en él se muestra todo el proceso que cualquier aplicación requiere para su paralelización, desde el planteamiento del problema y su diseño hasta la presentación de una parte de su codificación y de los resultados.

5.1. Optimización de Estructuras de Acero, Utilizando una Familia de Algoritmos Estocásticos en Paralelo

Este problema consiste en optimizar una armadura para soportar una nave industrial de acero, sujeta a cargas laterales y utilizando como información un catálogo de las secciones transversales de cada una de las barras que componen a la armadura; esto se consigue mediante el uso de una familia de algoritmos paramétricos que permiten realizar búsquedas estocásticas. Esta familia de algoritmos es una generalización de los algoritmos genéticos, las estrategias evolutivas y el recocido simulado.

5.1.1. Planteamiento del Problema

La estructura esta formada por un conjunto de barras articuladas, tal como se muestra en la figura 5.1.

Cada una de estas barras tienen ciertas características, tales como longitud, área transversal, material, resistencia, peso, etc., las cuales definen la resistencia, el costo y el peso total de la estructura. El problema es entonces encontrar una combinación óptima de estas barras de manera que la estructura soporte una determinada resistencia a costo y peso mínimos.

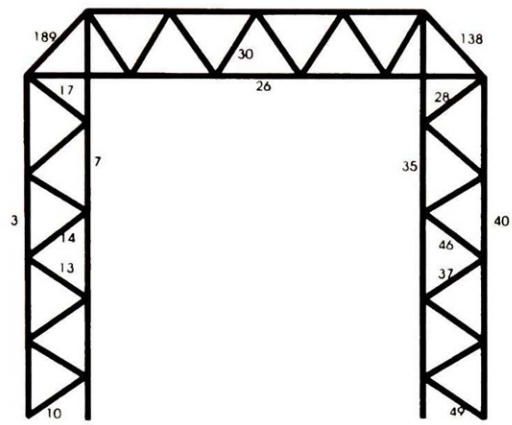
Para resolver este problema, se aplican algoritmos de búsqueda estocástica, que a continuación se describen.

El problema en forma más general que se trata de resolver es el siguiente:

Sea $\Omega = Q_1 \times ... \times Q_n$ un *espacio de estados* donde $Q_i, i = 1, ..., n$ son conjuntos de tamaño finito. Sea $U : \Omega \longmapsto \Re$ una *función de costo*. Se desea encontrar $x^* = (x^{*1}, ..., x^{*n}) \in \Omega$[12] que minimice U globalmente.

Los componentes básicos del algoritmo son una *población* X, formada por una serie ordenada de vectores $(x_1, ..., x_N), x_i \in \Omega$ sobre los que se aplican continuamente cuatro operadores básicos: *selección* (S_γ), *cruzamiento* (C_ξ), *mutación*

[12] A cada x^i de un individuo $x = (x^1, ..., x^n)$ se le llama *cromosoma*.



Estructura formada por barras articuladas. La numeración señala el tipo de barra desde un catálogo.

Figura 5.1: `Optimización de estructuras`

(M_μ) y *aceptación* (A_β). Cada uno de estos operadores está caracterizado por un parámetro que representa una elección particular entre la exploración y la explotación del espacio de búsqueda. La descripción de estos operadores se encuentra en el apéndice C.

Algoritmo General de Búsqueda Estocástica

El algoritmo general de búsqueda estocástica se define por el sistema dinámico

$$X^{(t+1)} = A_\beta(Y^{(t)}, M_\mu(C_\xi(Y^{(t)})))$$

con

$$Y^{(t)} = S_\gamma(X^{(t)})$$

donde en general, los parámetros β, μ, ξ, γ pueden variar en el tiempo, y la población inicial $X^{(0)}$ se selecciona aleatoriamente.

Ciertas combinaciones específicas de estos parámetros producen los siguientes casos:

1. Recocido simulado clásico con N puntos de inicio: $\xi = \gamma = 0$; β incrementándose.

2. Estrategia evolutiva: $\beta = \xi = 0$.

3. Algoritmo genético: $\beta = 0$.

4. Recocido simulado paralelo recombinado: $\gamma = 0$; β incrementándose.

Algunos resultados y la bondad de la aplicación de estos algoritmos, así como también sus propiedades de convergencia pueden ser consultados en [1] y [2] (dichos resultados se muestran en forma descriptiva, en la sección de resultados correspondiente a esta aplicación del presente trabajo).

La aplicación de estos algoritmos de búsqueda estocástica para resolver problemas de optimización es muy extensa, pues no pone restricciones a éstos, tal como lo hacen la mayoría de los métodos. Lo único que se tiene que hacer es codificar el dominio del problema; así como definir la función de costo.

La optimización de la estructura consiste en encontrar la sección transversal de cada elemento (de un conjunto discreto, por ejemplo, un catálogo), de tal forma que todos los esfuerzos que se encuentren sobre ella sean menores que un valor permisible, y el peso total de la estructura sea lo menor posible.

Es importante señalar que durante la ejecución del algoritmo, la forma física de la estructura siempre es la misma, es decir, la colocación (también la longitud) de cada barra permanece invariante. Lo que está continuamente cambiando en cada iteración es el área de la sección transversal de cada barra, lo cual define el peso de la barra, así como su resistencia.

Llamando $x(e)$ la entrada del catálogo selecionada para cada elemento e de la estructura, la función de costo a ser minimizada será:

$$U(x) = \sum_{e} \left[\rho_{x(e)} A_{x(e)} L_e + \lambda(\delta\sigma_e) \right]$$

donde ρ_x, A_x y L_e son el peso específico, la sección transversal de catálogo x_e y la longitud de e respectivamente; $\delta\sigma_e$ es la cantidad de esfuerzo que excede el valor máximo permisible y λ es factor de penalización para el esfuerzo en exceso.

5.1.2. Fase de Diseño

Cada elemento una población representa a una estructura, de forma tal que cada cromosoma de un elemento está asociado con el área transversal de una barra. El peso y la resistencia total de la estructura dependen de las áreas de las barras que componen la estructura.

Las operaciones genéticas (selección, cruzamiento, mutación y aceptación) pueden ser rápidamente ejecutados por una computadora. La consideración dominante respecto al tiempo de cómputo para estos algoritmos basados en métodos evolutivos, es la evaluación de la función de aptitud de cada elemento de la población. El cálculo de esta función consume mucho tiempo de cómputo por varias razones, incluyendo el tiempo requerido para decodificar e interpretar cada individuo de la población. El caso específico de la optimización de estructuras no es la excepción, pues su cálculo implica resolver un sistema de ecuaciones lineal de dimensión $N_b \times N_b$ (ver la referencia [2]), siendo N_b el número de barras diferentes. Afortunadamente, estas evaluaciones pueden ser realizadas independientemente para cada individuo. De aquí que podamos hacer uso de la estrategia *amo-esclavo* para explotar el paralelismo (ver la sección 4.1.1), en la cual un procesador *amo* (llamado CONTROL) realizará las operaciones genéticas y un proceso *esclavo* (llamado CFCRUDA) realizará el cálculo de la función de aptitud.

La paralelización se realizada, a partir del software existente para la implementación secuencial.

5.1.3. Descripción General de los Procesos

La estructura de los procesos y canales necesaria para esta aplicación, así como su asignación a la red de procesadores se muestran en la figura 5.2. CFCRUDA

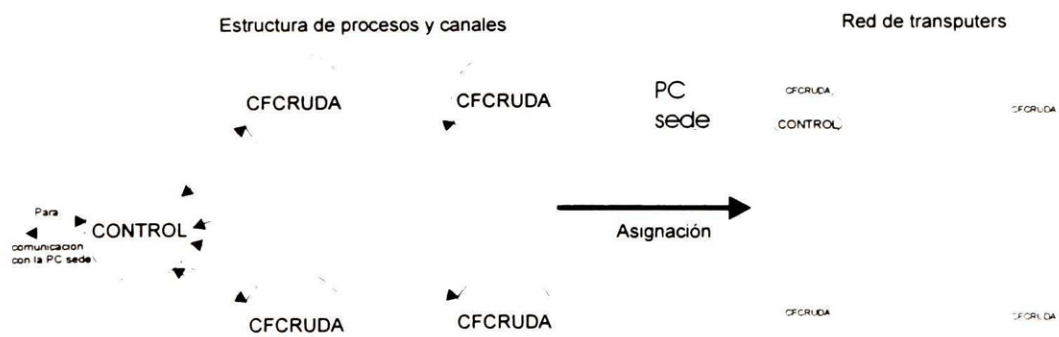


Figura 5.2: `Descripción de la configuración`

denota al proceso que representa al algoritmo que realiza el cálculo de la función de aptitud. Cada proceso CFCRUDA usa el mismo código.

CONTROL denota al proceso que establece la comunicación con la computadora sede y con cada uno de los procesos CFCRUDA. Además, este proceso realiza el algoritmo correspondiente a una instancia de la familia de algoritmos de búsqueda estocástica presentado en la subsección anterior, así como sus operaciones evolutivas.

Recordemos que la ejecución de cada uno de estos procesos se especifica en la sección de ASIGNACIÓN del archivo "*.cfs"

Por razones de espacio, no será presentado el código completo de cada uno de estos procesos. Aunque, éste se encuentra disponible, en su totalidad, en el disco auxiliar de esta tesis. Únicamente será mostrado el algoritmo correspondiente a cada proceso y el código más importante de éstos (en el apéndice); es decir, sólo aquel código relacionado exclusivamente con el paralelismo.

Respecto a la configuración de la tarjeta B008, recordemos que son necesarios dos archivos: *softwire* y *hardwire*. Este último, es el mismo para todas las aplicaciones, y en particular, para ésta. El contenido de este archivo y las instrucciones para fijar la configuración se encuentran en la sección 4.2. El contenido del archivo *softwire* debe ser el siguiente:

```
SOFTWIRE
PIPE 0
  SLOT 0, LINK 3 TO SLOT 3, LINK 0
  SLOT 1, LINK 0 TO SLOT 3, LINK 3
END
```

A continuación se muestra una breve descripción del funcionamiento de cada uno de los procesos:

Descripción General del Algoritmo Correspondiente al Proceso CONTROL.

La figura 5.3 muestra la estructura de archivos correspondiente al proceso CONTROL.

El código de *nube.c* contiene la familia algoritmos de búsqueda estocástica implementada, la cual es una herramienta general que puede ser usada para resolver una gran variedad de problemas. Lo único que se tiene que hacer es cambiar la forma de codificar el dominio del problema; así como redefinir la función de aptitud. La función *CalculaFCruda( )* se usa, en este archivo *nube.c*, para calcular la función de aptitud para cada elemento de la población en este problema

Figura 5.3: `Estructura de archivos que forman el proceso CONTROL.`

específico de optimización de estructuras. Así, para cada aplicación, la función *CalculaFCruda* debe ser implementada de acuerdo al problema.

El esquema del algoritmo general de búsqueda estocástica se muestra en la figura 5.4.

En la figura 5.4, *Reproduce( )* es una función que realiza las operaciones genéticas de: *selección, cruzamiento y mutación*, en este orden. Después de realizar el operador de mutación, se hace una llamada a la función *CalculaFCruda( )* para calcular la aptitud de cada elemento de la nueva población. La función *Reproduce( )* se encuentra definida en el archivo *nube.c*.

La operación genética de *aceptación* se realiza dentro de la función *main( )* asociada al proceso CONTROL del mismo archivo *nube.c*.

Así, el algoritmo general de búsqueda estocástica itera un número determinado de generaciones para ciertos valores de $(\gamma, \xi, \mu, \beta)$ mejorando la población en términos de la función de aptitud.

El proceso CONTROL realiza las siguientes operaciones:

A continuación se presenta un bosquejo de las principales operaciones, desde el punto de vista del paralelismo (es decir, la comunicacón de un proceso con otro),

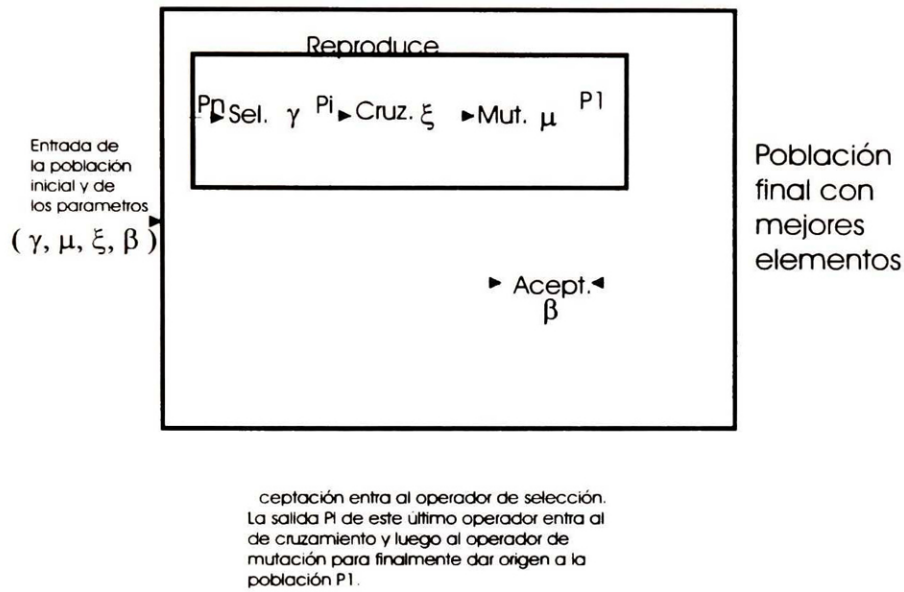


Figura 5.4: **Algoritmo general de búsqueda estocástica**

que realiza el proceso CONTROL.

- Declaración de 4 canales de entrada y 4 de salida; un par para cada proceso CFCRUDA.
- Se realiza la asignación de canales.
- Inicializa el proceso.
  - Se reserva memoria para cada variable y para los apuntadores a archivos de datos.
  - Algunas de estas variables se inicializan a partir del contenido de un archivo de datos.
  - Otras variables se inicializan directamente en el programa.
- El contenido de las variables se transmite por los canales de salida a cada proceso CFCRUDA.

- Se realiza una instancia de la familia de algoritmos para *ngen* generaciones, con ciertos valores de $(\gamma, \xi, \mu, \beta)$.
  - $\vdots$
  - Se envía una señal a cada proceso CFCRUDA que significa que recibirán información para comenzar a trabajar.
  - Se envían $\frac{nfam_i}{NP}$ individuos a cada proceso CFCRUDA (se envían sus cromosomas solamente, es decir, las áreas de la sección transversal de cada barra).
  - Se reciben los resultados de cada proceso CFCRUDA: *f*, *p*, *esfmax*, *esfmin*.
  - $\vdots$
- Se envía una señal de finalización a cada proceso CFCRUDA.
- Se libera la memoria reservada.

El código de las funciones principales de este proceso, desde el punto de vista del paralelismo, se presenta en el apéndice D.1.1.

Descripción General del Algoritmo Correspondiente al Proceso CFCRUDA.

La figura 5.5 muestra la estructura de archivos correspondiente al proceso CFCRUDA.

Este proceso realiza el cálculo de la función de aptidud, el cual implica resolver un sistema de ecuaciones lineal de dimensión $N_b/2$ (donde N_b es el número de barras).

Descripción del proceso CFCRUDA.

A continuación se presenta un bosquejo de las principales operaciones, desde el punto de vista del paralelismo, que realiza el proceso CFCRUDA.

- Declaración de un canal de entrada y otro de salida.
- Se realiza la asignación de canales.

Figura 5.5: `Estructura de archivos que forman el proceso CFCRUDA.`

- Inicializa el proceso reservando memoria para todas las variables.
- Se reciben los datos desde el proceso CONTROL, los cuales se asignan a sus respectivas variables.
- Se realiza el siguiente CICLO.
  - Se recibe una señal desde el proceso CFCRUDA. Si ésta es de terminar entonces finaliza el ciclo, en caso contrario continúa.
  - Se reciben $\frac{nfami}{NP}$ individuos (sus cromosomas solamente, es decir, las áreas de la sección transversal de cada barra).
  - Se calculan *f, p. esfmin, esfmax* para cada individuo.
  - Se envían *f, p, esfmin, esfmax* de cada individuo al proceso CONTROL.
- Se libera la memoria reservada.

El código de las funciones principales de este proceso, desde el punto de vista del paralelismo, se presenta en el apéndice D.1.2.

Esta aplicación, creada para su ejecución sobre los transputers, lee el archivo *coman.dat*, el cual contiene los nombres del archivo de datos (*nube.dat*) y del archivo donde serán almacenados los resultados del algoritmo (*nube.res*). Luego, cada cierto número de generaciones *igen* del algoritmo se exhiben en el monitor los resultados y se guardan las áreas de la estructura óptima en el archivo *barrasc.res*. El código completo de esta aplicación se encuentra en el disco auxiliar de esta tesis.

5.1.4. Resultados

Esta aplicación fue creada para ejecutarse en diferentes medios: en una red de cuatro transputers, en un solo transputer y sobre la PC. En cada uno de estos medios, la aplicación fue ejecutada hasta alcanzar 1000 generaciones, con la finalidad de comparar los tiempos de ejecución, que a continuación se muestran en la siguiente tabla:

Medio de ejecución\Tiempo	Tiempo en seg.	Respecto a un Transputer
Un transputer	275	1.00
Cuatro tranputers	69	3.99
PC 486 33MHz y 8MBytes	236	1.17
PC 586 90MHz y 8MBytes	41	6.71

Tabla 5.1: Comparación de tiempos de ejecución de una aplicación en diferentes medios

De esta tabla podemos calcular la razón de cambio de uno a cuatro transputers para esta aplicación,

$$S(N) = \frac{T_{Par}(1)}{T_{Par}(4)} = \frac{275}{69} = 3.99$$

de donde la eficiencia es,

$$\varepsilon = \frac{S(4)}{4} = 0.996$$

aproximadamente 1.0. Lo cual quiere decir, que el tiempo de cómputo de esta aplicación disminuye linealmente en función del número de procesadores y de manera óptima. Esto implica que la comunicación entre los procesos no adiciona tiempo de manera significativa.

Otra consecuencia importante que puede observarse de la tabla, es que la PC486 es aproximadamente equivalente a un transputer, y que la PC586 es aproximadamente equivalente a 7 transputers.

La figura 5.6 muestra la estructura articulada estudiada, la cual se encuentra sujeta a 11 cargas horizontales de 4,994 Kg cada una, y con condiciones de apoyo en su base. El material utilizado para todos las barras es acero, con un módulo elástico de $2.1 \times 10^{26} \mathrm{Kg/cm^2}$ y un máximo esfuerzo permisible de $3{,}500 \mathrm{Kg/cm^2}$ Las secciones transversales posibles van de 0.1 hasta 50.1 $\mathrm{cm^2}$, con incrementos de 0.5 $\mathrm{cm^2}$, de tal forma que el espacio de estados para cada elemento es de tamaño 100. Considerando que la estructura debe ser simétrica, se determinan solamente la mitad del número total de barras que forman la estructura.

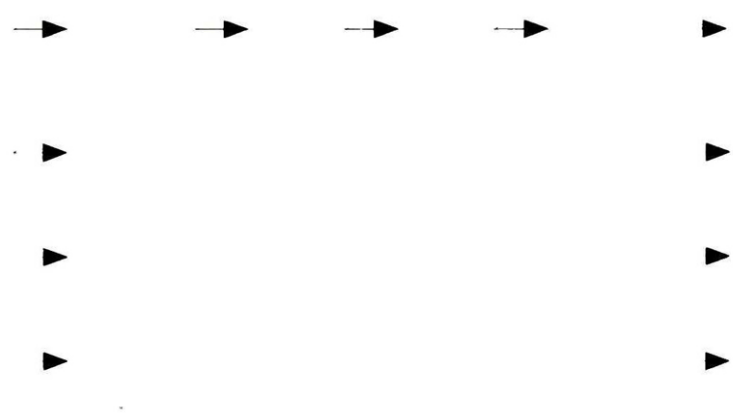


Figura 5.6: **Estructura de barras articuladas de una nave industrial.**

Se eligió una población de tamaño N = 20, la estructura fue optimizada para diferentes parámetros del algoritmo general de búsqueda estocástica. Los resultados obtenidos con los parámetros $\beta = 1.01, \mu = 0.1, \gamma = 1$ y $\xi = 0.8$ ofrecen la mejor relación entre velocidad de convergencia y calidad de la solución. Ver las referencias [1] y [2] para mayores detalles.

5.2. Un Filtro para la Eliminación de Ruido en Proceso de Imágenes, Usando un Autómata Celular Síncrono Determinístico.

5.2.1. Planteamiento del Problema

Sea g una imagen bidimensional[13]. Supongamos que g resulta de la aplicación de un operador H a la imagen original f. Supongamos además, que este operador H introduce un cierto ruido n a la imagen g, tal como se muestra en la figura 5.7.

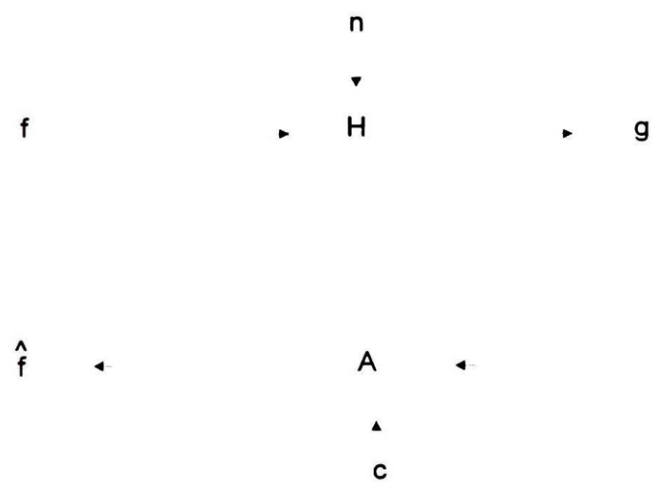


Figura 5.7: `Procesamiento de una imagen.`

El problema consiste en encontrar un algoritmo A para obtener una imagen $\hat{f}$ suficientemente parecida a f, la imagen a reconstruir.

De manera informal, podemos escribir este problema de la siguiente forma,

$$g = H(f) + n$$

Este es un problema "mal planteado" en el sentido de Hadamard [4] (pues no tiene una solución única). Para resolver este tipo de problemas es necesario introducir cierto conocimiento apriori "c" de la imagen , como pueden ser, el tipo de ruido, bordes, iluminación, textura, etc.

[13]Una *imagen bidimensional*, en el caso continuo, se define como una función, $f : \Omega \subset \Re^2 \longmapsto \Re$. En el caso discreto, el dominio de la función es una *retícula* $L \subset Z^2$ (subconjunto del plano de los enteros), tal que $L = \{(i,j)\,; i = 1 \ldots n, j = 1 \ldots n\}$

Sean $g_i, i \in L$, donde L es una retícula, las intensidades de la imagen g, y sean f_i las intensidades de la imagen a reconstruir. Una forma de resolver el problema es minimizando la siguiente expresión,

$$U(f) = \frac{1}{2}\left[\sum_i (f_i - g_i)^2 + \lambda \sum_{j \in V_i} (f_i - f_j)^2\right]$$

Para $\lambda > 0$ y $j \in V_i$, siendo V_i la vencidad[14], de primer orden, del i-ésimo elemento de L.

Donde el segundo término representa el conocimiento apriori. Para este algoritmo, en particular, el conocimiento apriori es asumir que f sea suave [3]; λ es un parámetro que controla el grado de suavidad en f.

Claramente se puede observar que U es una cuadrática no negativa, pues es una suma de términos no negativos.

Derivando U con respecto a f_i, se obtiene,

$$\frac{\partial U}{\partial f_i} = (f_i - g_i) + \lambda \sum_{j \in V_i} (f_i - f_j)$$

Igualando a cero y despejando f_i resulta,

$$f_i = \frac{g_i + \lambda \sum_{j \in V_i} f_j}{1 + \lambda |V_i|}$$

Si hacemos,

$$\begin{aligned} f_i^{(0)} &= g_i \\ f_i^{(t+1)} &= \frac{f_i^{(t)} + \lambda \sum_{j \in V_i} f_j^{(t)}}{1 + \lambda |V_i|} \end{aligned} \tag{5.1}$$

Para cada i, estas ecuaciones definen un autómata celular y su regla de evolución. Y para t fija, representan una imagen "suavizada". Así, para una t suficientemente grande, se espera que $f_i^{(t+1)}$ $(i \in L)$, sea un buen estimador de f (la imagen original).

[14] Una *vecindad* es $V_{ij} \subseteq L$, donde L es una retícula, y V_{ij} satisface las siguientes propiedades:
$(k,l) \in V_{ij} \iff (i,j) \in V_{kl}$
$(i,j) \notin V_{ij}$
$V_{ij} = \{(k,l) \in L : |i-k| = |j-l| = n_o\}$
donde n_o define el orden de la vecindad.

Se puede demostrar que $\nabla U = f^{(t+1)} - f^{(t)} \leq 0$ [5], ésto significa que f decrece en cada iteración, a menos que f sea un punto fijo de U, en cuyo caso sería el mínimo único que U tiene.

La demostración completa de la convergencia del algoritmo puede ser encontrada en [5].

5.2.2. Fase de Diseño

La implementación en paralelo de un algoritmo para eliminar ruido en una imagen, es un problema en el cuál se puede aplicar una de las estrategias para explotar el paralelismo: *la descomposición geométrica* (ver la sección 4.1.1).

Supongamos que V_i es la vencidad de primer orden del pixel i. El algoritmo definido por las ecuaciones 5.1 se aplica a cada pixel, como se muestra en la figura 5.8.

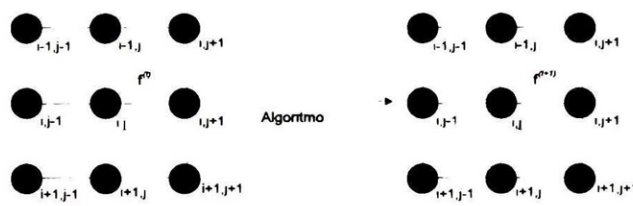


Figura 5.8: `Procesamiento de un pixel.`

El problema muestra claramente una simetría bidimensional, por lo que la descomposición geométrica puede ser realizada dividiendo la imagen en cuatro regiones, asignando cada una a un procesador, tal como se muestra en la figura 5.9.

Si analizamos el subproblema asignado a cada procesador, podemos observar que para calcular el nuevo estado de los pixeles contenidos en los bordes del subdominio, son necesarios los valores de los pixeles horizontales y verticales adyacentes, los cuales se encuentran en los procesadores vecinos. Esto lo podemos visualizar en la figura 5.10.

Antes de que los pixeles de los bordes actualicen sus intensidades de acuerdo al algoritmo, cada procesador debe intercambiar sus vecinos o regiones de translape. Por esta razón, la estructura de interconexión óptima entre los procesadores correspondiente a esta descomposición del problema, es la que se muestra en la

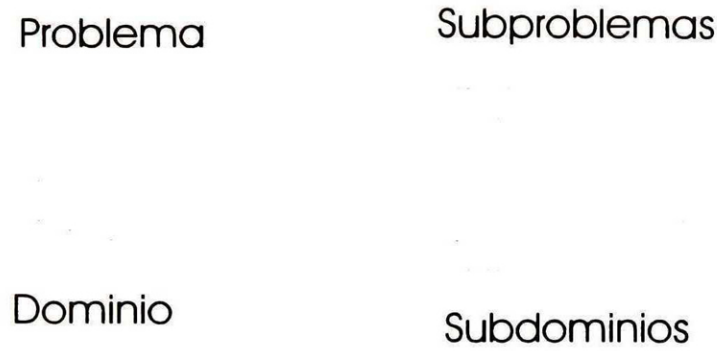


Figura 5.9: `Partición del dominio`

figura 5.11.

Esta topología no puede ser obtenida por los cuatro transputers incrustados en la tarjeta B008, debido a que el transputer raíz sólo tiene dos ligas libres, las cuales son insuficientes para obtener la topología requerida, pues cada procesador necesita de tres ligas. Ver sección 4.2.

En lugar de usar la topología requerida usaremos una muy similar a esta, la cual se muestra en la figura 5.12.

El uso de un sistema con una topología que no es exactamente igual a la requerida por el problema, conduce a la necesidad de comunicar dos procesadores no conectados directamente; lo cual, parece ser que disminuye el rendimiento del sistema, pero debido a que estos procesadores (los que no están conectados directamente) sólo transmiten un dato (el valor de la intensidad de un pixel) en cada iteración, resulta casi despreciable el tiempo necesario para transmitir este dato.

5.2.3. Descripción General de los Procesos

La estructura de los procesos y canales necesaria para esta aplicación, así como su asignación a la red de procesadores se muestran, de manera explícita, en la figura 5.13.

AC denota al proceso que representa al algoritmo principal, definido por las ecs. 5.1. Cada proceso AC usa el mismo código, y sólo se diferencian por un identificador *pid* asignado a cada uno de ellos.

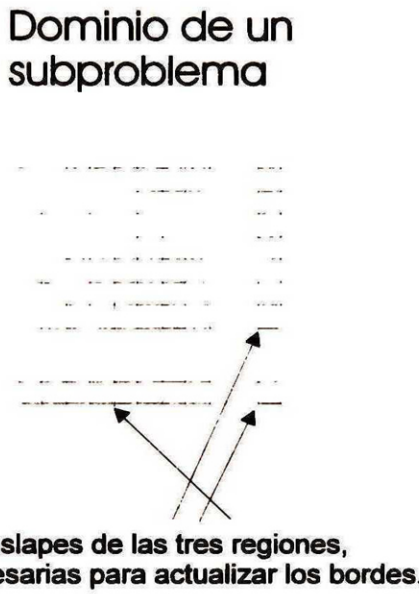


Figura 5.10: `Bordes de cada subdominio.`

CONTROL denota al proceso que establece la comunicación con la computadora sede y con cada uno de los procesos AC.

Recordemos que la ejecución de cada uno de estos procesos se especifica en la sección de ASIGNACIÓN del archivo "*.cfs".

Por razones de espacio, no será presentado el código completo de cada uno de estos procesos. Aunque, este se encuentra disponible, en su totalidad, en el disco auxiliar de esta tesis. Únicamente será mostrado el algoritmo correspondiente a cada proceso y el código más importante de éstos; es decir, sólo aquel código relacionado exclusivamente con el paralelismo.

Respecto a la configuración de la tarjeta B008, recordemos que son necesarios dos archivos: *softwire* y *hardwire*. Este último archivo, es el mismo para todas las aplicaciones, y en particular, para ésta. El contenido de este archivo y las instrucciones para fijar la configuración se encuentran en la sección 4.2. El contenido del archivo *softwire* debe ser el siguiente:

SOFTWIRE
PIPE 0

Figura 5.11: `Topología requerida.`

SLOT 0, LINK 3 TO SLOT 3, LINK 0
SLOT 1, LINK 0 TO SLOT 2, LINK 0
SLOT 2, LINK 3 TO SLOT 3, LINK 3
END

A continuación se muestra una breve descripción del funcionamiento de cada uno de los procesos:

Descripción General del Algoritmo Correspondiente al Proceso CONTROL.

- Envía datos a los procesos:
  - El identificador del proceso pid.
  - El número de procesadores NP.
  - El número máximo de iteraciones NMAX_ITER.
  - La frecuencia de almacenamiento de las imágenes en archivos INC_ITER.
  - El parámetro lambda del algoritmo.
  - El parámetro epsilon para comparar las imágenes.
  - Las dimensiones de la imagen nr y nc.

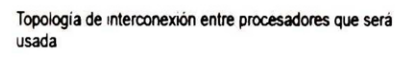


Figura 5.12: `Topología usada.`

- Leer la imagen a reconstruir $f^{(0)}$
- Envíar los bloques de la imagen $f^{(0)}$ a sus repectivos procesos.
- Realizar un CICLO con lo siguiente:
  - Recibir los bloques de cada uno de los procesos, cada cierto num. de iter. INC_ITER.
  - Integrar los bloques de la imagen procesada en una sola $f^{(t+1)}$.
  - Guardar la imagen $f^{(t+1)}$ en un archivo.
  - Comparar $f^{(t+1)}$ con $f^{(t)}$, Si $\left|f^{(t+1)} - f^{(t)}\right| \leq epsilon$ ó si $iteraciones \leq NMAX\_ITER$
    - Termina cada proceso y el ciclo.
    - En caso contrario, se continúa la ejecución del ciclo.

El código de las funciones principales de este proceso, desde el punto de vista del paralelismo, se presenta en el apéndice D.2.1.

Descripción General del Algoritmo Correspondiente al Proceso AC.

- Recibir datos del proceso CONTROL:

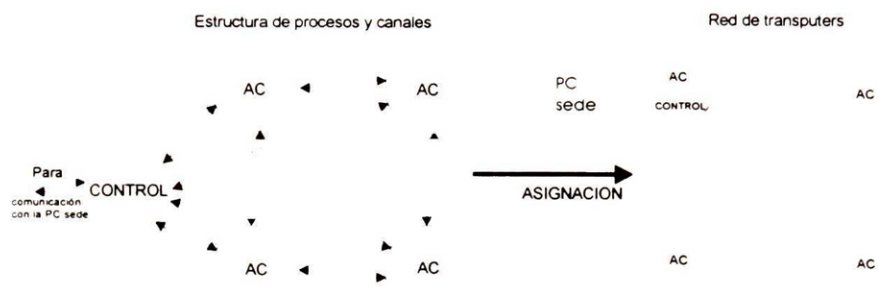


Figura 5.13: `Descripción de la configuración.`

- El identificador del proceso pid.
- El número de procesadores NP.
- El número máximo de iteraciones NMAX_ITER.
- La frecuencia de almacenamiento de las imágenes en archivos INC_ITER.
- El parámetro lambda del algoritmo.
- El parámetro epsilon para comparar las imágenes.
- Las dimensiones de la región de la imagen nr y nc.

- Recibir su correspondiente bloque de la imagen en $f^{(t)}$, con $t = 0$.
- Realizar el siguiente CICLO:
  - Intercambiar translapes con los procesos restantes.
  - Se aplican las ecs. 5.1 a cada pixel del bloque $f^{(t)}$, para formar $f^{(t+1)}$.
  - Se envía el bloque $f^{(t+1)}$ a CONTROL, cada cierto num. de iter. INC_ITER.
  - Comparar $f^{(t+1)}$ con $f^{(t)}$, Si $\left|f^{(t+1)} - f^{(t)}\right| \leq epsilon$ ó si $iteraciones \leq NMAX\_ITER$
    - Envía un mensaje a CONTROL y espera su respuesta para terminar o continuar.
    - En caso contrario, se continúa la ejecución del ciclo.

El código de las funciones principales de este proceso, desde el punto de vista del paralelismo, se presenta en el apéndice D.2.2.

Esta aplicación, creada para su ejecución sobre los transputers, recibe la imagen suave y con ruido como entrada a través de un archivo. Luego, cada cierto número de iteraciones del algoritmo, se crea una imagen correspondiente a esta etapa. Cada imagen se almacena en un archivo diferente sobre el sistema de archivos de la PC (el disco local). Estas imágenes se exhiben mediante una aplicación de Windows sobre la PC, la cual fue creada mediante el uso de una biblioteca de funciones llamada *shell.h*, que se usa para convertir aplicaciones de DOS a WINDOWS. El código de esta aplicación y de la biblioteca se encuentra en el disco auxiliar de esta tesis.

5.2.4. Resultados

Esta aplicación fue creada para su ejecución en una red de cuatro transputers, en un solo transputer y sobre la PC. En cada uno de estos medios, la aplicación fue ejecutada hasta alcanzar 1000 iteraciones, con la finalidad de comparar los tiempos de ejecución, que a continuación se presentan en la siguiente tabla:

Medio de ejecución\Tiempo	Tiempo en seg.	Respecto a un Transputer
Un transputer	112	1.00
Cuatro tranputers	34	3.29
PC 486 33MHz y 8MBytes	103	1.09
PC 586 90MHz y 8MBytes	16	7.00

Tabla 5.2: Comparación de tiempos de ejecución de una aplicación en diferentes medios

De esta tabla podemos calcular la razón de cambio de uno a cuatro transputers para esta aplicación,

$$S(N) = \frac{T_{Par}(1)}{T_{Par}(4)} = \frac{112}{34} = 3.29$$

de donde la eficiencia es,

$$\varepsilon = \frac{S(4)}{4} = 0.82$$

En este caso la eficiencia disminuye, respecto a la aplicación de optimización de estructuras, debido a que la sobrecarga en la comunicación no es nada despreciable, pues cada proceso envía las fronteras de su región a los procesos restantes. La

comunicación entre los procesos es la única causa de que la eficiencia sea menor que uno. En el caso de que el número de procesadores aumente, la eficiencia de los transputers para esta aplicación se acercará a uno, pues el tamaño de la región que a cada proceso le corresponde disminuirá y, por lo tanto, también disminuirá el tamaño de sus fronteras.

Esta aplicación es apropiada para una computadora masivamente paralela. Es decir, una computadora con tantos procesadores como pixeles sería lo óptimo para esta aplicación.

Otra consecuencia importante que puede observarse de la tabla, es que la PC486 es aproximadamente equivalente a un transputer, y que la PC586 es aproximadamente equivalente a siete transputers.

Considerando los costos de una PC586 y el de un transputer, podemos concluir que un sistema pentium es un sistema más *efectivo* que un sistema con cuatro transputers según la definición dada en la sección 4.1.4.

6. Conclusiones

La implementación de cada una de las aplicaciones (optimización de estructuras y proceso de imágenes) se llevó a cabo de manera exitosa. Una consecuencia importante es que un sistema de transputers efectivamente disminuye el tiempo de ejecución para la obtención de la solución del problema. Aunque considerando los precios de éstos y comparándolos con el precio medio de una pentium, los transputers no resultan muy recomendables. Esto se debe, principalmente, a que el mercado de los transputers es muy pequeño con respecto al de las pentium, lo cual se refleja en los precios.

En el caso concreto de un sistema con una tarjeta B008 y cuatro transputers, resulta menos económico que una pentium, y también menos potente, tal como se muestra en los resultados de cada aplicación. Sin embargo, los transputers no dejan de ser un buen equipo de laboratorio para iniciar este mundo nuevo del paralelismo. Personalmente, he adquirido las experiencias y los conocimientos más importantes del procesamiento en paralelo, necesarios para llevar a cabo un estudio más profundo de esta área de la computación, que tan importante será en el futuro por su mayor aplicación día a día en problemas prácticos.

Respecto a las herramientas disponibles para realizar aplicaciones sobre los transputers, se puede concluir que son actualmente bastante amigables, en el sentido de que ya no difultan tanto su desarrollo como algunas versiones anteriores; sobre todo el depurador con interfaz en Windows, que permite ejecutar paso a paso cada proceso y realizar todas las operaciones principales que cualquier depurador debe tener.

El presente trabajo tiene la característica de reunir la información más importante de cada una de estas herramientas y los problemas más comunes que se presentan al desarrollar las aplicaciones para los transputers. De manera que cualquier persona interesada en realizar aplicaciones sobre una red de transputers pueda hacerlo consultando este trabajo, y no tenga necesariamente que accesar toda la serie de manuales. Concretamente, a los usuarios de sistemas basados en transputers les será muy útil este reporte de tesis como una guía rápida para construir sus aplicaciones.

Próximamente, una de las aplicaciones de esta tesis será implementada sobre una red de ocho estaciones de trabajo SPARC usando PVM (Parallel Virtual Machine), aunque ya no como parte de este trabajo. Es de esperarse que los

resultados obtenidos serán mucho mejores que los obtenidos usando transputers, no sólo porque se tienen más unidades de procesamiento sino también porque su capacidad de cómputo es muy superior a éstos.

A. Programando en Occam

Occam es un lenguage de procesamiento en paralelo basado en el modelo *CSP* (communicating secuencial processes) de Hoare [9]. Cada programa *Occam* está estructurado en un conjunto de procesos concurrentes que se comunican a través de instrucciones de Entrada/Salida, sobre canales punto-a-punto, en forma síncrona. Cada proceso, a su vez, puede estar estructurado como un conjunto de procesos comunicándose concurrentemente. Por lo tanto, cada programa *Occam* tiene una estructura jerárquica-paralela.

La arquitectura del transputer está basada en los conceptos de concurrencia y comunicación de Occam, de manera que se puede establecer una correspondencia entre el hardware disponible y la estructura lógica de un programa.

A.1. Procesos Primitivos

En *Occam* todos los procesos están compuestos de procesos "atómicos" llamados procesos **PRIMITIVOS**:

1. **Asignación**. Asigna el valor de una expresión a una variable:

   variable : = expresión

2. **Entrada**. Espera hasta que un valor se recibe desde un canal, y lo asigna a una variable:

   canal ? variable

3. **Salida**. Transmite el valor de una expresión a un canal, y espera hasta que el valor se recibe por el canal de otro proceso:

   canal ! variable

4. Proceso **SKIP**, inicia y termina sin realizar ninguna acción.

5. Proceso **STOP**, inicia y, sin realizar ninguna acción, nunca termina. Este proceso puede ser pensado como un ciclo infinito pero sin instrucciones, y se presenta por ejemplo, cuando hay un abrazo mortal entre procesos (*deadlock*).

A.2. Uso de Canales

En *Occam* los canales pueden ser usados tanto para transmitir un solo tipo de valores como para varios tipos. Por ejemplo, en la siguiente declaración de canales,

```
CHAN OF INT ch1
CHAN OF ANY ch2
```

ch1 es un canal para transmitir sólo enteros y *ch2* es otro canal para transmitir datos de cualquier tipo definido.

A.3. Construcciones

Los procesos primitivos pueden ser combinados usando **construcciones**: secuenciales (*SEQ*), condicionales (*IF*), selecciones (*CASE*), ciclos (*WHILE*), de concurrencia (*PAR*), alternativas(*ALT*). Cada una de estas construcciones definen un nuevo proceso. Debido a que Occam fue diseñado de acuerdo al modelo CSP, en el que se tiene un conjunto de procesos organizados en forma jerárquica, a una instrucción de Occam se le llamará proceso primitivo o simplemente proceso.

A.3.1. Secuenciales

En la construcción *SEQ*, como en la programación convencional, cada componente se ejecuta una tras otra, en el orden en el que están escritas. Por ejemplo, la siguiente construcción

```
SEQ
  ch1 ? X
  X := X + 1
  ch2 ! X
```

ejecuta secuencialmente el bloque de intrucciones (procesos primitivos). Es decir, primero se recibe un valor por el canal *ch1* y se guarda en *X*, luego incrementa *X* y se se envía por el canal *ch2*.

A.3.2. Condicionales

Un condicional *IF* en Occam es muy diferente de la instrucción *if* convencional. El condicional de *Occam* contiene varios procesos los cuales se "vigilan" (*guarded*)[15] por una expresión condicional o booleana. Tales expresiones se evalúan en el orden dado hasta encontrar una VERDADERA, ejecutando el proceso correspondiente a esta expresión. Si ninguna de las expresiones es VERDADERA, la construcción se comporta como un proceso *STOP* (es decir, nunca termina).

En el siguiente ejemplo, la variable Y se incrementa si X es igual a cero; en otros casos, no se realiza ninguna acción (recordemos que la primitiva *SKIP* inicia y termina sin realizar ninguna acción).

```
IF X = 0
     Y := Y + 1
  X <> 0
     SKIP
```

A.3.3. Selecciones

La construcción *CASE* es muy similar a la construcción *IF*; ésta combina un número de procesos, cada uno de los cuales se asocian a los diferentes valores del *selector*. Entonces el proceso asociado al valor del *selector* elegido se ejecuta. Por ejemplo, en la construcción

```
CASE entero
  1,3,5,7,9
    par := FALSE
  0, 2,4,6,8
    par: = TRUE
```

[15]En adelante, cuando se hable de que un proceso es *vigilado* por una expresión booleana o condicional, significará que dicho proceso no podrá ser ejecutado mientras no sea verdadera tal expresión. A estas expresiones booleanas se les conoce como *guardias*, por su similitud a los guardias comunes (los guardias de los bancos, por ejemplo) que permiten o no que su correspondiente proceso vigilado sea ejecutado.

la variable *par* se fija a *TRUE* o *FALSE*[16], dependiendo del valor del selector (la variable *entero)*. La instrucción *ELSE* puede ser usada si el selector no se aparea a alguno de los casos anteriores (por ejemplo, si *entero* fuera negativo). Si la instrucción *ELSE* no se usa y no hay un apareamiento con el selector entonces la construcción *CASE* se comporta como la primitiva *STOP*.

A.3.4. Ciclos

La construcción *WHILE* se usa para repetir la ejecución de un proceso hasta que el valor asociado a una expresión booleana es *TRUE*. Por ejemplo, la siguiente construcción

```
WHILE X > 5
  X : = X   5
```

decrementa X en cinco hasta que su valor sea menor o igual a cinco.

A.3.5. Construcciones de concurrencia

En esta construcción *PAR*, los procesos componentes se ejecutan concurrentemente; todos inician al mismo tiempo y se pueden comunicar mediante operaciones de entrada/salida sobre canales previamente definidos. La construcción termina cuando todos los procesos componentes han terminado.

En el siguiente ejemplo, los dos procesos componentes se ejecutan en paralelo e intercambian el valor de una variable a través de los canales *ch1* y *ch2*, de entrada y salida respectivamente.

```
INT X, Y :                  - Declaración de las variables internas del
CHAN OF INT ch1, ch2 :-     proceso definido por la construcción PAR
PAR                         - Así son los comentarios en Occam
  SEQ
    X := 1
    ch1 ? X
  SEQ
    Y := -1
```

[16] *TRUE* y *FALSE* son los clásicos valores que se usan en varios lenguajes para definir el valor de una expresión booleana.

ch2 ! Y

Notar la sangría de dos espacios dada para cada construcción *SEQ*, lo cual significa que el proceso *PAR* consiste de dos procesos *SEQ*. El segundo nivel de sangría indica que cada proceso *SEQ* está compuesto de dos procesos primitivos, uno de asignación y otro de entrada/salida.

Cuando se tiene un bloque de más de un proceso primitivo (o un proceso en general, inclusive, otra construcción *PAR* que puede contener a su vez, construcciones *SEQ* y *PAR*) se debe especificar si se ejecutarán secuencialmente o en paralelo con las construcciones *SEQ* o *PAR* respectivamente. La sangría de dos espacios y la alineación de las componentes de cada construcción, definen su alcance y el ámbito de las variables declaradas. Esto equivale al *BEGIN* y al *END* del lenguaje de programación *PASCAL*.

Esta construcción *PAR* es el principio central de la programación en Occam. Por esta razón es de vital importancia enterderla muy bien.

Además, esta construcción puede ser usada dando dos niveles de prioridad a sus procesos. Es decir, la construcción *PRIPAR* sólo puede contener dos procesos, el primero de prioridad alta y el segundo de baja. Esta prioridad se da a los procesos por el orden en el que se escriben en la contrucción. Considerando la cola de procesos del transputer y su calendarización, un proceso de baja prioridad se ejecuta cuando no hay uno de prioridad alta. En el siguiente ejemplo, el proceso *p.high* se ejecuta siempre en preferencia a *p.low* (notemos el uso de procedimientos como procesos, en este caso sin parámetros, lo cuál será comentado posteriormente).

```
PRIPAR
  p.high( )
  p.low( )
```

A.3.6. Alternativas

Esta construcción *ALT*, al igual que *CASE*, contiene varios procesos, cada uno de los cuales se "vigila" por una primitiva de entrada o una expresión booleana o un combinación de ambos. Un canal de un proceso se dice que *está listo* para recibir si otro proceso está listo para enviar su salida sobre el mismo canal. Una construcción *ALT* selecciona uno y solamente uno de los procesos de quienes su

canal está listo, y ejecuta el proceso correspondiente. Si ninguno de los canales está listo, entonces la construcción espera hasta que al menos uno de ellos pueda ser seleccionado. Por ejemplo, la siguiente construcción

```
ALT
  ch1 ? X
    X := X + 1
  ch2 ? Y
    Y := Y + 1
```

es ejecutada esperando hasta que al menos uno de los dos canales *ch1*, *ch2* esté listo. Si solamente uno de ellos está listo, la entrada sobre aquel canal se realiza, y el proceso correspondiente se ejecuta. Si ambos canales están listos, solamente uno de las dos entradas se selecciona y se realiza, y el proceso correspondiente se ejecuta.

La construcción *ALT*, al igual que *PAR*, también da dos niveles de prioridad a sus procesos, a través de la construcción *PRIALT*. Así, esta construcción sólo puede tener dos procesos componentes, el primero de prioridad alta y el segundo de baja. Esta prioridad se da a los procesos por el orden en el que se escriben en la contrucción.

En *Occam* las construcciones *SEQ, IF, PAR,* y *ALT* pueden ser replicadas, es decir, pueden ejecutar repetidamente los procesos que éstas contienen. Por ejemplo, la construcción

```
SEQ i=0 FOR 3
  pr( )
```

es una notación compacta para

```
SEQ
  pr ( )
  pr( )
  pr( )
```

A.4. Temporizadores (timers)

Una forma de medir el tiempo y sincronizar los procesos es usando *timers*. En *Occam* es posible accesar el reloj del procesador mediante estos objetos llamados *timers* (tales como las variables y los canales) del tipo primitivo *TIMER*. El uso de un *timer* de entrada es similar a la operación de entrada de un canal. El valor regresado es un entero.

```
TIMER clock:
INT now:
  clock ? now:
```

Debido a que el tiempo en *Occam* es cíclico se ha introducido un operador de relación espacial (*AFTER*) para comparar los valores de reloj aún cuando sean de signo diferente. Si *tiempo1* y *tiempo2* son dos entradas del mismo *TIMER* la expresión:

```
tiempo2 AFTER tiempo1
```

es verdadera si el *tiempo2* corresponde a un tiempo posterior en el cual el *timer* ha asumido el valor *tiempo2*. Esta propiedad cíclica del *timer* de *occam* se ilustra en la siguiente figura,

Una vez que el timer ha alcanzado su máximo valor positivo, en el siguiente incremento toma el máximo valor negativo.

Una forma ligeramente diferente del *timer* de entrada, llamada entrada retardada, hace posible especificar un tiempo de retardo. Por ejemplo,

```
reloj ? AFTER ti
```

termina tan pronto el valor actual del reloj es posterior a *ti* Esta operación se usa para retrazar la ejecución de un proceso por un cierto intervalo de tiempo mediante la palabra adicional *PLUS* seguida por el tiempo de retrazo, como veremos en el siguiente ejemplo,

```
TIMER clock:
INT now:
SEQ
  clock ? now
```

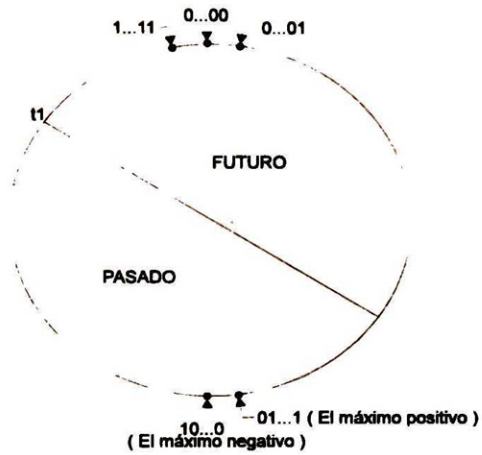


Figura A.1: **Manejo del reloj en occam**

clock ? AFTER now PLUS delay

Esta operación de entrada de tiempo con retardo puede ser usada como guardia en la construcción ALT, en exactamente la misma forma en la cuál se usan las entradas de canal.

O*ccam* proporciona la mayoría de los tipos de datos comunes *BOOL, BYTE, INT, REAL32, REAL64*, etc. Todos estos tipos de datos primitivos pueden ser usados como componentes de arreglos multi-dimensionales. Las variables y arreglos deben ser declarados antes de ser usados.

A continuación se presenta un ejemplo en el que se muestra la declaración y el uso de arreglos. El ejemplo consiste en leer una serie de datos desde un canal y almacenarlos en un arreglo.

```
PROC lee.canal( CHAN OF INT ent, []INT arreglo )
INT indice :
SEQ
  indice := 0
  WHILE indice < ( SIZE arreglo )
    SEQ
      indice := 0
```

```
      ent ? arreglo[ indice ]
      indice := indice + 1
:
```

El operador *PROC* define el procedimiento '*lee.canal*', mientras que el operador *SIZE* se usa para calcular la longitud del arreglo.

Cada declaración se aplica a los procesos en el mismo nivel de sangría. La expresiones pueden ser construidas por la combinación de variables y constantes por medio de un gran conjunto de operadores aritméticos, relacionales y lógicos. Cada expresión es de un tipo, y todas las variables y constantes en ésta deben ser de este tipo, debido a que la conversión implícita no se proporciona en *occam*. Y más aún, cada pareja de operandos debe ser situado entre paréntesis pues no hay ninguna regla de precedencia. Esto significa que una expresión que contiene las variables i (entero), x (real), que en un lenguaje común podrían ser escritas como

$$i := (x * x + 3.2)/2$$

en occam se escribiría

$$i := INT(((x * x) + 3.2(REAL32))/2.0(REAL32))$$

Ésta es una desventaja muy grave para los programadores, que en la siguiente versión de occam se planea eliminar, según se comenta en los manuales.

En *occam*, los procedimientos y funciones se definen como en la programación convencional, pero no pueden ser recursivamente llamados.

Por ejemplo, la declaración

```
PROC incrementar( INT a, VAL INT inc )
  a := a + inc
:
```

define el procedimiento *incrementar* con los parámetros formales a e inc pasados por referencia y por valor respectivamente, el cual incrementa a por el valor de inc. Similarmente, una función define un proceso. La declaración del proceso análogo al procedimiento anterior sería

```
INT FUNCTION incrementar( VAL INT a, inc )
  RESULT a + inc
```

Para más información sobre el lenguage Occam consultar [10] y [11].

Con lo visto hasta ahora de Occam, y lo referente a la configuración de la tarjeta B008 de la sección anterior, se tienen las bases y los conocimientos necesarios para desarrollar una primera aplicación, lo cual será realizado en la siguiente sección.

B. Programando en C

Los transputers han sido diseñados para soportar el procesamiento en paralelo. Esta arquitectura y su conjunto de instrucciones reflejan el modelo de programación CSP y facilitan la implementación de aplicaciones en lenguajes de alto nivel. INMOS C toma ventaja total de esta habilidad, proporcionando un medio de procesamiento en paralelo optimizado para el transputer, pero reteniendo todas las características de C.

La programación en paralelo en el lenguaje C, según el modelo de programación CSP, se consigue mediante las funciones de dos bibliotecas adicionales, *process.h y channel.h.* Además, el uso de semáforos se logra mediante las funciones de otra biblioteca, *semaphore.h*, la cual no forma parte del modelo de programación *CSP*, pero posiblita el desarrollo de programas paralelos en la manera tradicional.

Los procesos pueden ser ejecutados en uno o varios transputers y pueden comunicarse mediante canales implementados a través de ligas de transputer.

El programa compilado y los módulos de las bibliotecas usados se enlazan para formar un solo archivo llamado *unidad enlazada.* Cada unidad enlazada es un programa *main( )* en C, la cual puede ser asignada a un procesador para su ejecución mediante la descripción de configuración dada por el archivo "**.cfs*". Cada transputer puede ejecutar una o más unidades enlazadas en paralelo.

Las funciones de concurrencia tienen implementadas las siguientes operaciones de procesamiento en paralelo:

- Creación, comienzo y calendarización de procesos.
- Selección de una entrada de canal, a partir de varias entradas de canal que están listas para recibir.
- Comunicación a través de canales.
- Semáforos.

B.1. Creando y Manejando Procesos (*process.h*)

Cuando el proceso principal (el correspondiente a una unidad encadenada de un programa definido por la función *main( )*) comienza su ejecución en un procesador, otros procesos (también conocidos como *hilos*) pueden también comenzar

su ejecución sobre el mismo procesador si éstos forman parte del proceso principal. Tales procesos se ejecutan independientemente del proceso definido por la función *main( )* y de manera concurrente.

Se puede usar una función como punto de entrada a un proceso, dando a ésta un apuntador al proceso como su primer parámetro. Este apuntador al proceso puede ser usado por el programador, desde la función, para realizar operaciones con el proceso (por ejemplo, obtener la prioridad en la que se está ejecutando). El código de la función correspondiente a un proceso puede compartir datos externos con otros procesos de acuerdo a las reglas de ámbito de C.

La función en C, que se usa como proceso, debe ser definida en forma de interfaz como se muestra en el siguiente ejemplo,

```
void func( Process *p, int i, double d )
{
  /* cuerpo de la función */
}
```

La función *func* corresponde al código del proceso *p*.

Los procesos pueden ser creados por una de las funciones *ProcAlloc* o *ProcInit*; y ejecutados por una de las funciones *ProcRun*, *ProcPar*, o una de sus variantes.

A continuación se presenta un simple programa que ejecuta un proceso y espera a que este termine. El código completo de este ejemplo, que será usado en el resto de esta sección, puede ser encontrado en el directorio *\D7314A\examples\simple* correspondiente al sistema de herramientas *D7314A*.

```
/* Este programa comienza un proceso que imprime un mensaje */
#include <stdio.h>
#include <stdlib.h>
#include <process.h>

/* El proceso se declara como una función */
void hello_proc (Process * p)
{
  /* Cuando el apuntador, p, al proceso no se usa, el compilador
  envía un mensaje de error, el cual puede ser eliminado incluyendo
  la línea p = p. */
  p = p;
```

```
    printf ("Hello, world.\n");
    return;
}

int main (void)
{
    Process * hello;

    /* Crea un nuevo proceso */
    hello = ProcAlloc (hello_proc, 0, 0);
    /* El primer parámetro, hello_proc, es un apuntador a la función
       del proceso, el segundo parámetro, 0, es el número de
       parámetros de esta función, y el tercer parámetro,
       0, es el tamaño del stack, el default en este caso. */
    if (hello == NULL)
    {
       printf ("Could not allocate process.\n");
       exit (EXIT_FAILURE);
    }

    /* Comienza la ejecución del proceso */
    ProcRun (hello);

    /* Espera a que el proceso termine */
    ProcJoin (hello, NULL);

    /* Libera todo el espacio usado por el proceso */
    ProcAllocClean (hello);

    exit (EXIT_SUCCESS);
}
```

La siguiente función reserva memoria e inicializa procesos:

- **Process ProcAlloc(void (*func)(), int size, int nparam, ...)**

ProcAlloc reserva espacio de memoria sobre el área de memoria conocida como *heap* e inicializa la estructura *Process*.

ProcAlloc recibe un apuntador a la función, reserva memoria *stack* para el proceso, y fija los parámetros de la función de la cual su código se ejecuta cuando el proceso se inicia. *nparam* es el número de parámetros pasados a la función, excluyendo el apuntador al proceso.

El tamaño del *stack* se da por el parámetro *size*. Si *size* es cero el default es de 4K. *Procalloc* regresa un apuntador a la estructura del proceso (*Process* *).

Los procesos creados por *ProcAlloc* comparten el mismo espacio de datos global y por esta razón, pueden accesar las mismas variables estáticas y externas. El espacio de datos privado para un proceso debe ser reservado usando variables de tipo *auto*. Además, *ProcAlloc* usa las funciones *malloc* y *free* para reservar y liberar memoria *heap*.

La función

- **void ProcAllocClean(Process *p)**

es usada para liberar el espacio de trabajo y la memoria stack reservada y usada por un proceso.

Un proceso termina cuando la función del proceso termina. La función *ProcStop(void)* causará que el proceso espere indefinidamente y nunca termine. Un proceso no puede terminar el programa entero; solamente el proceso correspondiente a la función *main* puede hacer esto mediante la función *exit*.

Los procesos pueden ser inciados de manera síncrona o asíncrona. La inicialización asíncrona de procesos se realiza por la función *ProcJoin*, la cual no espera a que los procesos terminen de ejecutarse. En cambio, la función *ProcPar* ejecuta uno o más procesos en forma síncrona espera a que estos terminen de ejecutarse (el uso de las funciones *ProcJoin* y *ProcPar* y un ejemplo de éstas, será visto un poco más adelante).

Los procesos no pueden ser forzados a que terminen su ejecución, a excepción del proceso *main*.

B.1.1. Procesos Asíncronos

Los procesos asíncronos se inician por una de las siguientes funciones:

- **void ProcRun(Process *p)**
- **void ProcRunHigh(Process *p)**
- **void ProcRunLow(Process *p)**

El proceso p debe ser creado antes de ser ejecutado por alguna de estas funciones. El proceso p, a su vez puede ejecutar otros procesos. *ProcRun* inicia el proceso en la misma prioridad que el proceso desde el cual se ejecuta. *ProcRunHigh* inicia un proceso en prioridad alta y *ProcRunLow* en prioridad baja. Recordemos que un proceso de baja prioridad se ejecuta cuando no hay uno de prioridad alta en la cola de procesos del transputer.

Existen otras funciones que pueden ejecutar varios procesos en forma asíncrona, las cuales esperan hasta que todos los procesos asíncronos, ejecutados antes de la llamada a esta función, hayan terminado. Estas funciones son las siguientes:

- **int ProcJoin(Process * p1, ...)**
- **int ProcJoinList(Process ** plist)**

La función *ProcJoin* no regresa el control hasta que los procesos dados como parámetros hayan terminado. La función *ProcJoinList* hace lo mismo, pero con una lista de procesos que termina con un proceso *NULL*.

Ambas funciones regresan cero si fueron exitosas, y un número distinto de cero si la lista de procesos fue vacía.

B.1.2. Procesos Síncronos

Los procesos síncronos se inician por una de las siguientes funciones:

- **void ProcPar(Process * p1, ...)**
- **void ProcList(Process ** plist)**
- **void ProcPriPar(Process * phigh, Process * plow)**

Cada una de las dos primeras funciones inicia una lista de procesos y regresa el control hasta que todos los procesos hayan terminado. La función *ProcPriPar* inicia exactamente dos procesos, uno en alta y el otro en baja prioridad. Esta función regresa el control hasta ambos procesos hayan terminado.

A continuación se presenta un ejemplo del uso de dos procesos que imprimen un mensaje. La forma en la que se escribe el código no especifica el orden en el cual escriben los mensajes.

```
/* Este programa comienza dos procesos, cada uno de los cuales
imprime un mensaje. Los procesos se inician dos veces, la segunda
vez se intercambian de posición en la lista de parámetros de la función
ProcPar. Sin embargo, el orden a la salida estándar de estos
mensajes es indefinido. */
#include <stdio.h>
#include <stdlib.h>
#include <process.h>

/* El proceso con la primera palabra */
void hello_proc (Process * p)
{
  p = p;
  printf ("Hello ");
  return;
}

/* El proceso con la segunda palabra */
void world_proc (Process * p)
{
  p = p;
  printf ("world\n");
  return;
}

int main (void)
{
  Process * hello, * world;

  /* Crea dos procesos */
  hello = ProcAlloc (hello_proc, 0, 0);
  world = ProcAlloc (world_proc, 0, 0);
  if ((hello == NULL) || (world == NULL))
  {
    printf ("Could not allocate process.\n");
    exit (EXIT_FAILURE);
```

```
    }

    /* Ejecuta ambos procesos */
    ProcPar (hello, world, NULL);

    /* Intenta ejecutarlos en orden inverso al anterior */
    ProcPar (world, hello, NULL);

    /* Libera todo el espacio usado por cada proceso */
    ProcAllocClean (hello);
    ProcAllocClean (world);

    exit (EXIT_SUCCESS);
}
```

B.1.3. Sincronización entre Procesos

Los procesos pueden sincronizar sus acciones uno con otro por medio de canales o por medio de semáforos. Los semáforos es la forma tradicional de coordinar actividades en sistemas de memoria compartida. Los canales se usan para intercambiar datos entre dos procesos, pero pueden también ser usados para sincronizar procesos, debido a que los procesos no pueden continuar hasta que se realize la transferencia.

Nota: Los semáforos se implementan usando canales. Si la coordinación se requiere entre dos procesos, entonces un canal es la forma mas rápida de conseguir ésto.

B.2. Canales de Comunicación (*channel.h*)

Los canales son una forma de transmitir datos de un proceso a otro, o una forma de coordinar las acciones de los procesos. Si un proceso necesita esperar a que otro proceso alcance un estado particular, entonces el uso de canales es lo apropiado.

B.2.1. Inicialización de Canales

Los canales se declaran tal como se declaran las variables de otros tipos; pero antes, estos deben ser inicializados correctamente. Para los canales creados por la he-

rramienta de configuración *ICCONF* (es decir, los canales especificados mediante el archivo "**.cfs*") su inicialización se realiza automáticamente Para canales declarados en un programa C, esto puede ser realizado mediante las siguientes funciones:

- **Channel * ChanAlloc(void)**
- **void ChanInit(Channel *)**

La función *ChanAlloc* reserva espacio para un canal y lo inicializa. Si ya ha sido reservada memoria para el canal entonces éste puede ser inicializado por la función *ChanInit*.

B.2.2. Canales de Salida

Para transmitir datos a través de un canal, las siguientes funciones pueden ser usadas:

- **void ChanOut(Channel *c, void *cp, int n)**
- **void ChanOutChar(Channel *c, unsigned char ch)**
- **void ChanOutInt(Channel *c, int n)**
- **void DirectChanOut(Channel *c, void *cp, int count)**
- **void DirectChanOutChar(Channel *c, unsigned char ch)**
- **void DirectChanOutInt(Channel *c, int n)**

Las dos funciones *ChanOutChar* y *ChanOutInt* se usan para transmitir un *char* y un *int* respectivamente. La función *ChanOut* es una función general que envía n bytes de datos desde el apuntador *cp*, y puede ser usada para enviar cualquier tipo de datos.

Cuando se está operando sobre canales que comunican a procesos que se encuentran en el mismo procesador, se usan las funciones con nombres que comienzan con el prefijo *Direct*; estas funciones son equivalentes a las que no tienen el prefijo.

B.2.3. Canales de Entrada

Para recibir datos a través de un canal, las siguientes funciones pueden ser usadas:

- **void ChanIn(Channel *c, void *cp, int n)**
- **unsigned char ChanInChar(Channel *c)**
- **int ChanInInt(Channel *c)**
- **void DirectChanIn(Channel *c, void *cp, int count)**
- **unsigned char DirectChanInChar(Channel *c)**
- **int DirectChanInInt(Channel *c)**

Las dos funciones *ChanInChar* y *ChanInInt* se usan para recibir un *char* y un *int* respectivamente. La función *ChanIn* es una función general que recibe n bytes de datos desde el apuntador *cp*, y puede ser usada para recibir cualquier tipo de datos.

Cuando se está operando sobre canales que comunican a procesos, que se encuentran en el mismo procesador, se usan las funciones con nombres que comienzan con el prefijo *Direct*; estas funciones son equivalentes a las que no tienen el prefijo.

Ahora presentaremos el mismo programa del ejemplo anterior, pero haciendo esta vez, una sincronización de los procesos para que los mensajes que sean impresos en el orden requerido.

```
/* Este programa comienza dos procesos, cada uno de los cuales
imprime un mensaje. El primer proceso se sincroniza con el segundo,
después de que este imprime su mensaje.
Los procesos se ejecutan dos veces. La segunda vez se intercambian
de posición en la lista de parámetros de la función ProcPar. Sin embargo,
el orden a la salida estándar de estos mensajes se cambia. */
#include <stdio.h>
#include <stdlib.h>
#include <process.h>

/* El proceso con la primera palabra */
void hello_proc (Process * p, Channel * c)
```

```
{
  p = p;
  printf ("Hello ");
  ChanOutInt( c, 1 ); /* Envía el número después de imprimir */
  return;
}

/* El proceso con la segunda palabra */
void world_proc (Process * p, Channel * c)
{
  int k;
  p = p;
  k = ChanInInt( c ); /* Recibe el número antes de imprimir */
  printf ("world\n");
  return;
}

int main (void)
{
  Process * hello, * world;
  Channel sync;

  /* Se crea un canal de comunicación */
  sync = ChanAlloc( );

  /* Crea dos procesos */
  hello = ProcAlloc (hello_proc, 0, 1, sync);
  world = ProcAlloc (world_proc, 0, 1, sync);
  if ((hello == NULL) || (world == NULL))
  {
    printf ("Could not allocate process.\n");
    exit (EXIT_FAILURE);
  }

  /* Ejecuta ambos procesos */
  ProcPar (hello, world, NULL);
```

```
    /* Intenta ejecutarlos en orden inverso al anterior */
    ProcPar (world, hello, NULL);

    /* Libera todo el espacio usado por cada proceso */
    ProcAllocClean (hello);

    ProcAllocClean (world);

    exit (EXIT_SUCCESS);
}
```

B.2.4. Recibiendo Datos de Varios Canales

Existen casos en los cuales un proceso se encuentra esperando la entrada de varios canales, en donde se desea detectar cual dato llega primero. Esto puede ser conseguido usando una de las siguientes funciones:

- **int ProcAlt(Channel * c1, ...)**
- **int ProcAltList(Channel ** clist)**
- **int ProcSkipAlt(Channel * c1, ...)**
- **int ProcSkipAltList(Channel ** clist)**

Todas estas funciones toman una lista de apuntadores a canales. *ProcAlt* y *ProcAltList* regresan un índice (comenzando desde cero) del canal que está listo para transferir.

Las funciones *ProcAlt* y *ProcAltList* esperarán hasta que un canal esté listo.

Las variantes de las funciones *ProcSkipAlt* y *ProcSkipAltList* pueden ser usadas no solamente para realizar una lectura de un solo canal a partir de un conjunto de ellos, sino para detectar si algún canal está listo. Regresan "-1" cuando ninguno de los canales está listo para enviar; en caso contrario trabajan como las funciones *ProcAlt* y *ProcAltList* respectivamente.

A continuación se presenta un ejemplo que convierte a mayúscula el primer *char* recibido por un arreglo de canales, para luego, enviarlo por otro canal.

```
void upserve (Process * p, Channel * ins[], Channel * outs[])
{
  size_t i; char data;
  p = p;

  /* Ciclo infinito, este proceso nunca termina */
  for (;;)
  {

    /* Espera alguna entrada */
    i = ProcAltList (ins);

    /* Lee el dato y lo convierte a mayúscula */
    data = ChanInChar (ins[i]);
    data = toupper(data);

    /* Envía este dato al canal correspondiente */
    ChanOutChar (outs[i], data);
  }
}
```

B.3. Semáforos (*semaphor.h*)

En C, todos los procesos tienen acceso a las variables externas y estáticas que están dentro del ámbito. Para evitar conflictos a la hora de actualizar estas variables compartidas, los semáforos pueden ser usados para protejer su acceso. Típicamente, los semáforos se crean para permitir a no más de un proceso accesar un recurso en cualquier tiempo dado.

Los semáforos deben ser primero creados e inicializados mediante las siguientes funciones:

- **Semaphore * SemAlloc(int initvalue)**
- **void SemInit(Semaphore * sem, int initvalue)**
- **SEMAPHOREINIT (int initvalue)**

SemAlloc reserva memoria para el semáforo e inicializa este con el número de accesos simultaneos que serán permitidos. *SemInit* se usa cuando el semáforo se

crea de alguna otra forma, por ejemplo, declarando esta como variable, debiendo ser inicializada para su uso. El macro *SEMAPHOREINIT* puede ser usado para iniciar una variable estática de tipo *Semaphore*.

El acceso a un semáforo es controlado por las funciones:

- **void SemWait(Semaphore *)**
- **void SemSignal(Semaphore *)**

Cuando un proceso necesita adquirir un semáforo entonces *SemWait* se usa. Si el número de procesos que pueden usar el semáforo simultaneamente es alcanzado entonces el proceso solicitante esperará hasta que el número máximo se disminuya.

Cuando un proceso ha terminado de usar un recurso, la función *SemSignal* debe ser usada para liberar éste para otro proceso. Al primer proceso en espera del semáforo (si existe), le será concedido el acceso.

B.4. Usando el Reloj

Al igual que en Occam, en lenguaje C existen funciones para accesar cada reloj (uno de prioridad alta y el otro de baja), éstas son las siguientes:

- **int ProcTime(void)**
- **int ProcTimePlus(const int t1, const int t2)**
- **int ProcTimeMinus(const int t1, const int t2)**
- **int ProcTimeAfter(const int t1, const int t2)**

ProcTime lee el reloj correspondiente a la prioridad del proceso desde el cual se realiza su llamada. *ProcTimePlus* suma dos valores de reloj; *ProcTimeMinus* resta el segundo valor del primero y regresa su diferencia.

ProcTimeAfter determina si el primer tiempo es posterior al segundo. Un tiempo se considera como posterior a otro si no es mayor que la mitad de un ciclo completo del reloj. La mitad de un ciclo completo es de 2^{31} *ticks* sobre un transputer de *32* bits. La función regresa *1* si *t1* es posterior a *t2*. Recordemos que ésto se explicó en la sección A.4 y se ilustró con una figura.

B.4.1. Control de Procesos Usando el Reloj

Un proceso puede suspender su ejecución durante un cierto periodo de tiempo dado en número de ticks. Las funciones:

- **void ProcAfter(int t)**
- **void ProcWait(int t)**

Esperan un cierto periodo de tiempo y entonces regresan el control; si el tiempo t no es posterior al tiempo actual del reloj entonces la función ProcAfter no suspende la ejecución del proceso. La función *ProcWait* retraza la ejecución de un proceso por el número específico t de ticks.

También existen dos funciones que permiten a un proceso realizar una selección de entrada de canales por un tiempo específico. Si ningún canal llega a estar listo en el tiempo dado entonces la función termina y el proceso (desde el cual fue llamada la función) continúa su ejecución. Estas funciones son las siguientes:

- **int ProcTimeAlt(int time, Channel *c1, ...)**
- **int ProcTimeAltList(int time, Channel ** clist)**

El valor regresado por estas funciones es -1 si ningún canal llega a estar listo en el tiempo *time*. En caso contrario regresan el índice (comenzando desde 0 para el primer canal) del primer canal que llega a estar listo.

B.5. Otras Funciones

Si un proceso necesita conocer el nivel de prioridad en el que se está ejecutando, la siguiente función puede ser usada:

- **int ProcGetPriority(void)**

la cual regresa cero para prioridad alta y uno para baja.

Algunas veces, un proceso necesita forzosamente ceder el control del CPU para que otro proceso sea ejecutado; la siguiente función hace esto,

- **void ProcReschedule(void)**

El proceso que ejecuta esta función se agrega nuevamente a la lista de procesos calendarizados.

Información más detallada acerca del lenguage C para procesamiento en paralelo puede ser consultada en [17] cap. 5.

Con lo visto hasta ahora de procesamiento en paralelo usando C, podemos codificar en C, la misma aplicación-ejemplo dada en la sección 4.3. Esto se realizará en la siguiente sección.

C. Descripción de los Operadores Genéticos: *selección, cruzamiento, mutación y aceptación*

En este apéndice se describen brevemente cada uno de los operadores que forman el algoritmo general de búsqueda estocástica definido en la sección 5.1.1.

C.1. Selección

Para una función que mide la bondad de una solución de forma parametrizada $f_\gamma : \Omega \mapsto \Re$, la solución consiste en encontrar los elementos de una población determinada, con una probabilidad proporcionar al valor de dicha función.

La técnica de residuos estocásticos se elege por su simplicidad y eficiencia para colocar en forma determinística en la nueva población un número esperado de copias de cada elemento seleccionado (truncando a un número natural) y colocando las posiciones restantes en la forma clásica (estocástica).

El resultado es una familia de operadores de selección de un parámetro $S_\gamma : \Omega^N \mapsto \Omega^N$ definida por el siguiente algoritmo,

Operador $S_\gamma(X)$:

Calcula para cada elemento x_k de X, la adaptación
relativa f_k y truncando el número de ocurrencias
esperado será n_k:

$$f_k = \frac{f_\gamma(x_k)}{\sum_{j=1}^{N} f_\gamma(x_j)}$$

$$n_k = Int(N f_k)$$

Donde la función $Int()$ calcula la parte entera del argumento.
Hace $t = \sum_{k=1}^{N} n_k$ y construye el bloque $(y_1, \cdots, y_t)$
haciendo n_k copias para cada elemento x_k;
Elige $y_{t+1}, \cdots, y_N$ al azar de X con probabilidad de
seleccionar el elemento x_k igual a

$$\frac{Nf_k - n_k}{\sum_{j=1}^{N}(Nf_j - n_j)}$$

Hacer

$$S_\gamma(X) = (y_1, \cdots, y_N).$$

La familia de funciones de adaptación que mide la bondad f_γ está parametrizada por γ. Por ejemplo, si para todo $x \in \Omega$, la función de costo $U(x)$ es siempre positiva, tendremos:

$$f_\gamma(x) = 1 - \gamma(1 + U(x))$$

con $\gamma \in [0, 1]$.

Es posible usar también la forma exponencial como sigue:

$$f_\gamma(x) = \exp[-\gamma U(x)]$$

la cual, para valores grandes de γ genera un operador S_γ que selecciona el mejor elemento de una población y lo reproduce N veces.

C.2. Mutación

Podemos definir una familia de operadores de mutación con un parámetro M_μ : $\Omega^N \mapsto \Omega^N$ por medio del siguiente algoritmo,

Operador $M_\mu(X)$:

Para cada elemento x de la población hacer:
Construir un elemento y como sigue:
Para cada componente x^i de x hacer:
$y^i = r^i$ con prob. $p(\mu, x, X)$
$= x^i$con prob. $1 - p(\mu, x, X)$
donde r^i es el elemento selecionado de forma
aleatoria de Q_i con una probabilidad uniforme

Hacer

$$M_\mu(X) = (y_1, \cdots, y_N).$$

La probabilidad de mutación $p(\mu, x, X)$ puede ser uniforme (es decir, $p(\mu, x, X) = \mu$) o adaptativa según el valor de adaptación de x. La probabilidad adaptativa puede ser escrita como:

$$\begin{aligned} p(\mu, x, X) &= \mu \frac{f_{\max} - f(x)}{f_{\max} - \bar{f}} && \text{si } f(x) > \bar{f} \\ &= \mu && \text{si } f(x) \leq \bar{f} \end{aligned}$$

donde $f_{\max}$ y $\bar{f}$ son el máximo y el promedio de la función de adaptación respectivamente. Es importante para el mejor funcionamiento de este operador, que el mejor individuo de la población sea transmitido por el operador sin mutarlo. Para conseguirlo, se define $p(\mu, x, X) = 0$ si x es el mejor elemento en X e igual a μ en caso contrario.

C.3. Cruzamiento

Este operador realiza el intercambio de información mediante la división de dos elementos de la población en dos partes y concatenándolos de forma cruzada en cada uno de ellos. Con el fin de imponer diversidad, se utiliza cada elemento una sola vez en el operador de cruzamiento y se debe garantizar que no se pierda el mejor de todos ellos.

El resultado es una familia de operadores de cruzamiento de un parámetro $C_\xi : \Omega^N \mapsto \Omega^N$definida por el siguiente algoritmo,

Operador $C_\xi(X)$:

Mientras algunos elementos de X no hayan sido usados en el cruzamiento:
Seleccionar 2 dos elementos no usados x_j, x_k de X;
Con probabilidad $p(\xi, X, x_j, x_k)$, hacer un cruzamiento:
Determinar una posición r entre 1 y n
desde una distribución uniforme;
Construir

$$y_j = (x_j^1, \cdots, x_j^r, x_k^{r+1}, \cdots, x_k^n)$$

$$y_k = (x_k^1, \cdots, x_k^r, x_j^{r+1}, \cdots, x_j^n)$$

En caso contrario, colocar $y_j = x_j, \quad y_k = x_k$
Hacer

$$C_\xi(X) = (y_1, \cdots, y_N).$$

donde la probabilidad de cruzamiento $p(\xi, X, x, y)$ puede ser uniforme o variar en el tiempo, dependiendo del valor de adaptación del elemento implicado.

No obstante, se impondrá $p(\xi, X, x, y) = 0$ si x ó y es el mejor elemento de la población, y se hará igual a ξ en caso contrario.

C.4. Aceptación

Finalmente, para realizar la selección entre una población X y un candidato de la población mutada Y se aplica el criterio de aceptación de Metrópolis a cada elemento de X y Y. Esto define una familia de operadores de aceptación A_β : $\Omega^N \times \Omega^N \mapsto \Omega^N$, la cual es descrita en el siguiente algoritmo,

Operador $A_\beta(X, Y)$:

Para todos los elementos $x_k \in X, \quad y_k \in Y$ hacer

$$\Delta U = U(y_k) - U(x_k)$$

Si $\Delta U \leq 0$, hacer $u_k = y_k$;
Si no, hacer $u_k = y_k$ con prob. $\exp[-\beta \Delta U]$
x_k con prob. $1 - \exp[-\beta \Delta U]$

Hacer

$$A_\beta(X, Y) = (u_1, \cdots, u_N).$$

Los detalles completos de cada uno de estos operadores pueden ser consultados en [1] y [2].

D. Código Principal de las Aplicaciones

En este apéndice se muestra el código más importante de la implementación de cada una de las aplicaciones sobre transputers, que fueron tratadas en el capítulo 5.

D.1. Codificación del Problema de Optimización de Estructuras

D.1.1. Proceso CONTROL

El código del proceso *CONTROL* contiene la familia algoritmos de búsqueda estocástica implementada, la cual es una herramienta general que puede ser usada para resolver una gran variedad de problemas. Lo único que se tiene que hacer es cambiar la forma de codificar el dominio del problema; así como redefinir la función de aptitud, es decir, la función *CalculaFCruda( )*. Para cada aplicación, la función *CalculaFCruda* debe ser implementanda de acuerdo al problema.

A continuación se presenta el código de las funciones principales del proceso *CONTROL*, desde el punto de vista del paralelismo:

```
/*******************************************************/
/* Estructura que representa a los individuos de la población */
typedef struct
{
  unsigned char *c ; /* cromosomas: áreas de las barras */
  double f ; /* resistencia de la estructura */
  double p ; /* peso de la estructura */
  double esfmax ; /* esfuerzo máximo */
  double esfmin ; /* esfuerzo mínimo */
  int theta, n ; /* elementos usados para la selección */
}  GEN ;

/*********************************************************/
/* Declaración de variables globales */
GEN *p1, *pn, *pi ; /* apuntadores a tres poblaciones */
Channel * out_chan[4], * in_chan[4]; /* declaración de canales */


/*********************************************************/
/* Fragmentos de la función main( ) correspondiente al
  proceso CONTROL */
/*-------------------------------------------------------*/
/* Obtención de los canales para cada proceso */
for( j = 3, k = 0; j <= 9; j+=2, k++ )
{
```

```
  in_chan[k] = (Channel *) get_param (j);
  out_chan[k] = (Channel *) get_param (j+1);
}

/*************************************************/
/* Termina cada proceso CFCRUDA */
for( j = 0; j < NP; j++ )
  ChanOutChar( out_chan[ j ], (char)-1 );

/*-----------------------------------------------
/*************************************************/


/* Esta función envía los individuos a cada uno de los procesos
  CFCRUDA. Los resultados esfmax, esfmin, p, f se reciben también.
*/
void CalculaFCruda( void )
{
  /* Se envía el individuo "j" al proceso "j Mod NP" */
  for( i = 0; i < nfami; i+=NP )
  {
    for( j = i; j < i+NP; j++ )
    {
      ChanOutChar( out_chan[j-i], (char)1 );
      RecuperaAreas( j ); /* Decodifica al individuo 'j' en
                             el arreglo 'areas' */
      ChanOut( out_chan[j-i], &areas[1], nelem * sizeof( int ) );
    }
    for( j = i; j < i+NP; j++ ) Almacena( i, j );
  }
}

/*************************************************/
/* Recibe los resultados de los procesos CFCRUDA:
  esfmax, esfmin, p, f.
*/
void Almacena( int I, int j )
```

```
{
   ChanIn( in_chan[j-i], &p1[j].esfmax, sizeof( double ) );
   ChanIn( in_chan[j-i], &p1[j].esfmin, sizeof( double ) );
   ChanIn( in_chan[j-i], &p1[j].p, sizeof( double ) );
   ChanIn( in_chan[j-i], &p1[j].f, sizeof( double ) );
}

/*****************************************************/
/* Envía el contenido de las variables a cada proceso CFCRUDA */
void EnviaTodoInicia( void )
{
   int ielem;
   for( i = 0; i < NP; i++ )
   {
      ChanOutInt( out_chan[i], npnod );
      ChanOutInt( out_chan[i], nelem );
      :
      for( ielem=1;ielem <= nelem ; ielem++)
      {
         ChanOut( out_chan[i], &carga[ielem][1], nevab * sizeof(double) );
         ChanOut( out_chan[i], &fuemp[ielem][1], nevab * sizeof(double) );
      }
   }
}
```

D.1.2. Proceso CFCRUDA

A continuación se presenta el código de las funciones principales del proceso *CFCRUDA*, desde el punto de vista del paralelismo:

```
/*********************************************************/
/* declaración de canales */
Channel * out_chan, * in_chan;


/*********************************************************/
/* Fragmentos de la función main( ) correspondiente
```

```
  al proceso CFCRUDA */
/* ------------------------------------------------------------ */
  /* Obtencion de los canales */
  in_chan = (Channel *) get_param (1);
  out_chan = (Channel *) get_param (2);
  :
  RecibeTodoInicia();
  while( ChanInChar( in_chan ) >= 0 )
  {
    :
    /* calcula la solución al sistema de ecs. lineal */
    Solucion() ;
    :
    /* calcula los esfuerzos en los elementos */
    if(fuerc[1]< 1.e10) Tensiones() ;
    Almacena(esfmax,esfmin,sv,valor) ;
  }
  LiberaMemoria();
  return 1;
}

/* ------------------------------------------------------------ */
/**************************************************************/


/* Recibe el contenido de las variables del proceso CONTROL */
void RecibeTodoInicia(void)
{
  npnod = ChanInInt( in_chan );
  nelem = ChanInInt( in_chan );
  :
  PideMemoria(); /* reserva memoria para los apuntadores */
  :
  for( ielem=1;ielem <= nelem ; ielem++)
  {
```

```
        ChanIn( in_chan, &carga[ielem][1], nevab * sizeof( double ) );
        ChanIn( in_chan, &fuemp[ielem][1], nevab * sizeof( double ) );

}

/*************************************************/
/* Envía los resultados al proceso CONTROL:
   esfmax, esfmin, p, f.
*/
void Almacena(double esfmax, double esfmin, double v, double val )
{
   ChanOut( out_chan, &esfmax, sizeof( double ) );
   ChanOut( out_chan, &esfmin, sizeof( double ) );
   ChanOut( out_chan, &v, sizeof( double ) );
   ChanOut( out_chan, &val, sizeof( double ) );
}
/*************************************************/
```

D.2. Codificación del Filtro para Procesamiento de Imágenes

D.2.1. Proceso CONTROL

A continuación se presenta el código de las funciones principales del proceso *CONTROL*, desde el punto de vista del paralelismo:

```
/*************************************************
Envía los datos a cada proceso AC:
   el identificador de procesador 'pid',
   el num. de procesadores 'NP',
   el num. max. de iteraciones 'NMAX_ITER' del algoritmo,
   el parámetro 'lambda' del algoritmo,
   el parámetro 'epsilon' para comparar las imágenes.
*************************************************/
void EnviaDatos( void )
{
   for( i = 0; i < NP; i++ )
   {
```

```
    ChanOutInt( can_sal[i], i );
    ChanOutInt( can_sal[i], NP );
    ChanOutInt( can_sal[i], NMAX_ITER );
    ChanOutInt( can_sal[i], INC_ITER );
    ChanOut( can_sal[i], &lambda, sizeof(float) );
    ChanOut( can_sal[i], &epsilon, sizeof(float) );
    ChanOutInt( can_sal[i], nr );
    ChanOutInt( can_sal[i], nc );
  }
}

/*************************************************
Divide la imagen 'g' en bloques y los envía a cada proceso 'AC'
También envía las dimensiones de estos bloques.
*************************************************/
void EnviaBloques( void )
{
  /* Envia matriz */
  for( i = 0; i < NP; i++ )
  {
    ChanOutInt( can_sal[i], nr/2 );
    ChanOutInt( can_sal[i], nc/2 );
  }

  for( i = 0; i < nr/2; i++ ) /* Para proc1 */
    ChanOut( can_sal[0], g[i], nc * sizeof(float)/2 );

  for( i = 0; i < nr/2; i++ ) /* Para proc2*/
    ChanOut( can_sal[1], &g[i][nc/2], nc * sizeof(float)/2 );

  for( i = nr/2; i < nr; i++ ) /* Para proc3 */
    ChanOut( can_sal[2], g[i], nc * sizeof(float)/2 );

  for( i = nr/2; i < nr; i++ ) /* Para proc4 */
    ChanOut( can_sal[3], &g[i][nc/2], nc * sizeof(float)/2 );

}
```

```
/************************************************
Recibe bloques de cada proceso 'AC' y los acomoda en 'g' para
formar una imagen ya procesada.
************************************************/
void RecibeBloques( void )
{
  /* Recibe imagen */
  for( i = 0; i < nr/2: i++ ) /* De proc1 */
  ChanIn( can_ent[0], g[i], nc * sizeof(float)/2 );

  for( i = 0; i < nr/2: i++ ) /* De proc2*/
  ChanIn( can_ent[1], &g[i][nc/2], nc * sizeof(float)/2 );

  for( i = nr/2; i < nr; i++ ) /* De proc3 */
  ChanIn( can_ent[2], g[i], nc * sizeof(float)/2 );

  for( i = nr/2; i < nr; i++ ) /* De proc4 */
  ChanIn( can_ent[3], &g[i][nc/2], nc * sizeof(float)/2 );
}
/************************************************/
```

D.2.2. Proceso AC

A continuación se presenta el código de las funciones principales del proceso AC, desde el punto de vista del paralelismo:

```
/************************************************
Esta función recibe su bloque desde el proceso CONTROL
y lo acomoda en 'g', de manera que haya espacio para recibir
el translape.
************************************************/
void RecibeBloque( void )
{
  /* Las variables 'nr' y 'nc' son las dimensiones del bloque
     para este proceso 'pid' */
  nr = ChanInInt( ctrl_ent );
```

```
nc = ChanInInt( ctrl_ent );

/* Las variables 'm' y 'n' son las dimensiones del bloque
   incluyendo el translape, que serán usadas en la función
   'Cal_Sumaf_Nvecinos' */
m = nr + 1;
n = nc + 1;

/* Recibe el bloque de la matriz y lo acomoda en 'g' de
manera que haya espacio para recibir el translape.
'a1' y 'a2', 'b1' y 'b2' son los límites del bloque 'g'
en renglones y columnas ( visto g[ ][ ] como una matriz ),
que serán usados en la función 'Cal_Sumaf_Nvecinos' */
switch( pid )
{
   case 0: /* Proceso 0 */
      for( i = 0; i < nr; i++ )
      ChanIn( ctrl_ent, g[i], nc * sizeof(float) );
      a1 = 0; a2 = nr; b1 = 0; b2 = nc;
      break;

   case 1: /* Proceso 1 */
      for( i = 0; i < nr; i++ )
      ChanIn( ctrl_ent, &g[i][1], nc * sizeof(float) );
      a1 = 0; a2 = nr; b1 = 1; b2 = nc+1;
      break;

   case 2: /* Proceso 2 */
      for( i = 1; i <= nr; i++ )
      ChanIn( ctrl_ent, g[i], nc * sizeof(float) );
      a1 = 1; a2 = nr+1; b1 = 0; b2 = nc;
      break;

   case 3: /* Proceso 3 */
      for( i = 1; i <= nr; i++ )
      ChanIn( ctrl_ent, &g[i][1], nc * sizeof(float) );
      a1 = 1; a2 = nr+1; b1 = 1; b2 = nc+1; break;
```

```
  }
}
```

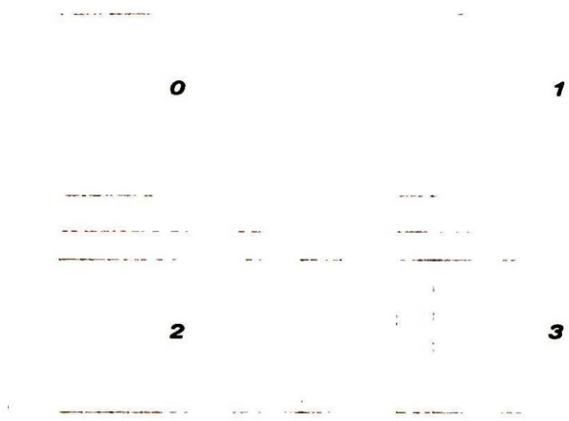


Figura D.1: **Auxiliar para la función** *EnviaRecibeTranslape*

```
/*************************************************
Esta función intercambia el translape de los bloques entre este
proceso 'pid' y los procesos restantes. La estrategia usada por
esta función para evitar los abrazos mortales en la comunicación
( deadlocks ) es la siguiente:
1) Recibir primero los translapes de los procesos anteriores
   ( si existen ) a este proceso 'pid'.
2) Enviar los translapes a sus respectivos procesos.
3) Recibe los translapes de los procesos restantes
   ( de los procesos posteriores ).
Observar la figura anterior para un rápido entendimiento.
/*************************************************
void EnviaRecibeTranslape( void )
{
  switch( pid )
  {
    case 0: /* Proceso 0 */
      for( i = 0; i < nr; i++ )
```

```
      ChanOut( can_sal[1], &g[i][nc-1], sizeof(float) );
    ChanOut( can_sal[2], g[nr-1], nc*sizeof(float) );
    ChanOut( can_sal[3], &g[nr-1][nc-1], sizeof(float) );
    for( i = 0; i < nr; i++ )
      ChanIn( can_ent[1], &g[i][nc], sizeof(float) );
    ChanIn( can_ent[2], g[nr], nc*sizeof(float) );
    ChanIn( can_ent[3], &g[nr][nc], sizeof(float) );
    break;

  case 1: /* Proceso 1 */
    for( i = 0; i < nr; i++ )
      ChanIn( can_ent[0], &g[i][0], sizeof(float) );
    for( i = 0; i < nr; i++ )
      ChanOut( can_sal[0], &g[i][1], sizeof(float) );
    ChanOut( can_sal[2], &g[nr-1][1], sizeof(float) );
    ChanOut( can_sal[3], &g[nr-1][1], nc*sizeof(float) );
    ChanIn( can_ent[2], &g[nr][0], sizeof(float) );
    ChanIn( can_ent[3], &g[nr][1], nc*sizeof(float) );
    break;

  case 2: /* Proceso 2 */
    ChanIn( can_ent[0], g[0], nc*sizeof(float) );
    ChanIn( can_ent[1], &g[0][nc], sizeof(float) );
    ChanOut( can_sal[0], g[1], nc*sizeof(float) );
    ChanOut( can_sal[1], &g[1][nc-1], sizeof(float) );
    for( i = 1; i <= nr; i++ )
      ChanOut( can_sal[3], &g[i][nc-1], sizeof(float) );
    for( i = 1; i <= nr; i++ )
      ChanIn( can_ent[3], &g[i][nc], sizeof(float) );
    break;

  case 3: /* Proceso 3 */
    ChanIn( can_ent[0], &g[0][0], sizeof(float) );
    ChanIn( can_ent[1], &g[0][1], nc*sizeof(float) );
    for( i = 1; i <= nr; i++ )
      ChanIn( can_ent[2], &g[i][0], sizeof(float) );
    ChanOut( can_sal[0], &g[1][1], sizeof(float) );
```

```
        ChanOut( can_sal[1], &g[1][1], nc*sizeof(float) );
        for( i = 1; i <= nr; i++ )
          ChanOut( can_sal[2], &g[i][1], sizeof(float) );
        break;
    }
}
```

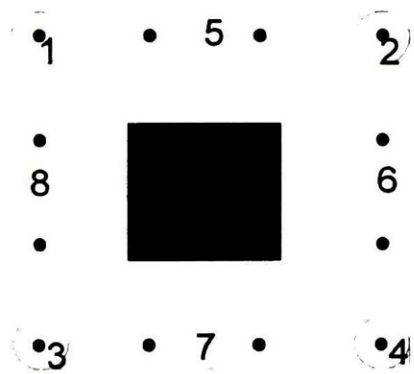


Figura D.2: **Auxiliar para la función** *Cal_Suma_f_Nvecinos*

```
/*******************************************
Esta función calcula la suma de las intensidades de los elementos
de la vecindad del elemento (i, j); además, calcula el número de
vecinos. Los resultados de estos cálculos se guardan en las
variables globales ( de este proceso ) 'suma_f' y 'n_vecinos'.
Para analizar rápidamente esta función, se sugiere ver la figura
D.2.
*******************************************/
void Cal_Sumaf_Nvecinos( int i, int j )
{
```

```
int caso;
/* Regiónes del bloque para calcular la suma 'suma_f'
   de las intensidades de la vecindad de cada pixel. */
/* Región 0: INTERIOR */
if( i != 0 && i != (n-1) && j != 0 && j != (m-1) ) caso = 0;
     /* Región 1: ESQ. SUP. IZQ. */
else if(i == 0 && j == 0) caso = 1;
          /* Región 2: ESQ. SUP. DER. */
     else if(i == 0 && j == (m-1)) caso = 2;
               /* Región 3: ESQ. INF. IZQ */
          else if(i == (n-1) && j == 0) caso = 3;
                    /* Región 4: ESQ. INF. DER. */
               else if(i == (n-1) && j == (m-1)) caso = 4;
                    /* Región 5: BORDE SUP. */
                 else if( i == 0 ) caso = 5;
                         /* Región 6: BORDE DER. */
                    else if( j == (m-1) ) caso = 6;
                              /* Región 7: BORDE INF. */
                         else if( i == (n-1) ) caso = 7;
                           /* Región 8: BORDE IZQ. */
                              else caso = 8;
switch( caso )
{
   case 0:
      suma_f = g[i-1][j] + g[i][j+1] + g[i+1][j] + g[i][j-1]
               g[i-1][j-1] + g[i-1][j+1] + g[i+1][j+1] + g[i+1][j-1];
      n_vecinos = 8;
      break;
   case 1:
      suma_f = g[i][j+1] + g[i+1][j] + g[i+1][j+1];
      n_vecinos = 3;
      break;
   case 2:
      suma_f = g[i+1][j] + g[i][j-1]+ g[i+1][j-1];
      n_vecinos = 3;
      break;
   case 3:
```

```
        suma_f = g[i-1][j] + g[i][j+1] + g[i-1][j+1];
        n_vecinos = 3;
        break;
      case 4:
        suma_f = g[i-1][j] + g[i][j-1] + g[i-1][j-1];
        n_vecinos = 3;
        break;
      case 5:
        suma_f = g[i][j+1] + g[i+1][j] + g[i][j-1] + g[i+1][j-1] + g[i+1][j+1];
        n_vecinos = 5;
        break;
      case 6:
        suma_f = g[i-1][j] + g[i+1][j] + g[i][j-1] + g[i-1][j-1] + g[i+1][j-1];
        n_vecinos = 5;
        break;
      case 7:
        suma_f = g[i-1][j] + g[i][j+1] + g[i][j-1] + g[i-1][j-1] + g[i-1][j+1];
        n_vecinos = 5;
        break;
      case 8:
        suma_f = g[i-1][j] + g[i][j+1] + g[i+1][j] + g[i-1][j+1] + g[i+1][j+1];
        n_vecinos = 5;
        break;
    }
}

/*************************************************/
/* El proceso 'pid' envía su bloque 'g' procesado ( sin translapes )
   al proceso CONTROL */
void EnviaBloque( void )
{
   switch( pid )
   {
     case 0: /* Proceso 0 */
        for( i = 0; i < nr; i++ )
        ChanOut( ctrl_sal, g[i], nc * sizeof(float) );
```

```
        break;

      case 1: /* Proceso 1 */
        for( i = 0; i < nr; i++ )
        ChanOut( ctrl_sal, &g[i][1], nc * sizeof(float) );
        break;

      case 2: /* Proceso 2 */
        for( i = 1; i <= nr; i++ )
        ChanOut( ctrl_sal, g[i], nc * sizeof(float) );
        break;

      case 3: /* Proceso 3 */
        for( i = 1; i <= nr; i++ )
        ChanOut( ctrl_sal, &g[i][1], nc * sizeof(float) );
        break;
    }
}
/*************************************************/
```

Referencias

[1] Botello Salvador, Marroquín J.L., Van Horebeek Johan. CIMAT 1995. *A Family of Parallel Stochastic Search Algorithms.*

[2] Botello Salvador, Marroquín J.L, Van Horebeek Johan. CIMAT 1995. *Una familia de Algoritmos Estocásticos en Paralelo Aplicados a la Optimización de Estructuras de Acero.*

[3] Blake A. and Zisseman A., *Visual Reconstruction*, MIT Press, Cambridge, Mass. (1987).

[4] Marroquín, S. Mitter and T. Poggio, *Probabilistic solution of ill-posed problems in computational vision*, J. Am. Stat. Asoc., 82, 76-89.

[5] Marroquín, J. L. Probabilistic solution of inverse problems. AI-TR860. Mass. (1985).

[6] D.E. Goldberg, *Genetic algorithms in search, optimization and machine learning*, Addison Wesley, Reading, MA (1989).

[7] McIntosh V. Harold. 1990. *Linear Cellular Automata.*

[8] Carlini and Villano. 1991. *Transputers and Parallel Architectures.* Ellis Horwood Limited.

[9] Hoare C. A. R. 1985. *Communicating Sequential Proccesses.* Prentice Hall.

[10] Hoare C.A.R. 1988. *Occam 2 Reference Manual.* Prentice Hall.

[11] Pountain Dick and May David. 1988. *A Tutorial Introduction to Occam Programming.* BSP Professional Books.

[12] Hoare C.A.R. 1988. *Occam Programming Manual.* Prentice Hall.

[13] Hoare C.A.R. 1988. *Programming in Occam.* Prentice Hall.

[14] INMOS, SGS-THOMSON Microelectronics Group. 1992. *IMS D7314A IBM 386 PC ANSI C Toolset Delivered Manual.*

[15] INMOS, SGS-THOMSON Microelectronics Group. 1992. *ANSI C Toolset Reference Manual.*

[16] INMOS, SGS-THOMSON Microelectronics Group. 1992. *ANSI C Languaje and Libraries Reference Manual.*

[17] INMOS, SGS-THOMSON Microelectronics Group. 1992. *ANSI C Toolset User Guide.*

[18] INMOS, SGS-THOMSON Microelectronics Group. 1992. *IMS D7314A IBM 386 PC Occam2 Toolset Delivered Manual.*

[19] INMOS, SGS-THOMSON Microelectronics Group. 1992. *Occam Toolset Reference Manual.*

[20] INMOS, SGS-THOMSON Microelectronics Group. 1992. *Occam Languaje and Libraries Reference Manual.*

[21] INMOS, SGS-THOMSON Microelectronics Group. 1992. *Occam Toolset User Guide.*

[22] INMOS, SGS-THOMSON Microelectronics Group. 1990. *B008 Support Software.*

[23] INMOS, SGS-THOMSON Microelectronics Group. 1990. *IMS B008 User Guide and Reference Manual.*

[24] INMOS, SGS-THOMSON Microelectronics Group. 1994. *IMS D7300A PC INQUEST Delivered Manual.*

[25] Hwang and Briggs. 1984. *Computer Architecture and Parallel Processing.* McGraw-Hill.

[26] Sincovec, Keyes, Leuze, Petzoid and Reed. 1993. *Parallel Processing for Scientific Computing.* Vol.1, 2. SIAM.

Los abajo firmantes, integrantes de jurado para el examen de grado que sustentará el **Lic. Julio Cesar Gallardo**, declaramos que hemos revisado la tesis titulada:

"APLICACIONES DE PROCESAMIENTO EN PARALELO USANDO TRANSPUTERS"

consideramos que cumple con los requisitos para obtener el grado de Maestro en Ciencias, con especialidad en Ingeniería Eléctrica.

Atentamente

Dr. Guillermo Morales Luna

M. en C. Feliú Davino Sagols Troncoso

Dr. José Luis Marroquín Zaleta

Dr. Salvador Botello Rionda

AUTOR GALLARDO, J.C.

TITULO APLICACIONES DE PROCESAMIENTO EN PARALELO USANDO TRANSPUTERS

CLASIF. XM96.29 RGTRO. BI 14954

NOMBRE DEL LECTOR	FECHA PREST.	FECHA DEVOL.
Martinez Guzman Roberto Carlos	26/03/01	27/3/01
Mireya Tovar Vidal	26/06/02	16/7/02